University of California
Santa Barbara

Identifying and Preventing Large-scale Internet Abuse

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Kevin Borgolte

Committee in Charge:

Professor Christopher Kruegel, Co-Chair

Professor Giovanni Vigna, Co-Chair

Professor Ben Zhao
The University of Chicago

September 2018

The dissertation of Kevin Borgolte is approved.

Professor Ben Zhao

Professor Giovanni Vigna, Committee Co-Chair

Professor Christopher Kruegel, Committee Co-Chair

June 2018

Identifying and Preventing Large-scale Internet Abuse

Acknowledgements

Without question, this dissertation would not have seen the light of day without the incredible support of numerous exceptional people along the way.

I thank my two advisors, Christopher Kruegel and Giovanni Vigna, for their inspiration, guidance and insight, for believing in me pursuing my own research ideas, and for encouraging me every step of the way. I also thank Mathias Payer, for being an unofficial academic mentor, and for supporting me in my academic development and career. From a research perspective alone, this dissertation stands on the shoulders of my co-authors. Most notably, Shuang Hao and Tobias Fiebig: Thank you for .⋆, for as if I would try to list all reasons individually, I would fail and forget at least one.

Of course, my acknowledgement would be incomplete without thanking my work family, the SecLab, current or former, postdoc, graduate student, or intern. Thank you for being there for me the last few years, the unforgettable deadlines, the nerve-wracking iCTFs and CTFs, all the joy and pleasantries that come with family, but also all the drama that we had, created, or otherwise were involved in, and which made us all grow closer together in the end. A special thank you goes to my divorced work wife, Francesco Disperati, for the "the good stuff," without whom overhauling the SecLab's entire infrastructure would have delayed my dissertation by a while, and the countless other things. I also want to thank Tim Robinson, for being the SecLab's financial and administrative wizard, and for taking care of everything that a horde of (wild) graduate students threw his way, however outlandish it.

I also cannot imagine that this dissertation would have been possible without the love and unwavering support of my family: my mother, Claudia, my late grandfather, Bernard, and, *alphabetically* (because otherwise I might not hear the end of it), Björn, Hansi, Heinz-Willi, Kolja, Monika, Sigrid, and Tina. Thank you!

And, finally, Martina: Thank you for your love, encouragement, joining me in Santa Barbara, putting up with me working endlessly, and, well, simply said, everything.

Curriculum Vitæ

Kevin Borgolte

Education

Doctor of Philosophy (Ph.D.), Computer Science
University of California, Santa Barbara, United States of America
September 2013 – September 2018 (Expected)

- Outstanding Graduate Student Award, May 2016
- Ph.D. Student Progress Award, March 2014

Master of Science ETH (M.Sc. ETH), Computer Science
Swiss Federal Institute of Technology Zurich, Switzerland
Eidgenössische Technische Hochschule Zürich, Schweiz
September 2010 – September 2012

Bachelor of Science (B.Sc.), Computer Science
University of Bonn, Germany
Rheinische Friedrich-Wilhelms-Universität Bonn, Deutschland
September 2007 – September 2010

Research Experience

Graduate Student Researcher
Computer Security Group, Department of Computer Science
University of California, Santa Barbara
September 2013 – September 2018

Research Scholar
Computer Security Group, Department of Computer Science
University of California, Santa Barbara
February 2012 – September 2013

Research Assistant
Chair of System Design, Department of Management, Technology and Economics
Swiss Federal Institute of Technology Zurich
August 2011 – July 2012

Research Assistant
Chair of Communication Systems, Department of Computer Science
University of Bonn
January 2009 – September 2009

Teaching Experience

Teaching Assistant in Compilers (CS 160)
Department of Computer Science
University of California, Santa Barbara
January 2016 – March 2016

Teaching Assistant in Computer Architecture (BA-INF 023)
Department of Computer Science
University of Bonn
April 2010 – September 2010

Teaching Assistant in Systems Programming (BA-INF 034)
Department of Computer Science
University of Bonn
October 2009 – March 2010

Community Service

- Program Committee Member, 38th IEEE International Conference on Distributed Computing Systems (ICDCS), July 2018
- External Reviewer, 15th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), June 2018
- Program Committee Member, 10th USENIX Workshop on Cyber Security Experimentation and Test (CSET), August 2017
- External Reviewer, 26th USENIX Security Symposium (USENIX Security), August 2017
- Reviewer, Journal of Information Security and Applications, March 2017
- Program Committee Member, 1st International Conference on Advances in Cyber-Technologies and Cyber-Systems (CYBER), October 2016
- Program Committee Chair, 10th Graduate Student Workshop on Computing (GSWC), March 2016 (Best Reviewer Award)
- External Reviewer, 1th IEEE European Symposium on Security and Privacy (EuroS&P), March 2016
- External Reviewer, 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS), October 2015
- Program Committee Member, 9th Graduate Student Workshop on Computing (GSWC), October 2014 (Best Reviewer Award)

Publications

[1] **Kevin Borgolte**, Christopher Kruegel, and Giovanni Vigna. "Delta: Automatic Identification of Unknown Web-based Infection Campaigns". In: *Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Virgil D. Gligor and Moti Yung. Berlin, Germany: Association for Computing Machinery (ACM), Nov. 2013, pp. 109–120. ISBN: 978-1-4503-2477-9. DOI: `10.1145/2508859.2516725`.

[2] **Kevin Borgolte**, Christopher Kruegel, and Giovanni Vigna. "Relevant Change Detection: Framework for the Precise Extraction of Modified and Novel Web-based Content as a Filtering Technique for Analysis Engines". In: *Proceedings of the 23rd World Wide Web Conference (WWW)*. Ed. by Andrei Z. Broder, Kyuseok Shim, and Torsten Suel. WWW Companion. Developers' Track. Seoul, Republic of Korea: International World Wide Web Conference Committee (IW3C2), Apr. 2014, pp. 595–598. ISBN: 978-1-4503-2745-9. DOI: `10.1145/2567948.2578039`.

[3] Giovanni Vigna, **Kevin Borgolte**, Jacopo Corbetta, Adam Doupé, Yanick Fratantonio, Luca Invernizzi, Dhilung Kirat, and Yan Shoshitaishvili. "Ten Years of iCTF: The Good, The Bad, and The Ugly". In: *Proceedings of the 1st USENIX Summit on Gaming, Games and Gamification in Security Education (3GSE)*. Ed. by Zachary N. J. Peterson. San Diego, CA: USENIX Association, Aug. 2014. URL: `https://www.usenix.org/conference/3gse14/summit-program/presentation/vigna` (visited on 08/20/2018).

[4] Yinzhi Cao, Yan Shoshitaishvili, **Kevin Borgolte**, Christopher Kruegel, Giovanni Vigna, and Yan Chen. "Protecting Web Single Sign-on against Relying Party Impersonation Attacks through a Bi-directional Secure Channel with Authentication". In: *Proceedings of the 17th International Symposium on Recent Advances in Intrusion Detection (RAID)*. Ed. by Angelos Stavrou, Herbert Bos, and Georgios Portokalidis. Vol. 8688. Lecture Notes in Computer Science (LNCS). Gothenburg, Sweden: Springer International Publishing, Sept. 2014, pp. 276–298. ISBN: 978-3-319-11379-1. DOI: `10.1007/978-3-319-11379-1_14`.

[5] Mathias Payer, Ling Huang, Neil Zhenqiang Gong, **Kevin Borgolte**, and Mario Frank. "What You Submit is Who You Are: A Multi-Modal Approach for Deanonymizing Scientific Publications". In: *IEEE Transactions on Information Forensics and Security (TIFS)* 10.1 (Jan. 2015), pp. 200–212. ISSN: 1556-6013. DOI: `10.1109/TIFS.2014.2368355`.

[6] **Kevin Borgolte**, Christopher Kruegel, and Giovanni Vigna. "Meerkat: Detecting Website Defacements through Image-based Object Recognition". In: *Proceedings of the 24th USENIX Security Symposium (USENIX Security)*. Ed. by Jaeyeon Jung Jung and Thorsten Holz. Washington, D.C., USA: USENIX Association, Aug. 2015, pp. 595–610. ISBN: 978-1-931971-232. URL: `https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/borgolte` (visited on 08/20/2018).

[7] Shuang Hao, **Kevin Borgolte**, Nick Nikiforakis, Gianluca Stringhini, Manuel Egele, Michael Eubanks, Brian Krebs, and Giovanni Vigna. "Drops for Stuff: An Analysis of Reshipping Mule Scams". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Ninghui Li and Christopher Kruegel. Denver, CO, USA: Association for Computing Machinery (ACM), Oct. 2015, pp. 1081–1092. ISBN: 978-1-4503-3832-5. DOI: `10.1145/2810103.2813620`.

[8] Antonio Bianchi, **Kevin Borgolte**, Jacopo Corbetta, Francesco Disperati, Andrew Dutcher, John Grosen, Paul Grosen, Aravind Machiry, Christopher Salls, Yan Shoshitaishvili, Nick Stephens, Giovanni Vigna, and Ruoyu Wang. "Cyber Grand Shellphish". In: *Phrack* 15.70 (Jan. 2017). Authors listed alphabetically. URL: `http://phrack.org/papers/cyber_grand_shellphish.html` (visited on 08/20/2018).

[9] Tobias Fiebig, **Kevin Borgolte**, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. "Something From Nothing (There): Collecting Global IPv6 Datasets From DNS". In: *Proceedings of the 18th Passive and Active Measurement (PAM)*. Ed. by Mohamed Ali Kâafar, Steve Uhlig, and Johanna Amann. Vol. 10176. Lecture Notes in Computer Science (LNCS). Sydney, Australia: Springer International Publishing, Mar. 2017, pp. 30–43. ISBN: 978-3-319-54328-4. DOI: `10.1007/978-3-319-54328-4_3`.

[10] **Kevin Borgolte**, Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. "Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates". In: *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS)*. Ed. by Patrick Traynor and Alina Oprea. San Diego, CA, USA: Internet Society (ISOC), Feb. 2018. ISBN: 1891562-49-5. DOI: `10.14722/ndss.2018.23327`.

[11] Yan Shoshitaishvili, Antonio Bianchi, **Kevin Borgolte**, Amat Cama, Jacopo Corbetta, Francesco Disperati, Andrew Dutcher, John Grosen, Paul Grosen, Aravind Machiry, Christopher Salls, Nick Stephens, Ruoyu Wang, and Giovanni Vigna. "Mechanical Phish: Resilient Autonomous Hacking". In: *IEEE Security & Privacy* 16.2 (Mar.–Apr. 2018), pp. 12–22. ISSN: 1558-4046. DOI: `10.1109/MSP.2018.1870858`.

[12] Tobias Fiebig, **Kevin Borgolte**, Shuang Hao, Christopher Kruegel, Giovanni Vigna, and Anja Feldmann. "In rDNS We Trust: Revisiting a Common Data-Source's Reliability". In: *Proceedings of the 19th Passive and Active Measurement (PAM)*. Ed. by Robert Beverly and Georgios Smaragdakis. Vol. 10771. Lecture Notes in Computer Science (LNCS). Berlin, Germany: Springer International Publishing, Mar. 2018, pp. 131–145. ISBN: 978-3-319-54327-7. DOI: `10.1007/978-3-319-76481-8_10`.

[13] **Kevin Borgolte**, Shuang Hao, Tobias Fiebig, and Giovanni Vigna. "Enumerating Active IPv6 Hosts for Large-scale Security Scans via DNSSEC-signed Reverse Zones". In: *Proceedings of the 39th IEEE Symposium on Security & Privacy (S&P)*. Ed. by Bryan Parno and Christopher Kruegel. San Francisco, CA, USA: Institute of Electrical and Electronics Engineers (IEEE), May 2018, pp. 438–452. ISBN: 978-1-5386-4353-2. DOI: `10.1109/SP.2018.00027`.

[14] Wei Meng, Chenxiong Qian, Shuang Hao, **Kevin Borgolte**, Giovanni Vigna, Christopher Kruegel, and Wenke Lee. “Rampart: Protecting Web Applications from CPU-Exhaustion Denial-of-Service Attacks”. In: *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*. Ed. by William Enck and Adrienne Porter Felt. Baltimore, MD, USA: USENIX Association, Aug. 2018. URL: `https://www.usenix.org/conference/usenixsecurity18/presentation/meng` (visited on 08/20/2018).

[15] Constanze Dietrich, Katharina Krombholz, **Kevin Borgolte**, and Tobias Fiebig. “Investigating Operators’ Perspective on Security Misconfigurations”. In: *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Michael Backes and XiaoFeng Wang. Toronto, ON, Canada: Association for Computing Machinery (ACM), Oct. 2018. ISBN: 978-1-4503-5693-0. DOI: `10.1145/3243734.3243794`.

This page intentionally left blank

Abstract

Identifying and Preventing Large-scale Internet Abuse

by

Kevin Borgolte

The widespread access to the Internet and the ubiquity of web-based services make it easy to communicate and interact globally. Unfortunately, the software and protocols implementing the functionality of these services are often vulnerable to attacks. In turn, an attacker can exploit them to compromise, take over, and abuse the services for her own nefarious purposes. In this dissertation, we aim to better understand such attacks, and we develop methods and algorithms to detect and prevent them, which we evaluate on large-scale datasets.

First, we detail Meerkat, a system to detect a visible way in which websites are being compromised, namely website defacements. They can inflict significant harm on the websites' operators through the loss of sales, the loss in reputation, or because of legal ramifications. Meerkat requires no prior knowledge about the websites' content or their structure, but only the Uniform Resource Identifier (URI) at which they can be reached. By design, Meerkat mimics how a human analyst decides if a website was defaced when viewing it in a browser, by using computer vision techniques. Thus, it tackles the problem of detecting website defacements through their attention-seeking nature, their goal and purpose, rather than code or data artifacts that they might exhibit. In turn, it is much harder for an attacker to evade our system, as she needs to change her modus operandi. When Meerkat detects a website as defaced, the website can automatically be put into maintenance mode or restored to a known good state.

An attacker, however, is not limited to abuse a compromised website in a way that is visible to the website's visitors. Instead, she can misuse the website to infect its visitors with malicious software (malware). Although malware is well studied, identifying malicious websites remains a major challenge in today's Internet. Second, we introduce Delta, a novel, purely static analysis approach that extracts change-related features between two versions of the same website, uses machine learning to derive a model of website changes, detects if an introduced change was malicious or benign, identifies the underlying infection vector based on clustering, and generates an identifying signature. Furthermore, due to the way Delta clusters campaigns, it can uncover infection campaigns that leverage specific vulnerable applications as a distribution channel, and it can greatly reduce the human labor necessary to uncover the application responsible for a service's compromise.

Third, we investigate the practicality and impact of domain takeover attacks, which an attacker can similarly abuse to spread misinformation or malware, and we present a defense on how such takeover attacks can be rendered toothless. Specifically, the new elasticity of Internet resources, in particular Internet protocol (IP) addresses in the context of Infrastructure-as-a-Service cloud service providers, combined with previously made protocol assumptions can lead to security issues. In Cloud Strife, we show that this dynamic component paired with recent developments in trust-based ecosystems (e.g., Transport Layer Security (TLS) certificates) creates so far unknown attack vectors. For example, a substantial number of stale domain name system (DNS) records points to readily available IP addresses in clouds, yet, they are still actively attempted to be accessed. Often, these records belong to discontinued services that were previously hosted in the cloud. We demonstrate that it is practical, and time and cost-efficient for attackers to allocate the IP addresses to which stale DNS records point. Further considering the ubiquity of domain validation in trust ecosystems, an attacker can impersonate the service by obtaining and using a valid certificate that is trusted by all major operating systems and browsers, which severely increases the attackers' capabilities. The attacker can then also exploit residual trust in the domain name for phishing, receiving and sending emails, or possibly distributing code to clients that load remote code from the domain (e.g., loading of native code by mobile apps, or JavaScript libraries by websites). To prevent such attacks, we

introduce a new authentication method for trust-based domain validation that mitigates staleness issues without incurring additional certificate requester effort by incorporating existing trust into the validation process.

Finally, the analyses of Delta, Meerkat, and Cloud Strife have made use of large-scale measurements to assess our approaches' impact and viability. Indeed, security research in general has made extensive use of exhaustive Internet-wide scans over the recent years, as they can provide significant insights into the state of security of the Internet (e.g., if classes of devices are behaving maliciously, or if they might be insecure and could turn malicious in an instant). However, the address space of the Internet's core addressing protocol (Internet Protocol version 4; IPv4) is exhausted, and a migration to its successor (Internet Protocol version 6; IPv6), the only accepted long-term solution, is inevitable. In turn, to better understand the security of devices connected to the Internet, in particular Internet of Things devices, it is imperative to include IPv6 addresses in security evaluations and scans. Unfortunately, it is practically infeasible to iterate through the entire IPv6 address space, as it is 2^{96} times larger than the IPv4 address space. Without enumerating hosts prior to scanning, we will be unable to retain visibility into the overall security of Internet-connected devices in the future, and we will be unable to detect and prevent their abuse or compromise. To mitigate this blind spot, we introduce a novel technique to enumerate part of the IPv6 address space by walking DNSSEC-signed IPv6 reverse zones. We show *(i)* that enumerating active IPv6 hosts is practical without a preferential network position contrary to common belief, *(ii)* that the security of active IPv6 hosts is currently still lagging behind the security state of IPv4 hosts, and *(iii)* that unintended default IPv6 connectivity is a major security issue.

This page intentionally left blank

Permissions and Attributions

1. The content of Chapter 2 is the result of a collaboration with Christopher Kruegel and Giovanni Vigna. It previously appeared in parts as "Meerkat: Detecting Website Defacements through Image-based Object Recognition" in the Proceedings of the 24th USENIX Security Symposium (USENIX Security) (Aug. 2015) [6]. It is reproduced with permission.

2. The content of Chapter 3 is the result of a collaboration with Christopher Kruegel and Giovanni Vigna. It previously appeared in parts as "Delta: Automatic Identification of Unknown Web-based Infection Campaigns" in the Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security (CCS) (Nov. 2013) [1]. It is reproduced with permission.

3. The content of Chapter 4 is the result of a collaboration with Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. It previously appeared in parts as "Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates" in the Proceedings of the 25th Network and Distributed System Security Symposium (NDSS) (Feb. 2018) [10]. It is reproduced with permission.

4. The content of Chapter 5 is the result of a collaboration with Shuang Hao, Tobias Fiebig, and Giovanni Vigna. It previously appeared in parts as "Enumerating Active IPv6 Hosts for Large-scale Security Scans via DNSSEC-signed Reverse Zones" in the Proceedings of the 39th IEEE Symposium on Security & Privacy (S&P) (May 2018) [13]. It is reproduced with permission.

5. Parts of the contents of Chapter 1, Chapter 6, and Chapter 7 are the results of the aforementioned collaborations. They are reproduced with permission.

This page intentionally left blank

Contents

List of Figures

This page intentionally left blank

List of Listings

This page intentionally left blank

List of Tables

This page intentionally left blank

Chapter 1

Introduction

The widespread access to the Internet and the ubiquity of Internet-connected services has made it easy to communicate and interact globally. Unfortunately, the software and protocols implementing the services' functionality are often vulnerable to attacks because of software or design bugs, legacy code, not being maintained, entirely new classes of vulnerabilities, a disconnect in what they are supposed to do and what they are actually doing, or just evolution in how they are being used. In turn, attackers can exploit them for their own nefarious purposes, to turn a profit from criminal enterprises, or to wreak havoc for fun. Today, such abuse on the Internet is rampant and users must tread carefully to not be defrauded, scammed, extorted, misled, influenced, or have their (private) data siphoned off and misused.

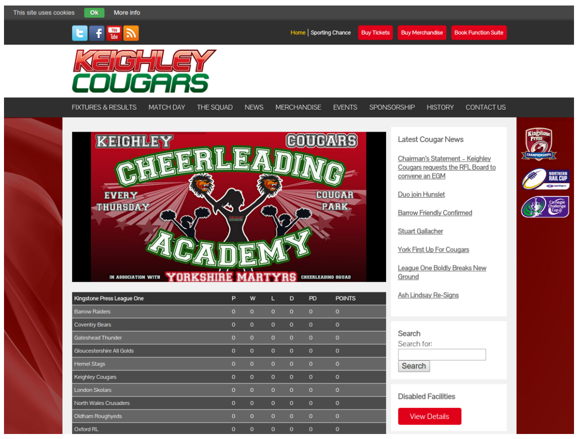
One attack plaguing the Internet is the defacement and vandalism of websites, which is an attack that disrupts the operation of companies and organizations, tarnishes their brand, and affects websites of all sizes, from those of large corporations to the websites of single individuals [16, 17, 18]. In a website defacement, an attacker replaces the content of a legitimate website with some of her own content. A website might be defaced for many different reasons and in many different ways. For example, an attacker might deface the website by brute-forcing the administrator's credentials, by leveraging a SQL injection to introduce content or code, or by hijacking the domain name. Ultimately, however, all defacements have one common characteristic: The defacer leaves a message that is shown to the visitors of the website instead of the legitimate content, changing the visual appearance of the website.

Although nearly all defacers vandalize websites for their "15 minutes of fame," and to get a platform to publicize their message, their messages vary. Some of them hope to embarrass the websites operators, others make a political or religious statement, and others again do it for "bragging rights." For instance, in the beginning of November 2014, as reported by the BBC [19], attackers defaced the website of the Keighley Cougars, a professional rugby club from England. The defacers modified the website so that visitors were greeted with a message in support of the terrorist organization "Islamic State of Iraq and the Levant/Syria" (ISIL/ISIS) (see Figure 1.1). In another example, in late 2012, defacers close to the Syrian regime defaced the homepage of the prominent Qatari television network Al Jazeera, and instead of news articles, visitors were shown a message alleging Al Jazeera of "spreading false fabricated news."

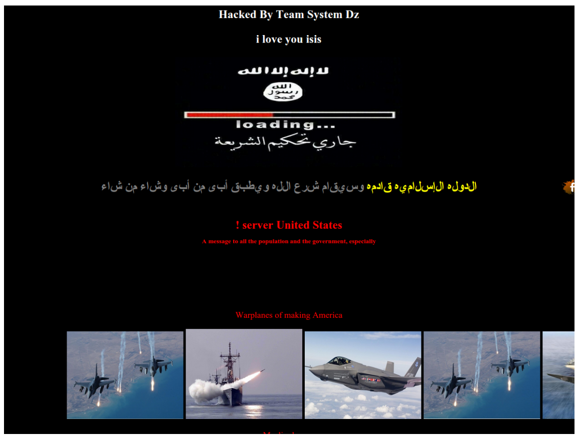
A prime example that quantifies the actual impact of defacements is the Telegraph, a major UK daily newspaper, which was defaced in September 2011. The Telegraph is the third most-visited website in the United Kingdom, according to MajesticSEO, and it is the 21st most visited website in the United States, according to Alexa. Each month, its homepage is visited over 125 million times (48 times per second), and, since reports state that the defacement lasted around three hours, an estimated more than 500,000 visitors saw the defacement instead of the legitimate website.[1]

Unfortunately, detecting website defacement abuse has not received much attention from the scientific community, while, at the same time, defacements have become more prominent. The number of reported defacements has been exceeding the number of reported phishing pages since

[1]The number of visitors was likely much higher because the website was defaced on a Sunday afternoon local time in the United Kingdom.



(a) Normal, non-defaced version



(b) Defaced version

Figure 1.1: Screenshots of the Keighley Cougars website, its normal version and after it was defaced in an attack on November 2, 2014 by the defacer group *Team System Dz*, who is using the defacement to show support for the terrorist organization Islamic State of Iraq and the Levant/Syria (ISIL/ISIS).

October 2006 by a factor of seven on average, and reached up to 33.39 defacements being reported to Zone-H[2] per phishing page reported to PhishTank[3] (see Figure 1.2). Similarly, while a mere 783 verified defacements were reported on average each day to Zone-H in 2003, the number of reports increased to 3,258 verified defacements per day for the year 2012, to over 4,785 verified defacements being reported each day to Zone-H in 2014. This corresponds to an increase of websites being defaced by 46.87% from 2012 to 2014 alone [20].

Defacements, however, are only one possible attack. An attacker can also exploit server-side vulnerabilities to inject malicious code snippets, called *infection vectors*, which, in turn, attack the website's visitors through drive-by install, download, or mining attacks. In drive-by mining attacks, the malicious code monetizes the compromise by abusing the visitors' computational resources to mine

[2]Zone-H [20] is an Internet archive containing only defaced websites, all reported defacements are mirrored locally and manually verified [21]. Upon manual inspection, a reported defacement is removed from the archive if it does not constitute a defacement, or it is marked as verified.

[3]PhishTank is the largest public clearinghouse of data about phishing scams, users report potential phishing scams and other users agree or disagree with the submitter, resulting in a user-assigned phishing score. Phishing pages are not being verified by expert analysts.

cryptocurrencies [22]. On the other hand, drive-by install and download attacks exploit client-side vulnerabilities to download or install malicious software (malware), or con the user into installing malware herself [23]. Once the malware has gained a foothold on the user's device, amongst other threats, it can steal a user's private data, like credit card numbers or social security information, or attempt to extort the user by encrypting her files and only disclosing the decryption key upon payment of a ransom. Furthermore, if an attacker finds a server-side vulnerability that affects multiple websites (possibly thousands), she can automate the exploitation, search for other vulnerable websites, and launch a carefully crafted *infection campaign* to maximize the number of potential victims.

Reports by the security company Sophos [24, 25] show that in 2012 over 80% of all websites attacking users were compromised legitimate websites, such as those of trade associations, nightclubs, television companies or elementary schools. All of these websites had been altered, in one way or another, to attack visitors. In another case, a range of websites hosting documentation for software developers were modified to serve carefully crafted infection vectors that exploited client-side vulnerabilities, which were then leveraged as the first stepping stone in sophisticated attacks against Twitter [26], Facebook's engineering team [27], and Apple [28].

Today's challenges in detecting such infection vectors are that websites have become more dynamic and that their static content changes regularly, that is, the underlying infection vector might not be easily detectable by analysis tools, or even by well-trained human security analysts. Adding new content to the website, showing different, personalized advertisements, or even comments left by visitors are legitimate modifications. Unwanted modifications on the other hand, that is, changes from which a user might want herself to be protected, include the aforementioned defacements or the insertion of an infection vector. Yet, any modification, legitimate or unwanted, resets what we know about the website's behavior and forces us to reanalyze it for maliciousness, either through an automatic detection system, or by manually inspecting any new content prior to its inclusion into the website. Even worse, prior work in the area of web evolution [29, 30, 31, 32, 33] suggests that websites do not change in well-defined and long intervals, but that they evolve constantly through small changes, and, if one takes into account personalized advertisements, a change might happen at every visit, which, in turn, makes it necessary to analyze the websites on each visit.

The state of the art in protecting a user from malicious web-based content is mainly implemented through blacklists, which are queried before the website is rendered or retrieved by the browser. The Google Safe Browsing list [23] is likely the most prominent example. By definition, blacklists are reactive, which is an undesirable property for a protection mechanism because a malicious website can potentially stay undetected for an extended period. Furthermore, each website compromised as part as an infection campaign needs to be identified and added to the blacklist separately, even though the websites attack visitors in the same, well-known way.

This is particularly problematic because Internet services have become much more elastic with the introduction of Infrastructure-as-a-Service (IaaS) and other as-a-Service offerings, such as those of the popular cloud service providers Amazon Web Services (AWS) or Microsoft Azure. Indeed, clouds have changed the entire landscape of system operations on the Internet because their elasticity allows operators to rapidly allocate and use resources as needed at the stroke of a key, without any prior commitment. While this elasticity is beneficial to the clouds' users and being credited for spurring innovation, attackers receive the same benefits, which means that they can evade reactive protection, such as blacklists, more easily. Even worse, the clouds' elasticity introduced new blind spots and made way for entirely new attacks that exploit previously ignored effects of resource sharing, which, for example, allow attackers to circumvent authentication to other cloud users' resources [34], or steal their cryptographic private keys [35].

In order to better assess the ramifications of these new blind spots and to understand the Internet's global state of security, the security community adopted Internet-wide security scans as a measurement method. These measurements allow us, for instance, to estimate the global impact of insecure configurations, shared cryptographic keys, or the scale of vulnerable software installations.

However, the increasing growth of the Internet will render state-of-the-art methods, which rely on exhaustive host scanning, soon unsuitable for accurate measurements: They are limited to the current version of the Internet protocol (version 4; IPv4), but IPv4's address space is exhausted, and the migration to version 6 of the Internet protocol (IPv6) is inevitable. In fact, some networks, such as the network of T-Mobile US [36], which is the third largest wireless carriers in the United States, are IPv6-only already. These networks cannot be scanned and analyzed by current approaches because the IPv6 address space is 2^{96} times larger than the IPv4 address space. In turn, to retain the capability of large-scale measurements of Internet abuse and abuse potential, we need to find new measurement techniques. Worse yet, IPv6 introduces additional attack surface that is ripe for abuse by miscreants because hosts previously accessible only in private networks might suddenly become publicly accessible, and they might not be secured properly, as it is often the case for Internet of Things devices. Indeed, a study comparing IPv6 and IPv4 security of roughly 300,000 servers and routers already indicated that IPv6 security posture is worse than its IPv4 counterpart [37]. Without access to the corresponding measurement techniques, we will lose the ability to understand future large-scale Internet abuse.

In this dissertation, we aim to better understand and address large-scale Internet abuse, and to improve visibility into the unknowns of the Internet, with the ultimate goal of making the Internet safer. We make the following contributions in the area of Internet abuse:

- We develop two techniques to *identify occurring abuse* on the Internet:
    - We develop Meerkat, a novel analysis method to automatically detect website defacement attacks (Chapter 2). Our analysis method does not rely on manual feature engineering or domain knowledge about website defacements, but it learns high-level features from data automatically. Meerkat detects a superset of defacement attacks compared to prior work, and it is more accurate.

  - We develop Delta, a new approach to automatically discover known and unknown web-based malware infection campaigns, assess their impact and scale, and facilitate root cause analysis (Chapter 3). Our approach leverages that malicious behavior requires modifications to be made, which Delta identifies, extracts, and analyzes.

- We present Cloud Strife, a mitigation to *prevent existing abuse potential*, which is caused by the increased elasticity of clouds (Chapter 4) Our mitigation stops the novel IP address use-after-free attack, which we discovered, from being successfully exploited for TLS-based services, for example, for phishing, impersonation attacks, or malware distribution.

- We introduce a new technique to *uncover future abuse potential*, that is, a technique to enumerate actively used IPv6 addresses for security scanning by pruning unused parts of the IPv6 address space (Chapter 5). We then use our technique to *evaluate* the security posture of IPv6-connected devices.

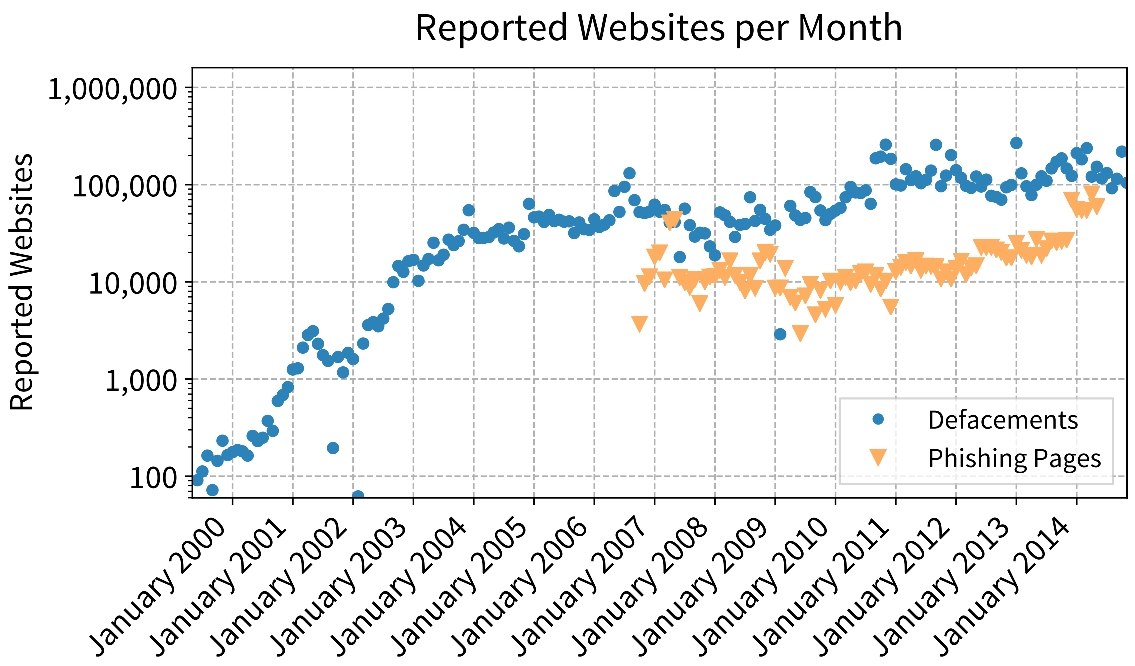


Figure 1.2: Defacements reported to Zone-H and phishing pages reported to PhishTank, per month from January 2000 to including October 2014. The drops in reported defacements in February 2002, February 2009, and March 2009 are because Zone-H was under maintenance, and they did not accept new reports. No data is available from PhishTank earlier than October 2006, when it was launched. The trend of an increasing number of defacements per month, as well as the gap in the number of defacements to the number of phishing pages of a factor of up to 33x are evident.

Chapter 2

Detecting Website Defacements

In this chapter, we approach the problem of defacement detection from a previously unexplored angle: We use computer vision techniques to recognize if a website was defaced, similarly to how a human analyst decides if a website was defaced when viewing it in a browser. We introduce Meerkat, a defacement detection system that requires no prior knowledge about the website's content or its structure, but only the Uniform Resource Locator (URL) at which it can be accessed. Upon detection of a defacement, the system notifies the website operator that her website is defaced, who can then take appropriate action. To detect defacements, Meerkat automatically learns high-level features from screenshots of defaced websites by combining recent advances in machine learning with techniques from computer vision. These features are then used to create models that allow for the detection of newly-defaced websites.

We show the practicality of Meerkat on the largest website defacement dataset to date, comprised of 10,053,772 defacements observed between January 1998 and May 2014, and 2,554,905 legitimate websites. Overall, Meerkat achieves true positive rates between 97.422% and 98.816%, false positive rates between 0.547% and 1.528%, and Bayesian detection rates[1] between 98.583% and 99.845%, thus significantly outperforming existing approaches.

2.1 Motivation and Contributions

The defacement and vandalism of websites is an attack that disrupts the operation of companies and organizations, tarnishes their brand, and plagues websites of all sizes, from those of large corporations to the websites of single individuals [16, 17, 18].

In a website defacement, an attacker replaces the content of a legitimate website with some of her own content. A website might be defaced for many different reasons and in many different ways: An attacker might deface the website by brute-forcing the administrator's credentials, by leveraging a SQL injection to introduce content or code, or by hijacking the domain name. Ultimately, however, all defaced websites share one characteristic: The defacer leaves a message that is shown to the website's visitors instead of the legitimate content, changing the visual appearance of the website. Unfortunately, reliably detecting such website defacements is challenging, as there are many ways in which an attacker can tamper with the website's appearance, including re-routing the traffic to a different website, which does not affect the legitimate website's content directly in any way.

According to the Malaysian Computer Emergency Response Team (CERT), 26.04% of all reported incidents in 2013 were website defacements, but only 1.5% of the reported incidents were defacements in 2003, and 10.81% were website defacements in 2007 [38, 39]. This surge and the increase in defacements and cyber-vandalism is generally attributed to the rise of hacktivist groups, like *anonymous* or *LulzSec* [40, 41], but also gained traction through the escalation of national and international conflicts [42, 43]. Although the scientific consensus is that the attacks employed to deface a website are rather primitive [40], hacktivist groups as well as other politically and religiously-

[1]The Bayesian detection rate is the likelihood that if we detect a website as defaced, it actually is defaced, that is, P(true positive|positive).

motivated defacers have been extremely successful recently: In February 2015, Google Vietnam was defaced by *Lizard Squad* and remained defaced for several hours [44]; in January 2015, the website of Malaysia Airlines was defaced by *Cyber Caliphate* [18]; in late 2014, the defacer group *Team System Dz* defaced over 1,700 websites to speak out against the actions of the US in the Syrian civil war and to advocate for ISIS/ISIL [17]; in April 2014, over 100 websites belonging to the government and major companies in Zambia were defaced by Syrian and Saudi Arabian defacers to voice against the Western world's "meddling" in the Syrian civil war [45]; in January 2014, the website of the popular mobile game Angry Birds was defaced in protest of governmental spying by the NSA and GHCQ [46]; and in October 2013, a Pakistani defacer group gained access to the domain registrars of Suriname, Antigua & Barbados, and Saint Lucia and defaced the regional websites of Audi, AVG, BlackBerry, BMW, Canon, Coca-Cola, Fujitsu, Hitachi, Honda, IBM, Intel, Microsoft, Samsung, Symantec, Rolls-Royce, Vodafone, and other companies for "bragging rights" [47]. Yet, website vandalism is still being played down as a problem, instead of being acknowledged and addressed.

Overall, attackers confirmedly defaced over 53,000 websites ranked on Alexa's, MajesticSEO's, and QuantCast's top 1 million lists in 2014, which shows that not only websites that are considering "low-hanging fruit" are being defaced, but that high-profile ones are being attacked alike (see Table 2.1). While the list of prominent defacements goes on [19, 48, 49, 50, 51, 52, 53, 54, 55, 56], it is important to note that most techniques to deface a website, like code and data injection attacks, improper access control, or DNS hijacking and poisoning, have been well-studied and protection mechanisms have been proposed by prior work [57, 58]. However, it is also extremely hard to protect against all defacement attacks simultaneously and at scale. Even worse, organizations are often responsible for hundreds (or thousands) of different websites, with different levels of security [40]. A single insecure website that is defaced, however, can inflict significant harm on the organization: in qualitative terms because of the loss of reputation, and in quantitative terms because of the cost of having to investigate and remove the defacement.

		Alexa		MajesticSEO		QuantCast	
Month	Website	US	Global	TLD[1]	Global	US	Page Views per Month ▼
Nov 2014	princeton.edu	999	3,412	17	273	3,444	796,000
	volvo.com	54,607	57,046	3,757	7,323	568,058	-
	cca.gov.in[2]	146,039[3]	780,660	-	-	-	-
Aug 2014	openelec.tv	7,226[4]	48,754	184	93,894	-	-
	omicsonline.org	7,561[3]	42,030	5,068	63,924	-	-
Jul 2014	ct.gov	2,454	10,976	72	2,054	3,548	809,000
	us.to	2,846[5]	28,100	18	11,061	-	-
	sunnewsonline.com	68[6]	9,958	31,315	58,277	236,740	-
	newsmoments.in	3,725	39,262	-	-	-	-
Jun 2014	wordpress.net	3,522[7]	41,295	1,410	28,021	321,317	-
May 2014	arynews.tv	72[8]	5,308	949	536,436	-	-
	sundaytimes.lk	120[9]	38,591	6	39,866	209,083	-
Mar 2014	taylorswift.com	3,560	23,425	12,161	23,608	15,678	1.2 million
	gbjobs.com	798[10]	9,181	-	-	-	-
Dec 2013	openssl.net	5,994	16,409	80	933	-	-
Oct 2013	avg.com	117	155	471	854	-	37 million
	aljazeera.net	25[11]	1,831	37	920	2,196	28 million
	bitdefender.com	5,934	5,898	1,132	2,094	3,963	1.4 million
	avira.com	24[12]	1,108	1,275	2,361	6,081	480,000
	leaseweb.com	359[4]	4,035	23,585	44,451	230,626	-
	metasploit.com	124,365	175,570	33,537	59,816	120,839	-
2011-2013[14]	telegraph.co.uk	21[13]	225	3	107	6[13]	125 million
	ups.com	71	231	319	549	101	40 million
	nationalgeographic.com	483	1,006	94	139	125	37 million
	acer.com	4,060	6,042	-	-	1,995	2.9 million
	theregister.co.uk	2,737	3,457	443	14	11,327	1 million
	vodafone.com	7,052[13]	20,625	5,833	2,980	101,624	-

Table 2.1: Recent high-profile websites that were defaced, with their respective page rank according to Alexa, MajesticSEO, and QuantCast, and their monthly page impressions. These defacements were reported to Zone-H and include a major logistics company (UPS), computer and information security vendors (BitDefender, Avira, AVG, MetaSploit), news websites (Al Jazeera, Ary News, News Moments, Sunday Times, Sun News Online, Telegraph, The Register), a scientific society (National Geographic), a hardware vendor (Acer), the world's second largest telecommunications provider (Vodafone), a singer-songwriter/actress (Taylor Swift), the state of Connecticut (ct.gov), an Indian federal ministry (cca.gov.in), an auto-mobile company (Volvo), an ivy-league university (Princeton), well-known open source projects (OpenSSL, OpenELEC), and a hosting provider (Leaseweb). Missing fields represent unavailable data, data is unavailable due to being kept secret by the website operators or requiring subscriptions to Alexa, MajesticSEO or QuantCast.

[1] Top-level domain rank [2] Government of India, Ministry of Communications & Information Technology [3] Rank in India
[4] Rank in Netherlands [5] Rank in Indonesia [6] Rank in Nigeria [7] Rank in Bulgaria [8] Rank in Pakistan [9] Rank in Sri Lanka
[10] Rank in China [11] Rank in Yemen [12] Rank in Iran [13] Rank in United Kingdom [14] Selected high-profile website defacements from Fortune 50 and Global 500 companies between 2011 to 2013

Prior work on website defacements detection focused on detecting unauthorized changes to the web server, for example, via host-based intrusion detection systems or file-based integrity checks. However, most previous approaches lack the capabilities to detect the most prevailing defacement techniques used today: code or data injection attacks, and DNS hijacking. This is because these attacks do not actually modify the code or configuration of the website, but instead they introduce new content or redirect the user to a different website.

Although defacements can inflict serious harm on the website operator, a two-month study by Bartoli et al. [59] shows that many website operators still react slowly to defacements with an average response time of over 72 hours. Moreover, their study finds that mere 24% of the defaced websites were restored within one day, about 50% defacements were removed within the first week, while more than 37% of the websites remained defaced for over two weeks. Overall, their findings suggest that prior website defacement protection techniques and detection methods have not been widely adopted, likely because they are not comprehensive and miss some classes of attacks.

The logical first step to reduce the harm inflicted by defacements on the website operator is to provide her means to *quickly* and *comprehensively* detect if her website has been defaced, so that she can put the website in maintenance mode or restore its content to a known good state. As such, an automatic, accurate, thorough, and lightweight defacement detection system that monitors websites, notifies the website's operator, and acts as an early warning system is desired. In this chapter, we introduce such a system, Meerkat, which is a monitoring system that automatically detects if a website has been defaced. Meerkat detects website defacements by rendering the website in a browser, like a human visitor would, and deciding, based on features learned exclusively from screenshots of defacements and legitimate websites observed in the past, if the website's *look and feel* is that of a defaced or a legitimate website. If the website is detected as being defaced, the system notifies the operator, who, in turn, can, depending on the confidence in Meerkat's decision, put the website (automatically) in maintenance mode or restore a known good state to reduce the damage.

We make the following contributions:

- We introduce Meerkat, a website defacement detection system that learns a high-level feature set from the visual representation of the website, that is, it learns a compressed representation of the *look and feel* of website defacements and legitimate websites. Based on the learned features, the system then produces a model to differentiate between defaced and legitimate websites, which it uses to detect website defacements in the wild. In addition, the system notifies the website's operator upon detection.
- We evaluate Meerkat on the largest website defacement dataset to date, spanning 10,053,772 website defacements observed between January 1998 to May 2014, and 2,554,905 legitimate and (supposedly) not defaced websites from the Alexa, MajesticSEO, and QuantCast top 1 million lists.

In the remainder of this chapter, we describe how Meerkat works in detail (Section 2.2), evaluate our system on the largest defacement dataset to date (Section 2.3), discuss some limitations of website defacement detection systems (Section 2.4), and, finally, we conclude (Section 2.5).

2.2 Approach

The approach Meerkat takes to detect website defacements is fundamentally different from prior work for three reasons. First, while the system does leverage machine learning for classification, it does not rely on handpicked features that were selected based on prior domain knowledge, that is, it requires no feature engineering. Instead, Meerkat relies on recent advances in machine learning, stacked autoencoders, to learn high-level features directly from data. Second, Meerkat does not require the website operator to supply any information other than the domain name at which her website can be accessed. We designed our system in this way because other defacement detection systems that require the operator to define keywords and other metadata, provide a reference version of her website, or describe the website's legitimate content, have rarely been adopted. By reducing the effort required from the website operator to actually use a defacement detection system, we hope to improve on this situation. Finally, MEERKAT approaches defacement detection visually. The system analyzes the *look and feel* of the website and how a user would experience it by rendering it in a web browser and analyzing a screenshot of the website, instead of analyzing its source code or content.

Approaching the problem of detecting website defacements visually has several advantages over analyzing the source code or content of a website. Some defacements rely heavily on JavaScript and Cascading Style Sheets (CSS) to stylize the defacement, which all must be analyzed in an overarching browser context, and others again rely heavily on images. In fact, similar to spam, phishing, and many scams, defacements often do not contain much textual content, but include images to display text instead [60], thus they trivially evade text-based detection approaches. Furthermore, the source code of two websites can be vastly different, yet they appear the same to the human eye when rendered in a browser. Therefore, leveraging prior work, to analyze the DOM tree, the website's code, or parts thereof, is unlikely to be successful when trying to detect website defacements accurately, which is why we opted for a perceptual approach that does not suffer from the aforementioned problems.

Following, we describe how MEERKAT learns from defacements and legitimate websites, and how it detects defacements in the wild. Next, we motivate the structure of our deep neural network briefly, then, we discuss the concept and motivation of fine-tuning the network, then, we provide some notes on our implementation, and, last, we briefly recap how MEERKAT can be deployed in practice.

2.2.1 Training and Detection

Before Meerkat can be trained, two crucial parameters must be selected that determine how and from what data the system learns the *look and feel* of defacements:

Window Size.

Meerkat is not trained on whole screenshots of websites, but on a window "into" each website (i.e., only a part of the screenshot), thus we must select the size of these *representative windows*. Some important considerations must be made before picking the size of the windows that we extract.

A small window can be more accurate because it might only contain the exact representative part of the defacement but not any noise, like an irrelevant background color. However, if the windows are too small, the system will also have more false positives because the windows are not representative of defacements; instead, they are representative for only parts of the defacements, which might also occur in legitimate websites.

On the other hand, when using larger windows, it will take significantly longer to train the network initially, but the network might learn a more accurate model. However, if the windows are too large, then the system will learn about specific kinds of defacements in-detail and overfit. For example, the system might learn that two defacements are different, while the two defacements are actually the same but have a slightly different, dynamically-generated background image.

Considering the trade-offs for different window sizes, for our implementation, we decided to extract windows that are `160×160` pixels in size. Our evaluation later shows that this window size works well in practice to detect website defacements (Section 2.3). We briefly explored other window sizes, like `30×30`, which fared worse.

Window Extraction Strategy.

The strategy to extract the representative window from a screenshot is fundamental to learn the *look and feel* of defacements and legitimate websites. If the windows are extracted according to some poorly chosen strategy, then we expect the classification accuracy to be poor as well. For instance, if the strategy always extracts the part of a website that is just a plain background, then the system will only detect plain backgrounds. Therefore, it is crucial that the window extraction strategy is chosen well, and we compare some suitable strategies, like extracting the window always from the center or at random, later (Section 2.2.1).

After selecting these parameters carefully, the system can be trained. This is where most of the complexity of Meerkat lies. The training phase works as follows:

1. We collect a considerable amount of labeled legitimate websites and defacements, and we extract their graphic representation (i.e., a screenshot of the browser window; Section 2.2.1).
2. For each sample, we extract the `160×160` representative window from each screenshot according to the selected extraction strategy (see Section 2.2.1).
3. The representative windows are first used to learn the features of our approach, and then to learn the model for classification, for which we use a neural network (see Section 2.2.2).

Once the neural network is trained, Meerkat detects defacements in the wild. Its detection phase consists of only two steps, on which we expand later:

1. The website is visited with a browser to retrieve a representative screenshot (Section 2.2.1).
2. A sliding window approach is used to check if the website is defaced and, if so, an alert is raised (Section 2.2.1).

Screenshot Collection

The first step to detect if a website has been defaced based on its *look and feel* is to collect a screenshot of how the website looks for a normal visitor. Meerkat visits the website with a browser that renders the website like any other browser would, and takes a screenshot once the browser finished rendering the website. In our implementation, we use PhantomJS to collect the screenshots of the websites. PhantomJS is a headless browser based on the WebKit layout engine that renders websites (nearly) identical to Safari or Google Chrome. PhantomJS also executes included JavaScript code, renders Cascading Style Sheets (CSS), and includes dynamic content, such as advertisements, like a browser that a human would use.

Another important aspect in collecting a representative screenshot of a website with a headless browser is the resolution of the simulated screen. The resolution of the display is important when collecting screenshots because many websites render differently for different screen sizes, such as for mobile devices, tablets, small laptops, or large displays. In our case, we decided to fix the resolution to `1600×900` pixels, which is a display resolution often found in budget and mid-range laptops.

Window Extraction Techniques

For training the system, after collecting the screenshots, we need to extract a representative window from each screenshot so that we can train the neural network to detect defacements. Various techniques can be used to extract the representative window, which can be grouped into deterministic and non-deterministic techniques. Hereinafter, we discuss the trade-offs for four possible techniques: *(i)* selecting the center window, *(ii)* selecting n non-overlapping windows according to some measure (explained later), *(iii)* uniformly selecting the window at random, and *(iv)* randomly sampling the window's center from a Gaussian distribution for the x and y dimension separately.

Deterministic Window Extraction The most straightforward deterministic technique is to always extracts the window from the center of the screenshot of the website. However, this makes evading the system trivial. Generally, if an attacker can accurately predict the window that will be extracted, he can force the system to learn about defacements poorly, and, in turn, deteriorate classification performance drastically. Therefore, such a simple technique is unsuitable for a detection system in an adversarial context.

Alternatively, one can extract the window according to some measure. Identifying the most representative window according to a measure (e.g., the Shannon entropy), however, forces us to compute it for all possible windows and then pick the top ranking one. In turn, for a `1600×900` screenshot and a `160×160` window, we would need to evaluate over one million candidate windows for each sample in the dataset. In total, for our dataset, this would require over 13 *trillion* computations of the measure just to extract the representative windows. Clearly, this is impractical.

Nonetheless, a deterministic selection strategy based on a clever measure can increase the accuracy of the system, and it can also be extended trivially to extract multiple top-ranking windows at no additional cost. However, using more than one window per sample increases the dataset size by a factor of n and prolongs training time. Therefore, n would have to be chosen carefully.

Taking into account the trade-offs the different deterministic extraction strategies bear (increased training and detection time, ease of evasion, or computationally impractical) and considering that a comprehensive evaluation of them would require at least an order of magnitude of additional experiments,[2] we decided to select a non-deterministic extraction strategy that follows intuition and is based on user interface and user experience design principles instead. This selection makes our classification performance a lower bound: Other window extraction strategies might be more accu-

[2]Performing these additional experiments would require at least six months just in computational time on our current GPU infrastructure, which is why we decided against performing them.

rate and/or robust, but (at the same time) they also incur significant additional cost at training and detection time.

Non-deterministic Window Extraction A relatively straightforward non-deterministic strategy to extract a window from a screenshot is to select it uniformly at random. However, one cannot simply take any point from the website's screenshot as the center of the window. Instead, it must be sampled so that the whole window contains only valid data, forcing us to sample its center from the interval $[80, 1520]$ for x and $[80, 820]$ for y (these intervals are specific to the screenshot size (`1600×900`) and window size (`160×160`)). Therefore, pixels at the border have a slightly lower probability to occur in a window than those in the center. Although this is an unintended side effect, it has negligible impact in practice because the center of a website is more likely to be descriptive anyways. Alternatively, we could create an "infinite" image by wrapping the screenshot at its borders, which would, however, yield artifacts because we would combine parts of the top of the website with parts of the bottom (and left and right, respectively), resulting in windows that do not occur on the real website, which, in turn, might disturb or confuse detection.

Alternatively to selecting the window's center uniformly at random, one can sample it from any other distribution, discretizing the sampled point. For instance, from a Gaussian distribution to extract windows from mostly the center of the screenshot, but not extracting from it exclusively. A focus on the center of the website is often desirable because it is likely to be more descriptive of the website's *look and feel*. For robustness, however, we also want to the system to not learn exclusively from the center but to also learn about defacements that occur at the border of the website. Therefore, for our implementation, we extract a single window per website with a Gaussian extraction strategy with $\mu_x = 800$ and $\sigma_x = 134.63975$ for x and $\mu_y = 450$ and $\sigma_y = 61.00864$ for y, so that the windows at the border of the screenshot have a lower probability to be sampled but are not ignored completely. If x and y values outside of the screenshot are sampled, we simply resample the value for x or y respectively. We selected these specific μ and σ values so that we sample values outside of the screenshot only with likelihood 0.0001%.

Defacement Detection

After Meerkat has been trained on a set of extracted windows, it can detect if a website has been defaced. Detecting website defacements with Meerkat is conceptually simple:

1. We visit the website that we want to check with our browser, and we take a screenshot of the rendered website (see Section 2.2.1).
2. We apply a standard sliding window detection approach on the screenshot we took to check if a part of the screenshot is detected as being defaced, similarly to prior work in image classification [61].
3. If a window is detected to be a defacement by Meerkat, we raise an alert and inform the website operator that her website has been defaced.

Note that Meerkat does not compare a possibly-defaced website to an older, legitimate version of it, and, thus, does not need to analyze or store an older version. Instead, it detects defacements solely by examining how the current version looks like.

Exclusively to improve performance, instead of starting in a corner of the screenshot, our system starts in the center and moves outward. This behavior is motivated by the fact that the center of the website is likely more descriptive, and our training set was focused on the center region of the screenshots. This does not mean, however, that Meerkat misses defacements that are at the border of a website, they will be detected when the sliding window reaches the actually-defaced part, the border. The same is also true if a website is only partially defaced: once the sliding window reaches the defaced area, Meerkat detects that the website is defaced.

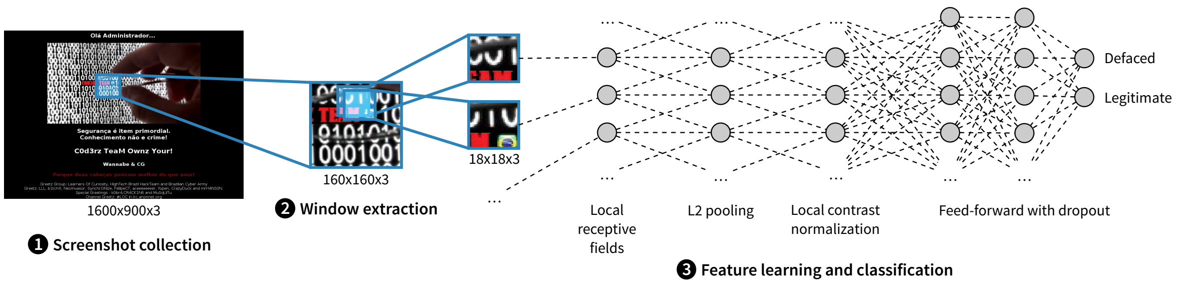


Figure 2.1: Meerkat architecture

Additionally, a special case worth mentioning is that a legitimate website might show a large promotional screen or an advertisement with the same intention of a website defacer: attracting attention. In turn, such a promotional screen might be similar in its *look and feel* to that of a website defacement. While Meerkat might currently (theoretically) mislabel them as defacements, our evaluation shows that they do not matter much (Section 2.3). Furthermore, if they start to matter at one point in the future, it is straightforward to consider them: the defacement engine can make use of an advertisement blocker, and the website operator could whitelist the system to not be shown any promotional screens.

2.2.2 Neural Network Structure

In this section, we briefly discuss the design of our deep neural network and how the different layers of the network interact with the input image. The structure of our deep neural network was notably inspired by prior work by Le at al. [62], Krizhevsky et al. [63], Sermanet et al. [61], and Girshick et al. [64]. We refer to them for further details.

The main components of our deep neural network are autoencoders, which we stack on top of each other, and a standard feed-forward neural network. Autoencoders are a special type of neural network that are used for unsupervised learning. The goal of an autoencoder is to find a compressed, possibly approximated encoding/representation of the input, which can be used to remove noise from the input, or, when autoencoders are stacked, they can learn high-level features directly from the input, like where edges in an image are, or if cats or human faces are part of an image [62].

Overall, the structure of our deep neural network is based on the following idea: First, we use a stacked autoencoder to denoise the input image and learn a compressed representation of both defaced and legitimate websites, that is, we leverage the stacked autoencoder to learn high-level features, similar to Le et al. [62]; and, second, we utilize a feed-forward neural network with dropout for classification, similar to Krizhevsky et al. [63].

The initial layer of our stacked autoencoder is composed of local receptive fields. This layer is motivated by the need to scale the autoencoders to large images [65, 66, 62, 67, 68], this layer groups parts of the image to connect to the next layer of the autoencoder, instead of allowing the whole image to be used as input to each node of the following layer. It takes 20,164 (142^2) sub-images of size 18×18 as input, extracted at a stride of 1 from the 160×160 representative window (see Figure 2.1; note that each pixel in each sub-image has three dimensions for the three colors: red, green, and blue). The second layer of our stacked autoencoder employs L2 pooling to denoise local deformations of the image and to learn invariant features [69, 67, 70, 71]. Finally, the last layer of our autoencoder performs local contrast normalization for robustness [72].

The output of the stacked autoencoder is then used as the input to a feed-forward neural network with dropout that provides a 2-way softmax output. The 2-way softmax output corresponds to the two classes that we want to detect: defaced websites and legitimate websites. We use dropout in our deep neural network to prevent overfitting of the network, and to force it to learn more robust features by preventing neurons to rely on other neurons of the network being available (i.e., to prevent the co-adaptation of neurons) [73].

2.2.3 Fine-Tuning the Network's Parameters

In an adversarial context, such as when trying to detect if an attacker defaced a website, concept drift can be introduced intentionally by the attacker and impede the accuracy of the detection system drastically. Furthermore, concept drift also occurs naturally, such as when the style of defacements evolves over time in such a way that the features cannot distinguish between legitimate and defacement anymore. Therefore, concept drift can be a severe limitation of any detection system, if it is not taken into account and addressed properly.

Meerkat can deal with concept drift in two different, fully-automatic ways: fine-tuning the network's parameters (adjusting feature weights), and retraining the entire network on new data. While the latter is conceptually straightforward and addresses all kinds of concept drift, it is computationally expensive. The former, on the other hand, allows us to deal with some forms of concept drift gracefully and is computationally much less expensive. However, it requires some further attention: when fine-tuning the neural network, Meerkat does not learn new features, but adjusts how important the already learned features are. Therefore, fine-tuning cannot address major concept drift for which the already learned features do not model defacements accurately anymore. Instead, when we fine-tune the network's parameters, we adjust the already learned weights of the deeper layers of the neural network so that new observations of defacements and legitimate websites are classified properly. As such, fine-tuning the network to maintain an accurate detection performance requires no additional information about the websites at all, but only defacements and legitimate websites that were not part of the training set before.

Conceptually speaking, when fine-tuning the network given new defacements and legitimate websites, we search for a better and, given the new data, *more optimal* set of weights in the space of all possible weights. To do so more efficiently, instead of initializing the weights at random, we initialize them based on the previously-learned weights.

2.2.4 Implementation

We implemented a prototype of Meerkat using Python and the "Convolutional Architecture for Fast Feature Embedding" (Caffe) framework by Jia et al. [74]. Caffe was used because of its high-performance and ease of use, however, it does not offer all functionality that our neural network requires and some modifications were made.

Overall, the general architecture of Meerkat is embarrassingly parallel: the screenshot collection engine is completely separate from the detection engine except for providing its input. For instance, to quickly collect the screenshots of all websites, we utilized 125 machines (with 2 cores and 2 GiB memory each), and collection peaked at about 300 screenshots per second. Similarly, once the neural network has been trained, the learned parameters can be distributed to multiple machines and detection can be scaled out horizontally, and, although the system is trained on a GPU, once trained, the detection engine does not require a GPU and can run on common CPUs instead.

Training the system, on the other hand, is not parallelized to multiple machines yet, but some clever tricks can be used to reduce training time significantly [63], which we leave for future work.

2.2.5 Real-world Deployment

Meerkat's main deployment is as a monitoring service, acting as an early warning system for website defacements, to which a website operator subscribes with only the URL at which his website can be reached. For each monitored website, the system regularly checks, such as every few minutes (or even seconds), that the website is not defaced. If it detects it as being defaced, it notifies the website's operator, who, in turn, depending on the confidence in the warning, manually investigates, or automatically puts the website in maintenance mode or restores a known good state. Acting as an early warning system, Meerkat reduces the reaction time to defacements from hours, days, and even weeks (see Section 2.1) down to minutes (or even seconds), and, therefore, it reduces the damage inflicted on the website's operator by the defacement significantly.

Furthermore, Meerkat can also reduce human labor: currently, Zone-H manually vets all submissions for defacements [21], of which nearly two-thirds are invalid. Meerkat automates this significant amount of work.

2.3 Evaluation

We evaluate our implementation of Meerkat in various settings. However, first, we provide details on what data our dataset is composed of, and how we partition it to simulate various defacement scenarios.

Our evaluation scenarios are traditional and simulations of real-world events, such as a new defacer group emerging, or how the system's accuracy changes over time, with and without fine-tuning the neural network.

In our experiments, a true positive is a website defacement being detected as a defacement and a true negative is a legitimate website being detected as legitimate. Correspondingly, a false positive is a legitimate website that is being detected as being defaced, and a false negative is a defacement being detected as being legitimate.

2.3.1 Dataset

The dataset on which we evaluate Meerkat contains data from two different sources. First, it includes a comprehensive dataset of 10,053,772 defacements observed from January 1998 to May 9, 2014. We obtained this data through a subscription from Zone-H, but it is also freely available from `http://zone-h.org` under a more restrictive license. From those defacements, 9,258,176 defacements were verified manually by Zone-H [21]. The remaining 795,596 website defacements were pending verification, and we do not include them in our dataset. Second, our dataset contains 2,554,905 unique (supposedly) undefaced websites from the top 1 million lists from Alexa, Majestic-SEO, and QuantCast.[3] Note that we cannot be certain that the legitimate websites in our dataset are not defaced, and since manual verification is impractical at such a large scale, the true negative rate is actually a lower bound and the false positive rate is an upper bound, correspondingly. In layman's terms: the system might be more accurate than our results suggest.[4]

To accurately evaluate the classification performance of Meerkat in a real-world deployment, we report its accuracy in three different scenarios:

- Traditional, to compare to prior work, that is, by performing 10-fold cross-validation by sampling from all data uniformly at random, so that each bin contains 925,817 defacements and 255,490 legitimate websites.
- Reporter, to simulate a new defacer emerging, that is, by performing 10-fold cross-validation on the reporters of a defacement and including only their defacements in their respective bin. Legitimate website are sampled from the legitimate data uniformly at random.
- Time-wise, to evaluate the practicality of our approach in a real-world setting, that is, we start by training the system on all data from December 2012 to December 2013, and, then, we detect defacements from January to May 2014. We report the system's detection accuracy for each month.

We evaluate our system in these settings to prevent a positive skew of our results that might be the result of the different evaluation method and how the dataset is composed. For instance, a reporter

[3]The list of all 2,554,905 legitimate websites included in our dataset is available upon request.

[4]Over 191,000 website in our legitimate dataset have been defaced at one point in the past, thus, it is likely that some of them are actually defaced and therefore mislabeled; thus, if classified correctly as a defacement by Meerkat, they appear as false positives in our results.

of a defacement might introduce an inherit bias to the distribution of the defacement by only reporting the defacements of one specific defacer (such as themselves), or there might be a bias in how defacements and how the web evolved. Those potential pitfalls might skew the results positively or negatively and must be considered for an accurate comparison to prior work.[5]

Finally, to account for the different number of samples of legitimate websites (2,554,905) and defaced websites (10,053,772), we report the Bayesian detection rate [75]. The Bayesian detection rate is normalized to the number of samples and corresponds to the likelihood if we detect a website as being defaced, it is actually defaced (the likelihood of a positive prediction being correct, that is a true positive; that is, P(true positive|positive)).

2.3.2 Features Learned

The features that MEERKAT learns depend on the data it is being trained on. Although one can treat the system as a black-box and not worry about its internal details, understanding how it comes to its final decision helps to reason about its robustness and to understand how difficult the system is to evade or to estimate when the system must be retrained to retain its accuracy. In our experiments, MEERKAT learned various features automatically and directly from image data, of which we manually grouped some on a higher, more conceptual level. We manually identified the learned features by evaluating which representative windows activate the same neuron of the neural network, that is, which windows trigger the same feature to be recognized by MEERKAT. Note that all the features we discuss hereinafter have been learned *automatically* from data and no domain knowledge whatsoever was required to learn and use these features; yet, the overlap with features that an analyst with do-

[5]We cannot compare prior work (Section 6.1) on our dataset directly as they do not scale to its size, and we cannot compare on their datasets because they are too small to train MEERKAT accurately.

main knowledge would use confirms the prospects of feature/representational learning for website defacement detection. Some of the learned features can be best described as:

Defacement Group Logos.

MEERKAT learned to recognize the individual logos of some of the most prolific defacement groups directly (see Figure 2.2). Clearly, the logos of the defacer groups themselves are extremely descriptive of website defacements because they are very unlikely to be included in legitimate websites.

Color Combinations.

MEERKAT also learned to recognize unique or specific color combinations indicative of legitimate and defaced websites, including but not limited to one of the most prominent combinations: bright red or green text on a black background, which is an often used color combination by defacers, but rarely seen on legitimate websites. On the other hand, small black text on a white or brightly colored background is being consulted as a non-definitive indicator for a legitimate, non-defaced website.

Letter Combinations.

Interestingly, defacers often not only mix colors, but also mix characters from different alphabets right next to each other, such as Arabic or Cyrillic script being mixed with Latin script, to promote their message in both their native language and also in English as the web's *lingua franca*. Additionally, sometimes the defacement contains characters in a character set encoding specific to the defacer's native language, like ISO-8859-13 for Baltic languages or Windows-1256 for Arabic. As such, characters appear differently or are replaced by special characters if the browser does not support it, or if the website does not specify the character set and if the browser's fallback is different (like in our case, as we fall back to UTF-8), resulting in a *look* *and feel* that is descriptive of defacements, and, correspondingly, it was automatically learned by Meerkat.

Leetspeak.

Similarly to letter combinations, Meerkat learned that defacers often use "leetspeak," an English alphabet in which some characters are replaced by numbers or special characters (e.g., "leetspeak" as "1337sp34k") and in which some words are deliberately misspelled ("owned" as "pwned," "the" as "teh," or "hax0red" instead of "hacked"). Defacers often use leetspeak to discern themselves from "common folks," and to show that they are "elite" and special, which, in turn, makes it often a good indicator that a website has indeed been defaced.

Typographical and Grammatical Errors.

While some typographical mistakes are deliberate (as in the case of leetspeak, see above), many defacers make other unintentional typographical and grammatical mistakes, which rarely occurred on the legitimate websites in our dataset. Many defacers make these mistakes most likely because they are non-native English speakers (the country of the reporter of the defacement, part of the meta-data in our dataset, suggests that most defacers do not speak English as their first language). Meerkat learned to detect some of these mistakes at training and values them as a supporting indicator of a website defacement. Some of the examples of (supposedly) unintentional typographical and grammatical errors include "greats to" (instead of "greets to"), "goals is" (instead of "goals are"), or "visit us in our website" (instead of "visit our website").

Note that, since Meerkat works on image data, the system is unaware that it analyzes text and the textual features, such as unique letter combinations, leetspeak, or typographical and grammatical errors, are actually being evaluated on rendered text. As such, it seems likely that the textual features are specific to the font, possibly overfitting on the specific font type. However, we manually

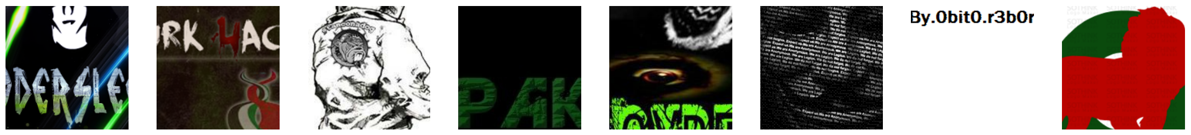


Figure 2.2: Example representative windows of defacement group logos that MEERKAT learned to recognized to be a significant indicator for defacements. Note that MEERKAT also recognizes variations and that there are many other features used for classification.

confirmed that the system actually learned a more robust feature and is not overfitting: it combines slight variances in the font family and size in a single high-level feature. Furthermore, given the sliding window approach MEERKAT employs for detection, the features are also completely independent of the position of the text in the representative window and website.

While some of the learned features can be evaded theoretically, evading them almost always contradicts the defacer's goal: Making a name for themselves in the most "stylish" and personalized way possible, thus, it is unlikely that these features will change drastically in the near future. Furthermore, MEERKAT also consults features that were not as easy to discern into high-level feature groups manually, such as artifacts unique to legitimate or defaced websites, or features that are indicative for one group but are not definitive because they might appear more often in defaced websites, but also sometimes legitimately. MEERKAT can also be retrained easily and new features are learned *automatically* once the old features do not model defacements accurately anymore (i.e., if the concept of a defacement drifted significantly). Finally, since MEERKAT uses a non-linear classifier to combine those features, it can learn more complex models about defacements and legitimate websites, and simply evading only some features will not be sufficient to evade detection.

Interestingly, some high-level features (letter and color combinations) that MEERKAT learned automatically from data have been leveraged to a smaller degree by prior work [76, 77] (through manual feature engineering), while others (logos, leetspeak, and typographical mistakes) had not been utilized yet. Further suggesting that representation learning and inspection of the learned features can yield important insight into security challenges that were dominated by feature engineering in the past, such as intrusion, malware, or phishing detection.

2.3.3 Traditional Split

First, for an accurate comparison to prior work, we evaluate Meerkat on our dataset using 10-fold cross-validation, that is, we split the dataset into 10 bins that contain 925,817 website defacements and 255,490 legitimate websites each. Note that we discard six website defacements and five legitimate websites from our dataset at random to have the same number of samples in each bin. Next, for each bin, we train the system on the other nine bins (training bins) and measure its classification performance on the 10th bin (test bin). Considering the 10 different 90% training and 10% test-set partitions of our dataset separately, Meerkat achieves true positive rates between 97.422% and 98.375%, and false positive rates ranging from 0.547% to 1.419%. The Bayesian detection rate is between 99.603% and 99.845%.

More interestingly, as a partition-independent measure of the system's classification performance, the average true positive rate is 97.878%, the average false positive rate is 1.012%, and the average Bayesian detection rate is 99.716%. If Meerkat detects a defacement and raises an alert, with likelihood 99.716% it is a website defacement. Therefore, Meerkat is significantly outperforming current state-of-the-art approaches.

2.3.4 Reporter Split

For the reporter split, we partition our dataset by the reporter of the defaced website. We deliberately designed the experiment this way to show that Meerkat is not overfitting on specific defacements, which our results verify.

While a partition by reporter might seem counter-intuitive at first, it becomes clear that such a split is meaningful and that it can be used to evaluate that a new defacer group emerges once it is taken into account that these groups often have unique defacement designs and that defaced websites are most often reported by the defacers themselves. Therefore, if we split by reporter, we are practically splitting by defacer group. Meaning, we create the most difficult scenario for a defacement detection system: detecting a defacer and her defacement style although we have never seen defacements from him/her before.

In the same way as for the traditional split, we employ 10-fold cross-validation. However, we do so slightly differently: first, we separate the reporters of the defacements into 10 bins uniformly at random (each bin containing 7,602 reporters). Second, we construct the corresponding defacement bins, that is, we construct a defacement bin for each reporter bin so that it contains only the defacements reported by these reporters. For each bin, we then train Meerkat on the remaining nine bins and use the 10th bin for testing. Note that the defacement bins contain a different number of samples, simply because the number of reported defacements varies per reporter (see Table 2.2). We account for the uneven distribution of defacements by reporting the average true positive and false positive rate weighted by the number of samples.

Overall, when simulating the emergence of a new defacer, Meerkat achieves a true positive rate of 97.882% and a false positive rate of 1.528% if bins are weighted, and 97.933% and 1.546% if they are not (see Figure 2.3; the true positive rate is between 97.061% and 98.465%, the false positive rate is between 0.661% and 2.564%). The Bayesian detection rates for the reporter split are 99.567% (unweighted) and 99.571% (weighted) respectively (per split, the Bayesian detection rate is between 99.286% and 99.814%).

2.3.5 Time-wise Split

The time-wise experiment evaluates how well Meerkat detects website defacements in the wild, that is, in a real-world deployment. Here, we train the system on defacements seen in the past, and we

Bin	**Defacements**	**Legitimate Websites**
1	1,116,808	308,202
2	992,232	273,823
3	712,270	196,563
4	907,306	250,387
5	696,069	192,092
6	734,208	202,617
7	1,276,764	352,345
8	789,895	217,985
9	979,309	270,257
10	1,053,147	290,634
Total	9,258,008	2,554,905

Table 2.2: Number of samples per cross-validation bins used for the reporter split. The total number of defacements in the reporter split contains fewer defacements than available in the whole dataset because otherwise reporters would be distributed unevenly per bin. However, due to the considerable size of the dataset, omitting these defacements has negligible impact.

detect defacements in the present. Similar to the reporter split, the time-wise experiment shows that MEERKAT does not overfit on past defacements, and that it successfully detects present defacements.

Our training set selection follows a simple argument: It is extremely unlikely that websites today will be defaced in the same way as they were defaced in 2005 or even 1998. Including those defacements in our training set would then very likely decrease classification performance for defacement detection in 2014. Equivalently, one would not include this data to train the system in practice.

Therefore, we train MEERKAT on all defacements that were reported between December 2012 and December 2013 (i.e., 13 months with 1,778,660 defacements observed in total), and 1,762,966 legitimate websites that we sample from all legitimate websites uniformly at random. We then detect defacements over a five months time frame, from January to May 2014, and we report the classification performance for each month. The test data from January to May 2014 spans a total of 1,538,878 unique samples that are distributed as follows: 421,758 samples from January 2014, 364,168 samples from February 2014, 474,758 samples from March 2014, 241,926 samples from April 2014, and 81,268 samples from the beginning of May 2014.

In detail, MEERKAT achieves a true positive rate between 98.310% and 98.816% when the system is fine-tuned after each month on the data observed in that month, and 97.603% to 98.606% when it is not. Although there is no significant difference in its accuracy from January to March when the neural network is fine-tuned and when it is not (see Figure 2.4), a non-negligible difference between their accuracy can be observed for April and the beginning of May (increase in 0.452 percentage points (pp) and 1.211 pp for the true positive rate; decrease of 1.513 pp and 1.550 pp for the false positive rate). The Bayesian detection rate, if no fine-tuning is used, decreases from 98.583% in January 2014 to 97.666% in February (0.917 pp decrease) to 97.177% in May (1.406 pp decrease to January). If fine-tuning is utilized, the Bayesian detection rate increases from 98.583% in January 2014 to 98.717% in May (0.134 pp).

Unsurprisingly, the regularly fine-tuned system performs better over time, probably because some defacers became significantly more active in 2014, like *Team System Dz*, who started to deface websites just in January 2014 and who were not active before at all, and because some defacers changed their defacements to spread a different message as opposed to the one they spread the year before. When the system is not fine-tuned, however, these minor changes to the defacements allow attackers to evade detection without actively trying to evade it, with a minor accuracy deterioration already visible after just four to five months, confirming that detection systems need to be able to tackle even minor concept drift adequately and gracefully to maintain accurate detection capabilities over time, like MEERKAT does with fine-tuning.

Split	True Positive Rate	False Positive Rate	Bayesian Detection Rate
Traditional	97.878%	1.012%	99.716%
Reporter (weighted)	97.882%	1.528%	99.571%
Reporter (unweighted)	97.933%	1.546%	99.567%
Time-wise with Fine-tuning	98.310% – 98.816%	1.233% – 1.413%	98.583% – 98.767%
Time-wise without Fine-tuning	97.603% – 98.606%	1.413% – 2.835%	97.177% – 98.583%

Table 2.3: Average true positive, false positive, and Bayesian detection rates for traditional and reporter split. Lower and upper bound of true positive, false positive, and Bayesian detection rate for time-wise split from January to May 2014.

2.4 Limitations

Similar to other systems leveraging machine learning, our system has some limitations that can be used to evade detection. We discuss some of these limitations and show how they can be addressed for a real-world deployment. First, we discuss *concept drift*, a problem all systems leveraging machine learning have to deal with; second, we remark on *browser fingerprinting* and *delayed defacement*, an issue all client-based detection approaches have to address; and, lastly, we introduce the concept of *tiny defacements*, a limitation specific to defacement detection systems.

2.4.1 Concept Drift

Concept drift is the problem of predictive analysis approaches, such as detection systems, that the statistical properties of the input used to train the models change. In turn, a direct result of concept drift is often a heavy deterioration of the classification performance, up to the point where the system cannot differentiate between good and bad behavior anymore. For instance, prior work [78, 79, 80, 81, 82, 83, 84, 85] has shown that concept drift (in the sense of adversarial learning) can actually be leveraged to evade detection systems and classifiers in practice. Therefore, a detection system must address it.

While concept drift is a major issue for all systems using machine learning, it can generally be addressed, due to its nature, by adopting a new feature space or retraining the machine learning model on new data, or with an increased weight on new data. However, often, old instances do not follow the statistical properties of the new feature models, and, therefore, they are classified less accurately than before. This has little impact in practice, because old instances are less likely to occur in the future anyways, but it is important to realize that this approach allows attackers to evade the system by oscillating their attack strategy.

For Meerkat, those shortcomings can be addressed more easily than for traditional systems. For minor concept drift, the system's accuracy can be maintained by fine-tuning the parameters of the network. Here, the system simply needs to learn minor adjustments to the weights of existing features from new data, because some features have become more important and others have become less important (they differ now more from other features than they did previously, relatively speaking; since we start with already-initialized weights, fine-tuning requires much less time than training the whole system again). The features still model the differences between defacements and legitimate websites, however, the weights are not optimal anymore and need to be adjusted. Once the new weights are learned, classification performance is restored. Therefore, to address minor concept drift adequately, we recommend fine-tuning the model regularly, for example, every month (see Section 2.3.5).

While fine-tuning the system's parameters can theoretically address major concept drift similar to retraining the system on new data, in practice, we expect prediction accuracy to decrease, since different or more features must be modeled with the same amount of resources. Instead, for major concept drift, increasing the number of hidden nodes of the neural network that learn the compressed representation (the features) and their weights, and then retraining the system can maintain the system's accuracy. Simply adding nodes to the hidden layers of the neural network can counteract the issue of major concept drift because we increase the number of features that Meerkat learns from data directly. Therefore, introducing more hidden units allows the system to learn additional and different internal representations about the *look and feel* of defacements, while, at the same time, maintaining a model of how the old defacements look like. However, it requires computationally-costly retraining of the network (previously, having those additional hidden units in the network would result in overfitting because the system would learn more complex representations than necessary, and each would only differ little from one another; the system would then be prone to missing minor variations of defacements).

It is important to note that in both cases, for minor and major concept drift, Meerkat requires no additional feature engineering because the features are learned *automatically* from data. In turn, this allows Meerkat to handle any form of concept drift much more gracefully than approaches introduced by prior approaches, which require manual feature engineering.

2.4.2 Fingerprinting and Delayed Defacement

A second limitation of detection systems is fingerprinting. Since we are leveraging a web browser to collect the data that we are analyzing, in our case fingerprinting corresponds to IP-based and browser fingerprinting. For IP-based fingerprinting, a set of VPNs and proxies can be used to cloak and regularly change the browser's IP address. In case of browser fingerprinting, the server or some client-side JavaScript code detects what browser is rendering the website, and then displays the website differently for different browsers. In its current form, the screenshot engine from Meerkat might be detectable (to some degree) by browser fingerprinting. It is theoretically possible to detect it because it is currently built on the headless browser PhantomJS rather than a "headful" browser typically used by a normal user, like Google Chrome. However, since PhantomJS is built from the same components as Google Chrome, fingerprinting the current screenshot engine is not trivial and requires intimate knowledge of the differences between the different versions of the components and their interaction. Therefore, we argue that the evasion through browser fingerprinting is unlikely. If, however, the screenshot engine is evaded this way in the future, only some minor engineering effort is required to utilize a browser extension for a headful browser to retrieve the websites' screenshots instead.

Additionally, the issue of delaying the defacement emerges, also referred to as the snapshot problem [1]. With the increased popularity and use of JavaScript, client-side rendering, and asynchronous requests to backends by websites to provide a seamless and "reload-free" user experience, it is uncertain at what point in time a website is representative of how a user would experience it. This then bears the issue of when a detection system can take a representative snapshot of the website and stop executing client-side scripts. For instance, if a detection system takes a snapshot always after five seconds, to evade detection, defacers could simply inject JavaScript that only defaces the website if a user interacts with it for at least six seconds.

While delayed defacements are currently scarce, it is likely that they will gain some traction once more detection systems have been put in place, in a way similar to mimicry attacks and the evasions of malware detection systems [86, 87]. However, prior work can be leveraged to detect evasions [88] or trigger the functionality [89] to force the defacement to be shown. Both approaches are complementary to MEERKAT and we leave their adoption to defacement detection for future work, once delayed defacements are actually occurring in the wild.

2.4.3 Tiny Defacements

A third limitation of all current defacement detection systems, including MEERKAT, is the lack of detection capabilities for tiny defacements. Tiny defacements describe a class of defacements in which only a very minor modification is made to part of the content of the defaced website. For instance, a defacer might be dissatisfied by an article published by a newspaper. Instead of defacing the website as a whole, the attacker modifies (or deletes) the news article. It is clear that such defacements are very hard to differentiate from the original content because they might only have minor semantic changes to text or images. Thus, to detect tiny defacements, the detection system must understand the semantics of the website's content, its language, and its general behavior to derive a meaningful model for the website.

However, while those defacements exist, they are extremely scarce in numbers, or they are rarely noticed. In fact, it is seldom the case that a defacer wants to modify a website without embarrassing the operator more publicly. Most often, the goal of the defacer is to expose the insecurity of the website, ridicule the operator, show their own "superiority," and place their opinion and beliefs in the most public space possible. Therefore, tiny defacements are currently of little interest to the defacers themselves, and, hence, also of little impact for detection systems. However, we acknowledge that tiny defacements must be addressed once they increase in numbers, possibly leveraging recent work to extract relevant changes from websites [2], and advances in natural language processing.

2.5 Conclusion

In this chapter, we introduced Meerkat, a monitoring system to detect website defacements, which utilizes a novel approach based on the *look and feel* of a website to identify if the website has been defaced. To accurately identify website defacements, Meerkat leverages recent advances in machine learning, like stacked autoencoders and deep neural networks, and combines them with computer vision techniques. Different from prior work, Meerkat does not rely on additional information supplied by the website's operator, or on manually-engineered features based on domain knowledge acquired *a priori*, such as how defacements look. Instead, Meerkat automatically learns high-level features from data directly. By deciding if a website has been defaced based on a region of the screenshot of the website instead of the whole screenshot, the system is robust to the normal evolution of websites and defacements and can be used at scale. Additionally, to prevent the evasion of the system through changes to the *look and feel* of defacements and to be robust against defacement variants, MEERKAT employs various techniques, such as dropout and fine-tuning.

We showed the practicality of MEERKAT on the largest website defacement dataset to date, spanning 10,053,772 defacements observed between January 1998 and May 2014, and 2,554,905 legitimate websites. On this dataset, in different scenarios, the system accurately detects defacements with a true positive rate between 97.422% and 98.816%, a false positive rate between 0.547% and 1.528%, and a Bayesian detection rate between 98.583% and 99.845%, thus significantly outperforming existing state-of-the-art approaches.

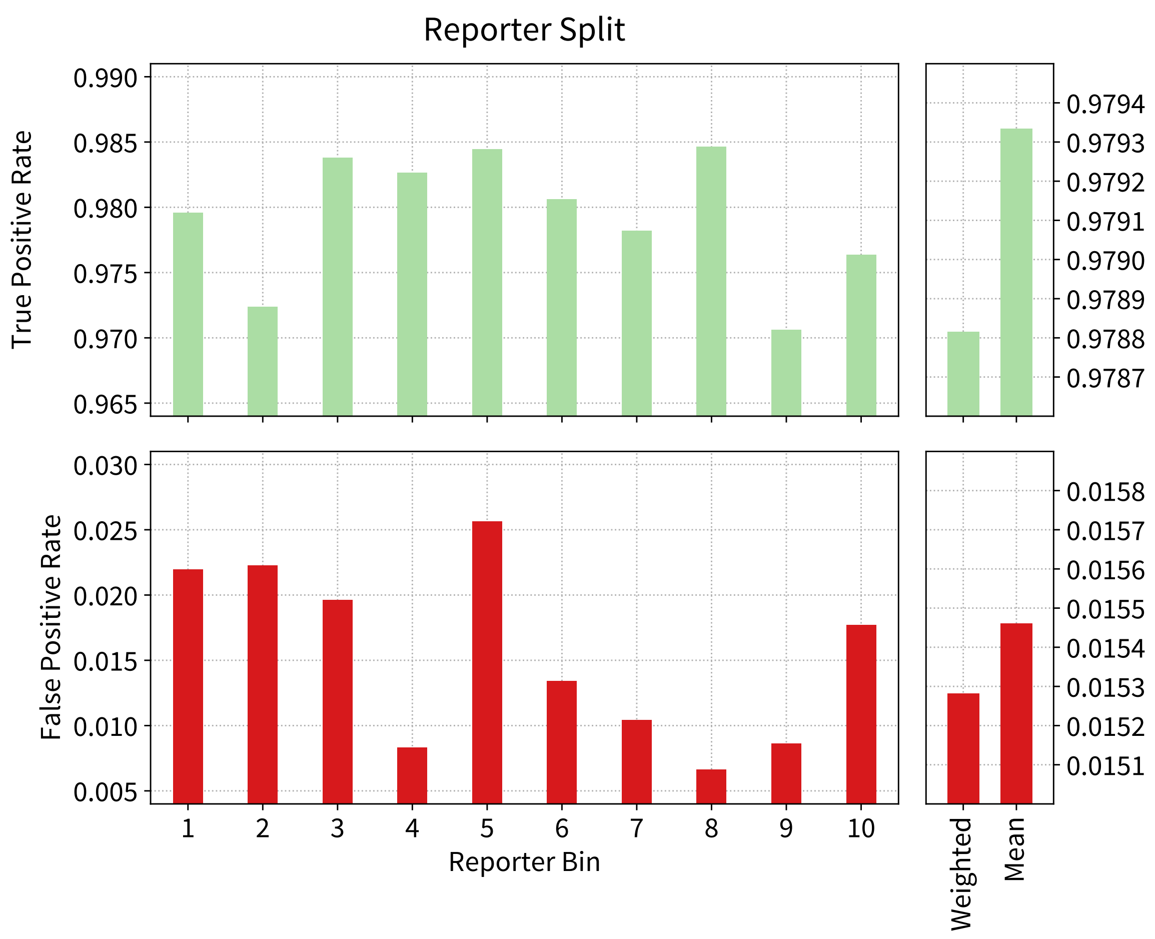


Figure 2.3: True positive and false positive rates for the reporter split per bin of the 10-fold cross-validation set. Note that the scales for true positives and false negatives are the same, but that the y-axis goes from 0.965 to 0.99 for the true positive rate and 0.005 to 0.03 for the false positive rate. The weighted mean true positive rate is 97.882% and its false positive rate is 1.528% (weighted by samples per bin). The unweighted mean true positive rate is 97.933% and its false positive rate is 1.546%.

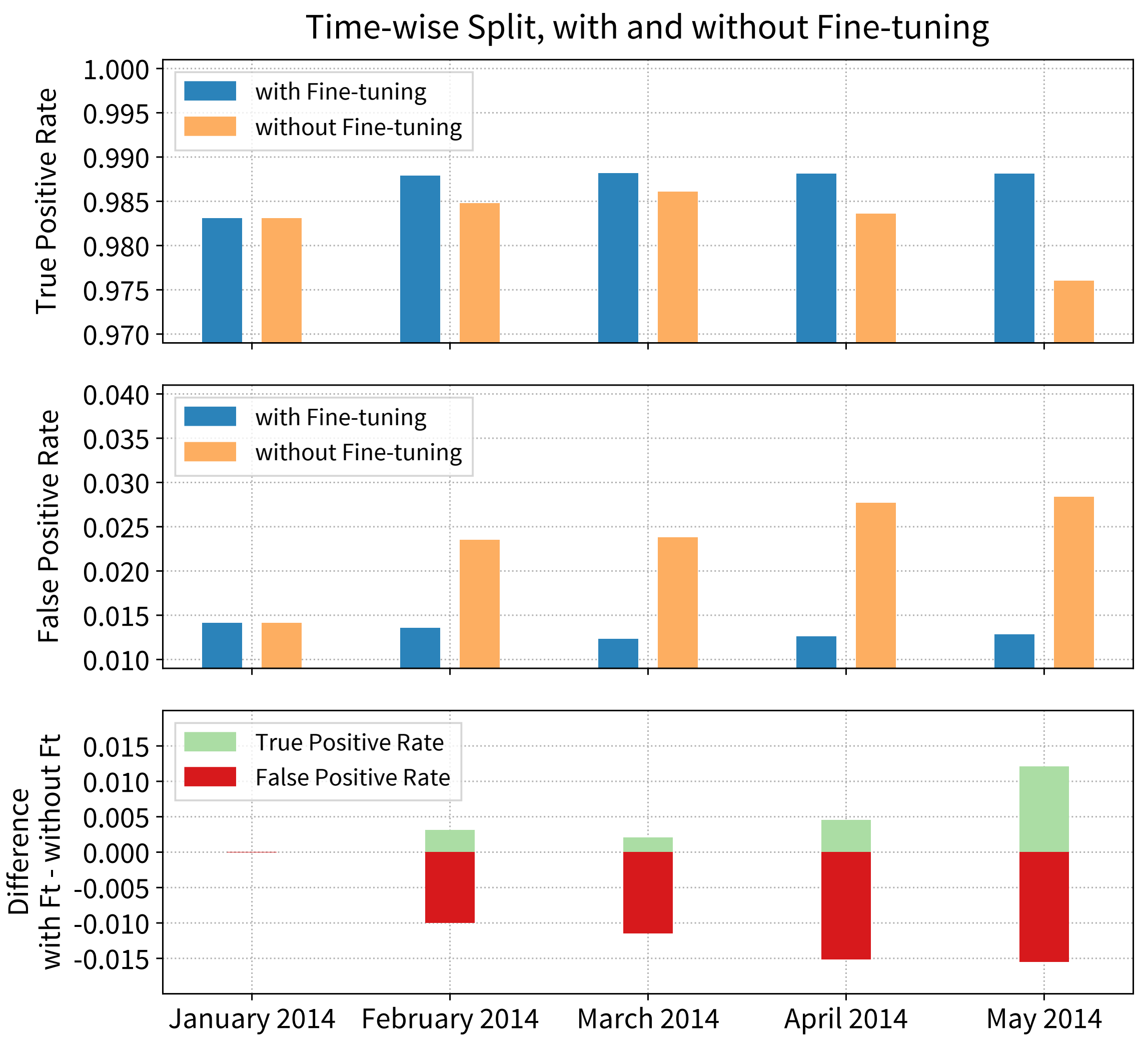


Figure 2.4: True positive and false positive rates, and their differences with and without fine-tuning, for the time-wise split. Note that the scales for true positives and false negatives are the same, but that the y-axis goes from 0.97 to 1 for the true positive rate and 0.01 to 0.04 for the false positive rate. No significant change is visible for the true positive rate in the beginning regardless if the network is fine-tuned regularly or not, however, a non-negligible difference is observable for May 2014. A difference is observable for the false positive rate starting in February 2014, after the network was first fine-tuned.

This page intentionally left blank

Chapter 3

Identifying Web-based Malware Infection Campaigns

Identifying malicious websites has become a major challenge in today's Internet. Previous work focused on detecting if a website is malicious by dynamically executing JavaScript in instrumented environments or by rendering them in client honeypots. Both techniques bear a significant evaluation overhead, since the analysis can take up to tens of seconds or even minutes per sample.

In this chapter, we introduce DELTA, a novel, purely static analysis approach, which *(i)* extracts change-related features between two versions of the same website, *(ii)* uses a machine-learning algorithm to derive a model of website changes, *(iii)* detects if a change was malicious or benign, *(iv)* identifies the underlying infection vector campaign based on clustering, and *(v)* generates an identifying signature.

We demonstrate the effectiveness of DELTA by evaluating it on a dataset of over 26 million pairs of websites, by pairing it with a web crawler for four months. Over this time span, DELTA successfully identified previously unknown infection campaigns, including a campaign that targeted installations of the Discuz!X Internet forum software, in which infection vectors were injected that redirected the forums' visitors to an installation of the Cool Exploit Kit.

3.1 Motivation and Contributions

The rapid growth of the Internet and the pervasiveness of web-based services has made it easy to interact with others globally. The software used to implement the functionality of websites, however, is often vulnerable to different attack vectors, like cross-site scripting or SQL injections. In turn, an attacker can exploit these server-side vulnerabilities to inject malicious code snippets, so called *infection vectors*, that attack visitors of a compromised website through drive-by attacks. In such drive-by install and download attacks, these malicious code snippets exploit client-side vulnerabilities to install malware, either directly themselves or indirectly through redirections, or they try to convince the user to install malware herself. The malware, once installed, can then steal her credentials to leverage them for phishing attacks on colleagues and acquaintances, siphon off sensitive data, such as her credit card information, or encrypt her files and demand a ransom. Moreover, if an attacker automates the exploitation of a server-side vulnerability, she can launch an *infection campaign* spanning thousands of websites, to maximize the number of her victims.

A major challenge for accurately detecting infection vectors is that websites have become more complex and more dynamic, and, as a result, the infection vector might not be perceptible by analysis techniques, or even difficult to discover for experienced human analysts. For example, legitimate content changes might be made to the website, like showing personalized advertisements, or users might have interacted with the website and left a comment or review. Unfortunately, as soon as a modification was made, the website must be reanalyzed, as a single modification can change the website's behavior and turn it malicious. Even worse, previous work in the area of web evolution [29, 30, 31, 32, 33] suggests that websites evolve constantly through small changes, and if one takes into account personalization, a change that requires reanalysis might occur every visit for each user.

The most prominent method to protect users from malicious web-based content are, as we discussed before in Chapter 1, blacklists, such as the Google Safe Browsing list, which are reactive. This is problematic because their reactive character implies that blacklists are not guaranteed to be comprehensive, and that malicious websites can potentially stay undetected for an extended time, as each website of an infection campaign needs to be identified and added to the blacklist individually. Considering that the same infection vector is often reused by an attacker and spread among a multitude of different websites in an infection campaign to maximize its impact, current defenses might not be able to protect users from *known* attacks. Furthermore, once a website is blacklisted, justified or by accident, its operator must go to great lengths to remove her website from the blacklist, although it might now be benign. Such a removal process can take a frustrating amount of time since it is often subject to some form of verification that the website is now benign, a process that might not happen immediately, during which the website remains inaccessible to a potentially large share of its users. In turn, users are inconvenienced, possibly alienated, and companies might experience financial loss or a loss in reputation. Clearly, a proactive on-demand approach is preferable to detect infection vectors, especially those that are *unknown* and not covered by blacklists at all. Unfortunately, the most promising proactive approach, that is, scanning a website preemptively with an online analyzer system [90, 91, 92] is computationally very expensive, and it introduces delays up to multiple seconds per website. Since such a delay imposes an unacceptable cost to the users' experience, it is unlikely that these approaches would be widely deployed, or that they would find their way into current browsers as a protection mechanism.

However, the compromised websites that are part of an infection campaign usually follow a simple pattern: The infection vector was inserted in the same or very similar way, and its appearance is similar across the campaign. In terms of compromise, they might be inserted by taking advantage of improperly controlled access, or by exploiting software vulnerabilities in a web framework or application used by all of the campaign's websites. Often, the infected websites share some commonalities, such as employing the same underlying software stack, or sharing a server that was attacked. For example, in 2013, Apache web server installations were attacked, and the web server's executable was augmented with the backdoor "Linux/Cdorked.A" [93, 94], which injects malicious code to redirect the web server's visitors to exploit pages. Consequently, by being able to identify an infection vector as part of a campaign, instead of just detecting that the website is malicious, we can provide important additional feedback, and we can analyze its root cause more easily because of commonalities in different observations.

To overcome the limitations of current approaches mainly based on dynamic analysis of websites, and to facilitate root cause analysis, we introduce Delta, which identifies malicious activity in a website based on static analysis of the differences between the current and previous versions of the website. We cluster these differences, determine if the introduced or removed elements are associated with malicious behavior, identify the infection campaign it belongs to, pinpoint the actual infection vector, and automatically generate an identifying signature that can be leveraged for content-based protection.

The main contributions of this chapter are:

- We introduce Delta, which is based on a novel approach to statically analyze and detect web-based infection vectors, and which identifies infection campaigns based on features associated with modifications observed between two versions of a website.
- We develop a tree difference algorithm that is resistant to tiny changes, such as typographical corrections or the small evolutionary modifications a website undergoes.
- We develop a set of modification-motivated similarity measures to model the concepts of inserting and removing malicious behavior into and from a website.

- We evaluate Delta on a large scale dataset, containing 26 million unique pairs of websites, to show its applicability in real-world scenarios in terms of infection campaign detection and identification capabilities.

3.2 Approach

Delta, instead of trying to solve the problem of deciding if a website is malicious or benign provides a solution to the search problem of finding new infection campaigns and identifying similar, known infection campaigns. Nonetheless, we are still interested in deciding if a website's current behavior is malicious or benign. Instead of analyzing websites in their entirety, Delta investigates only the difference between two versions of the same website.

The main idea of Delta is to identify if the change made to a website altered the behavior of the website, that is, if we can be certain that the new version of the website is malicious or benign, by investigating if the modifications are similar to already observed ones, such as modifications associated with an ongoing infection campaign. In order to identify the changes that were made to a website, we need a base version, that is, an older version of the same website.

The analysis process of our system is described hereinafter, followed by a discussion on potential uses of our system, and the impact of deploying Delta.

3.2.1 Analysis Process

Delta's analysis process follows a simple four-step process, which is shown in Figure 3.1, and whose steps are:

1. Retrieval and normalization of the website.

2. Similarity measurement with respect to a base version.

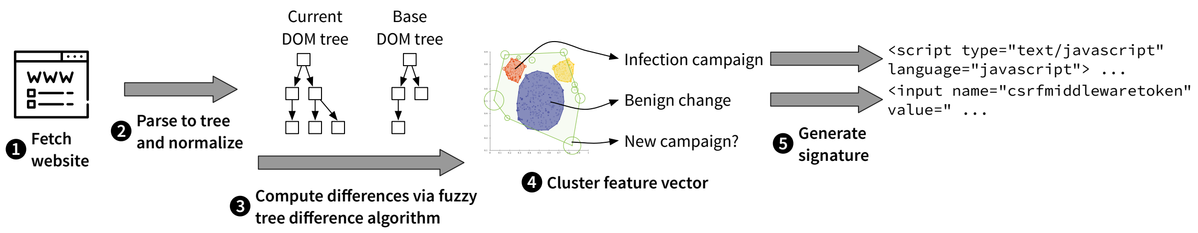

Fetch website
Parse to tree and normalize
Current DOM tree
Base DOM tree
Compute differences via fuzzy tree difference algorithm
Infection campaign
Benign change
New campaign?
Cluster feature vector
Generate signature
<script type="text/javascript" language="javascript"> ...
<input name="csrfmiddlewaretoken" value=" ...

Figure 3.1: Delta architecture

3. Cluster assignment of the similarity vector.

4. Generation of the identifying signature.

Evidently, a base version of a website has to be available. In the case that a local base version does not exist, however, we might still be able to retrieve an older version through web archives, such as the Internet Archive [95] or a web cache provided by a search engine. This makes our approach applicable for websites that are visited rarely and were no local base version is kept, if we accept the overhead to retrieve the base version from a remote archive. While this might seem counter-intuitive because of the potentially large time difference between the archived and current version, we show in our evaluation that this is indeed a possible alternative.

Following this brief overview, we discuss the important steps of the analysis process in more detail. First, normalization of a website; second, how the similarity to the base version is measured; third, how the identifying signature is generated.

Retrieval and Normalization

First, we retrieve the current version of a website, for instance the website a user requested. Then, after we have retrieved the source code of that website, excluding all external references, such as included scripts or frames, we perform multiple normalization steps: we normalize capitalization of all tags, we reorder attributes of each tag and discard invalid attributes, and we normalize the quotation of an attribute's value. We perform this normalization step to ensure that functional equivalent tags are treated equally during our evaluation, and that changes such as changing the capitalization of a tag or switching from single to double quotes do not affect our final results.

Similarity Measurement and Clustering

Following these normalization steps, we measure the similarity to an already known (and normalized) base version. Measuring the similarity between two versions of the same website in a meaningful way is non-trivial. The Delta performs unordered tree-to-tree comparison via a novel algorithm that we introduce in Section 3.3. The algorithm extracts the nodes (or tags; subsequently, we use both terms interchangeably) from the Domain Object Model (DOM) tree of a website that are different between base and current version. Second, based on the extracted nodes, we leverage a variety of different features to extract meaningful information from the two versions (described in detail in Section 3.4). The system then tries assigning these feature vectors to a cluster, or detects them as outliers, if they are not similar to any previously-observed modifications. Each different tag type, for example., <input> or <script>, is treated separately, that is, each type is assigned its own feature space; we do not project two tags of a different type into the same feature space. Additionally, due to the different nature of our features, where different distance metrics are essential for accurate cluster assignment, we perform consensus clustering for different groups: binary features, absolute and relative features are all treated as separate clustering instances. The cluster assignment and outlier detection process then distinguishes between three different cases:

- Assignment to an existing cluster:
  - Insertion or removal of an infection vector, if the cluster corresponds to a known infection campaign.
  - Legitimate modification, for example, a version update of a library or the insertion of Facebook's like button, if the cluster does not correspond to an infection campaign.

- Detection as an outlier:
  - Potentially the start of a new infection campaign, if malicious behavior was inserted.
  - Potentially the end of a running infection campaign, if malicious behavior was removed.
  - A modification that is not of primary interest to us, such as a new article, template modifications, or a redesign of the website.
- Formation of a new cluster (the similarity vector we are clustering and other vectors that are close, which were outliers before, put the number of total vectors in this area of the feature space above the threshold to form a new cluster, that is, we observed the number of the same modification in the wild that we require to constitute a trend, see Section 3.5.1):
  - New infection campaign, if the node was inserted and is associated with malicious behavior.
  - End of an infection campaign, if the node was removed and is associated with malicious behavior.
  - Legitimate modification, for example, an update to a new, bleeding-edge version of a library or the content-management system used, such as Wordpress.

Upon cluster assignment of the similarity vector, we output the associated cluster, that is, the corresponding trend (subsequently, we use these terms interchangeably). For instance, an infection campaign if the corresponding modification inserted or removed a known infection vector. Here, it is important to note that the detected clusters do not discriminate between removed and inserted nodes but treat them equally because we do not leverage the notion of removal or insertion in the feature computation, but attach it to the vector as "external" meta information that is not used during clustering. This supports the detection of removal and insertion of the same trend with the exact same cluster in both cases and, therefore, increases robustness of our system.

Delta does not provide detection capabilities for malicious behavior on its own, but rather relies on an external detection system. This detection system is queried once a new cluster is formed to identify if this observed trend constitutes a malicious or a benign cluster. In order to guarantee a high likelihood, we "bootstrap" each cluster by querying for 10 random samples, and acquire a consensus decision for the returned labels. For instance, a new cluster is observed and nine out of 10 of the random samples from this cluster have been assigned the label malicious, then Delta will assign the label malicious to any new observation in this cluster.

Signature Generation

For each of the identified trends, we generate a signature that matches the textual representation of all the nodes assigned to a cluster (e.g., a signature might describe the cluster containing "<script src='http://$random-url/exploit.js'>"). This signature is generated by simply interpreting the textual representation of each node as a deterministic finite automaton (DFA), merging them together, and calculating the minimal version, which can be done in polynomial time. The resulting DFA can then be translated into a regular expression that can be used by intrusion detection/prevention systems.[1]

Such an identifying signature is, generally, an under/approximation of the actual (unknown) signature. For instance, in the above example the URL is randomized. Here, the generated signature only describes the observed samples, that is, where $random-url might be "a.com" or "b.org," while the trend could be more general and also include "c.net." Leveraging only the identifying signature would miss websites that follow the same trend, that is, websites who might serve the same infection vector. While this is of no concern in the case of leveraging Delta for every request (here, we would assign a similar, but unobserved, tag to the same cluster), it can be an issue if the generated signature is used as input to other tools. A possible remedy is to generalize the signature and to introduce a

[1]Although generated signatures match normalized tags by default, it is trivial to normalize incoming data in the same way and match arbitrary tags that follow the same trend.

widening operator to describe the different parts of the nodes following this trend. For instance, one could simply widen five different characters at the same position in five different random samples picked from a cluster to an over-approximating wildcard. An over-approximation, however, is also likely to introduce incorrect matches, which is why we recommend using Delta if no exact signature matched.

3.2.2 Use Cases

We see Delta to be deployed in two main scenarios: paired with a web crawler to actively search for new infection campaigns, and paired with a proxy to identify infection campaigns (passive), or to improve user-protection (active). Additionally, there is a third, minor scenario: providing feedback on evasions of detection systems. Subsequently, we describe all three use cases in more detail, however, in the remaining of the chapter, we focus on the first use case: paired with a web crawler.

The most interesting use case, in our opinion, is the active identification of new infection campaigns. In this case, one deploys the system side-by-side to a web crawler. While the web crawler retrieves potentially interesting websites multiple times over a given period, our system analyzes the differences. When our system detects a new cluster, that is, a significant number of very similar modifications, an external detection system then decides if this change is associated with malicious behavior or not. If malicious behavior was introduced then we found a new infection campaign, and we can generate the identifying signature for this cluster. Based on the elements of the cluster, we can then pinpoint the infection vector (e.g., identify parts of the tag that are common among all websites in that cluster) and investigate other similarities manually (e.g., only online stores running a specific version of the PHP application osCommerce were infected). Starting from those similarities, it is then possible to: generate a more precise fingerprint for the campaign, find other infections via search engines, and estimate the scope of an infection campaign.

The second envisioned deployment of Delta is the extension of a web browser or a proxy. In most cases, the browser or proxy already caches visited websites for performance reasons. Moreover, in security-sensitive environments, it is very likely that a detection system (e.g., an anti-malware engine) is already in place to ensure that only benign websites can be accessed by the user. Such a detection system can be leveraged by Delta to analyze inserted tags. The system can complement these tools to prevent repetitive scanning of websites, to improve user experience by increasing analysis performance, and to provide insight into targeted attacks. For example, small changes a user might encounter include automatic page impressions counter, updated weather or date information, or the output of the processing time to render the website on the server's side. While previous work requires the reevaluation of the entire website, Delta can identify these changes as benign much more easily. It is even possible to obtain more accurate results with our system than with the detection system, for example, if it is based upon simple detection methods, such as fingerprinting of known malicious scripts, or if the detection system is being evaded. Additionally, once a malicious website is identified, Delta can verify that a malicious modification was removed and that the website is now benign. Particularly, if an infection campaign is dormant or the exploit page is offline, dynamic analysis systems detect that the website is benign because it does not detect any malicious behavior. Since Delta is purely static and verifies that the malicious content was removed, it does not have this disadvantage.

Lastly, Delta can also be leveraged to detect evasions and bugs in detection systems and online analyzers. For example, if the analyzer is dynamic, but behaves different than a standard browser in even a single case, then malware can fingerprint the detection system. Such a fingerprinting method allows the attacker to thwart detection much more easily, for instance, without having to utilize a blacklist of the IP addresses used by the online analyzer. Leveraging our system, we can detect these evasions when they are introduced. The system can pinpoint the changed content precisely and, by doing this, support the developer in identifying the reason why the analyzer is behaving differently, correcting the corresponding bug, and preventing further evasions leveraging the same bug.

3.3 Fuzzy Tree Difference

First, to be able to measure the similarity between two websites in a meaningful way, we need to define the notion of difference. We are primarily concerned if a website behaves in a benign or malicious way. To this end, we need to understand what modifications to the content can result in behavioral changes, and how we can isolate the modifications from other parts of the website that have no effect on the overall behavior. We identify these interesting parts by leveraging the hierarchical structure of a website and interpreting a website through its DOM tree.

Previous work introduced various algorithms to detect the semantic change in hierarchical structured data. The main idea behind HTML, that is, describing how to display data instead of describing the semantics of the data itself, renders nearly all introduced XML-centered approaches unsuitable to extract meaningful information about the modifications. An often made assumption is that the underlying tree structure has a significant semantic relationship with the content, which is not necessarily the case for HTML. Moreover, leveraging standard maximum cardinality matching on cryptographic hashes and simple edge weights of 1 (based in the nature of cryptographic hash functions), any change would be visible, including very small changes that are uninteresting to us, such as single character or word changes and legitimate evolutions. We denote such a tree-to-tree comparison as *not tiny change resistant* or *not fuzzy*. To solve this problem, and to identify interesting modifications made to a website more precisely and more efficiently, we generalize the previous notion of tree difference algorithms and introduce a similarity weight. We refer to our algorithm as the *fuzzy tree difference algorithm*, which is heavily influenced by the unordered tree-to-tree comparison algorithms by Chawathe et al. [96] and Wang et al. [97]. Such a fuzzy algorithm is necessary when comparing websites that have evolved over an extended period, for example, have been edited constantly over a two-week period. Otherwise, the sheer number of remaining nodes to analyze makes it infeasible to leverage computationally expensive features with reasonable performance overhead.

While we provide a formal description of the algorithm in Algorithm 1, we give a brief informal description first: the algorithm expects three parameters, T_1, T_2 and t_r. T_1 and T_2 are normalized DOM trees, that is, all tags are capitalized in the same way, all attributes occur in the same order and their values are enclosed in the same way (quote-wise). t_r is the threshold value for the similarity measurement, and can range from 0 to 1. Starting from the trees T_1 and T_2, we create a temporary graph to match pairs of similar nodes through maximum weighted bipartite graph matching (Hungarian algorithm [98]). This graph is constructed by inserting every node of T_1, then inserting every node of T_2. For each node from T_2, we connect it with an edge to every node from T_1 that has a similar fuzzy hash value (i.e., the Jaro distance of both hashes must be greater or equal to t_r) and that takes the exact same path (in the sense of unordered tree-traversal) as the node from T_2. The edge's weight is equal to the similarity measured through the Jaro distance between both hashes (i.e., at least t_r). Additionally, we color all matched nodes blue. In the last step, we remove the corresponding matched nodes from the trees T_1 and T_2 and output a list of removed (remaining in T_1) and inserted (remaining in T_2) nodes.

While the reason for coloring nodes might not be obvious, later on, we leverage the color of a node in the remaining nodes of T_1 and T_2 in our similarity measures to detect a matching asymmetry, that is, if a tag with a very similar hash and the same path from the root node was matched, such as a template that was used more often in T_2 than in T_1.

The implementation of Delta under evaluation leverages *ssdeep* [99] as the fuzzy hash function and a threshold of 0.99 for the Jaro distance [100] (which is normalized to 0 to 1, that is, we require very similar tags). Similar to cryptographic hash functions like MD5 or SHA, a fuzzy hash function, such as *ssdeep*, maps arbitrary long values to a short hash. In contrast to cryptographic hash function, however, a fuzzy hash function maps similar values to similar hashes that can be then used to measure their similarity. This property allows us to efficiently compare nodes of the DOM tree or their content regardless of their actual length, which otherwise might be computational too expensive when using standard string similarity measures for longer tags or content. We selected the Jaro distance function to compare two hash values because it is a simple string similarity measure originally introduced for duplicate detection by Jaro [100] and best suited for short strings while accounting for exactly matched characters as well as transpositions, therefore it quantifies the similarity of fuzzy hashes for similar data accurately.

In general, a threshold value of 1 when used with a cryptographic hash function is equivalent to standard unordered tree-to-tree algorithms. On the other hand, a threshold value of 0 regardless of the hash function is equivalent to comparing every element to every other element and impractical for any modern website due to the sheer number of possible combinations, which is why a reduction of potential matches is essential.

3.3.1 Example

An example of the tree difference algorithm is shown in Figure 3.2. The source code of a simple base version and current version of a website are shown in Listing 3.1 and Listing 3.2 respectively. Two modifications to the source code were made: first, a head tag including a script tag with an external source URL was inserted, and, second, a typographical mistake in the class of the p tag was fixed and one word in its content was changed: “foo” was replaced by “bar.” Figure 3.2 illustrates that for a standard tree difference algorithm the modified p tag would, correctly, constitute a modified p tag (the removed p tag is marked with a red background, while the inserted p tag is marked with a green background). However, since we are interested in severe changes and modifications associated with behavioral changes, these tiny changes are uninteresting to us, and, like the example shows, they are discarded by our algorithm.

```
<html>
  <body>
    <p class="sumamry">
      foo
    </p>
  </body>
</html>
```

Listing 3.1: Example HTML source, base version

```
<html>
  <head>
    <script src="//url/malicious.js">
    </script>
  </head>
  <body>
    <p class="summary">
      bar
    </p>
  </body>
</html>
```

Listing 3.2: Example HTML source, current version

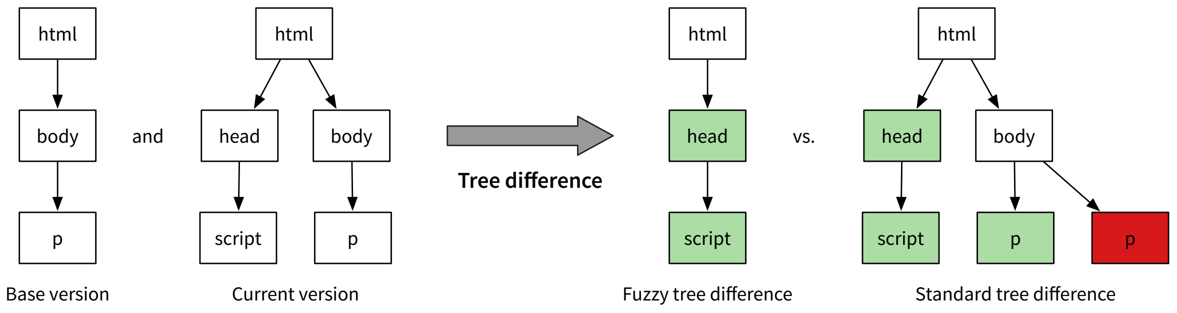


Figure 3.2: Comparison between a general tree difference algorithm and our fuzzy tree difference algorithm (Section 3.3). Nodes with a green background denote nodes that were detected as inserted in our example, while nodes with a red background were detected as being removed.

Algorithm 1 Fuzzy Tree Difference

```
1: function FuzzyTreeDifference($T_1, T_2, t_r$)
2:     $G \leftarrow$ Graph
3:     for all $n \in T_1$.nodes do
4:         $G \leftarrow G$.insert_node($n$)
5:     for all $n \in T_2$.nodes do
6:         for all $m \in T_1$.nodes do
7:             if path($m$) = path($n$) then
8:                 $d_{(m,n)} \leftarrow$ jaro(hash($m$), hash($n$))
9:                 if $d_{(m,n)} \geq t_r$ then
10:                    $G$.insert_node($n$)
11:                    $m$.color $\leftarrow$ blue
12:                    $n$.color $\leftarrow$ blue
13:                    $G$.insert_edge($m, n, d_{(m,n)}$)
14:    $M \leftarrow$ max_weight_matching($G$)
15:    for all $(m, n) \in M$ do
16:        $T_1$.remove_node($m$)
17:        $T_2$.remove_node($n$)
18:    return $T_1, T_2$
```

3.4 Similarity Measures

The most interesting part of websites from a malicious code point of view is described by the HTML markup language: JavaScript, inline frames, or the use of plugins. Most research on document similarity, however, assumes that markup language is not of major interest and that it can be removed without substantial loss of information. For detecting infection vectors, this assumption does not hold. Essentially, this violation makes applying existing work in document similarity for identifying infection vectors impractical, because core elements are discarded.

Therefore, we introduce our own similarity measures. Once we have extracted the different tags between two versions of a website, we can map each tag into the feature space in which we cluster similar changes together. In this section, we describe the features we are using and the intuition behind them. Each of our features we apply on multiple levels (where applicable): the whole tag and for every value of its attributes.

3.4.1 Template Propagation

First, we introduce the template propagation measure, a binary feature that simply models what content was introduced or removed from the website in terms of their similarity to previous DOM tree nodes, that is, it captures the concept of reused templates by checking if a node exist already in the base version, but are unmatched, for example, because there are more matching candidates than actual matches are possible. Since all matched nodes in the output T_1 and T_2 of our tree difference algorithm are colored blue, we can simply set the value of this feature to 1 if the node is blue and 0 if it is not.

The motivation for this measure is that many websites, for example blogs, use templates when publishing a new article or when showing a new comment, classified, or advertisement. Detecting that a template is repeated allows us to model the degree to which a website has drifted away from expected changes, for example, in terms of character count distributions for a blog with articles written in English.

3.4.2 Shannon Entropy

Second, we leverage the Shannon entropy as a feature of information in a tag or an attribute's value. Two different features are derived from the Shannon entropy: (a) the absolute Shannon entropy, which is dependent on the length of the string, (b) the normalized Shannon entropy, that is, the absolute Shannon entropy divided by the ideal Shannon entropy of a string of the same length (i.e., it is normalized to the interval from 0 to 1).

Our intuition behind the Shannon entropy is to measure the distance on how far the tag or attribute is away from a random source. For instance, to model that the URL in a "src" attribute of a <script> tag was generated by a random source.

3.4.3 Character Count

In the third set of features we employ a character count. The first feature in this set simply measures how often a single character occurs in the tag or the attribute's value and discriminates between upper- and lower-case characters. The second feature also measures how often a single character occurs, however, it ignores capitalization and counts an "A" as an "a." The third feature follows in simplicity and is the count of each digit. A fourth, fifth and sixth feature are taking advantage of the same method, but are performed on the fuzzy hash value (ssdeep in our implementation) of the tag or attribute instead.

Beyond these six features, we are also computing relative features for both of those two sets, as we did already in case of the Shannon entropy. Alike to the Shannon entropy features, the motivation behind these features is to model the character and digit distribution in a string.

3.4.4 Kolmogorov Complexity

The third set of measures we introduce is based on an approximation of the upper-bound on the Kolmogorov complexity [101]. Kolmogorov complexity denotes a complexity measure specifying the lower-bound of text necessary to describe another piece of text in an algorithmic way. One of the most important properties of the complexity is its incomputability. An upper-bound on the other hand is easy to compute by taking the length of the text compressed by any lossless compression algorithm. Since these features are based on a second information theoretical measure, next to the Shannon entropy, it is necessary to emphasize that they are complementary in our scenario: on the one hand, Kolmogorov complexity is conceptually different from the Shannon entropy, and on the other hand, we approximate the Kolmogorov complexity up to (at best) an additive constant.

These measures exploit the upper-bound and the fact that compressing already packed data results in nearly no benefit, in order to measure a change that introduces packed or encrypted data, such as malicious data trying to evade detection.

We introduce, again, two different features based on computing an upper-bound of the Kolmogorov complexity. First, the absolute upper-bound on the tag extracted by our tree difference algorithm; second, the ratio of the upper-bound over the length of the string. In case of very short strings, the upper-bound might even take up more space than the actual string.

3.4.5 Script Inclusion

Scripts included in websites are the most prominent way to infect a user with malware, but they are also used legitimately. Differences exist between how malicious scripts and legitimate ones are used and included. For instance, malicious scripts are rarely including local files; instead, they usually include from an external source or provide the source code directly. The following two binary features model these differences.

Absolute Source URL

The enduring rise of content-delivery networks, which are often heavily relying on load-balancing based on the domain name system (DNS), lead to scripts being included much more often with an absolute and external source address in legitimate cases than it was the case prior to the predominance of these networks (due to potential compatibility issues if scripts are included differently). It is important to understand if a website is hosted on a content-delivery network since it bears the reasoning that these websites are generally much more optimized than personal websites, to save on bandwidth on account of the smaller size. This then has an impact on the importance of other features. This feature is also binary; it is 1 for an absolute non-external source URL and 0 otherwise.

The many legitimate use cases of including scripts from an absolute URL suggest that this feature will not have a discriminatory impact on its own; rather, it supports other features by modeling the inclusion-style in a website. The notion of a single inclusion-style roots in previous work by Nikiforakis et al. [102], which suggests that websites follow the same inclusion patterns, that is, the distribution of how scripts are included is biased toward either relative or absolute inclusions, and only rarely uniform.

External Source URL

While the last feature is a bias function to judge the use of scripts with an absolute source, the next feature is a bias function for the concept of external source URLs. It is important to mention that, if an external script is included, assuming the external domain is maintained by a third party, then the website operator has to trust that the third party providing the external script will not insert any malicious code.

3.4.6 Inline Frames

Similar to the features to model the use of script tags, the following three binary features try to model the inclusion of malicious inline frames, by looking into properties that are uncommon for benign inclusions.

Absolute and External Source URL

Likewise to the nature of the source URL features for scripts, these features give an intuition on the use of inline frames. Both features are identical to their script sibling, but they examine <iframe> tags instead of <script> tags.

Their motivation follows closely the motivation for the script features: that is, the feature for absolute source URLs is supporting other inline frame measures as a bias function. In the past, inline frames with an external source address were often used to include either advertisements or third party widgets. Recently, those moved to inline JavaScript or embedding plugins directly. Adversaries, on the other hand, still use these frames because they allow for easier fingerprinting and support delivering different infection vectors per user, for example depending on the browser's patch-level, installed plugins, or by obfuscating each reply differently. This fine-grained control helps the adversary to maximize the attack efficacy while reducing the likelihood of detection.

Hidden Frame

Beyond absolute and external frames, another indicator for malicious content being included exists: hidden frames. Legitimate frames are generally made visible to the user. Adversaries on the other hand prefer that the included website is invisible, which is why they often resort to setting width and height of the frame to a low value, so that the frame is visually hard to spot for a human. We investigated a random sample of 10,000 inline frame tags that we extracted from our dataset and found that legitimate inline frames are often set to a width and height of larger than 100 and rarely hidden (the style attribute "display: none" is rarely used). We model this phenomenon, assuming that a majority of malicious inline frames uses a much smaller area of screen space, by restricting width and height of our feature to a maximum of 15 pixels. The feature is simply 1 for hidden frames and 0 otherwise.

3.5 Evaluation

Generally, the problem we are trying to solve is an instance of *knowledge discovery in databases* [103, 104]. More precisely, when searching actively for infection campaigns paired with a web crawler, we are interested in detecting outliers, that is, novel changes, and the appearance of clusters, that is, when a new trend is observed, for instance an infection campaign. While various clustering algorithms can be employed, the design of our system encourages the use of an algorithm that detects local outliers. Additionally, the distribution of a cluster can differ from the distribution of any other cluster, particularly for clusters with a low member count, that is, it is not reasonable to assume that all changes follow a very similar distribution in the feature space. Since we are primarily interested in the formation of new clusters, when it is even less likely that this assumption will hold, centroid- or distribution-based clustering algorithms such as k-means (which will give spherically shaped clusters) or expectation-maximization (e.g., Gaussian mixture models) are unlikely to provide any valuable insight on new infection campaigns early enough. To counter this issue, we adopt a variant of the density-based clustering algorithm OPTICS (Ordering Points To Identify the Clustering Structure) by Ankerst et al. [105], namely OPTICS-OF (OPTICS with Outlier Factors) by Breunig et al. [106].

3.5.1 OPTICS-OF

The OPTICS-OF algorithm takes two parameters: the maximum distance for a cluster and the minimal number of vectors necessary to form a cluster. In the scenario of trend analysis, the maximum distance corresponds to the similarity of a change, while the minimal number of vectors necessary describes the number of instances of a change we want to observe before we consider it a trend, that is, before we want to verify that we found a previously-unknown infection campaign.

The algorithm works, in essence, as follows: if two vectors in the feature space are closer than the maximum distance, then they are directly density-reachable. If at least the minimal number of vectors are directly density-reachable from a vector, then this vector is a core object and forms a cluster. A cluster does not only contain directly density-reachable vectors from this core object, but is defined transitively, that is, it contains all vectors that are directly and transitively density-reachable from the core objects. Therefore, an outlier is either not density-reachable to any other vector at all, or only density-reachable to a number of vectors, where none of the vectors is a core object, that is, none of the vectors forms a cluster. In our experiments, we require 10 similarity vectors that are directly density-reachable to form a cluster.

3.5.2 Dataset

First, in this section, we describe in detail what constraints we imposed on our dataset, why these constraints were imposed, and how we obtained our dataset. In general, it is a challenging problem to obtain a representative sample of different and distinct malicious websites. Invernizzi et al. [107] have shown that this is even the case when only mediocre toxicity[2] is required, that is, it is even difficult for a dataset with a small but non-negligible percentage of malicious websites. This poses a problem for collecting our dataset because we desire moderate toxicity and diversity among malicious infection vectors to verify that we can correctly, and without bias, identify similar trends, and by this, similar infection campaigns. Moreover, to discard trivial cases, the requirements on the websites in our dataset are even more restrictive:

- Websites must have been set-up for a legitimate reason, that is, we are interested in landing pages and not interested in exploit pages. Exploit pages denote websites that are set up by an adversary to exclusively deliver malicious code, while landing pages denote the infected legitimate page. We enforce this restriction because recent work establishes that legitimate websites are nowadays the primary target and because other approaches by Provos et al. [23, 92], Ratanaworabhan et al. [108], Curtsinger et al. [109] or Seifert et al. [110] are already able to detect purely malicious websites with outstanding accuracy.
- Two distinct versions of a website are required, that is, a website must have been modified (legitimately or maliciously) to constitute a realistic sample.

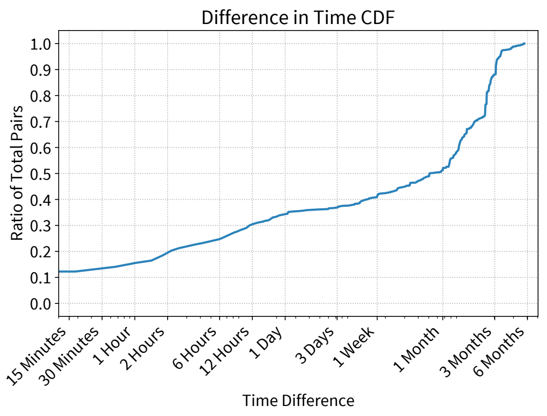
[2]Toxicity measures the maliciousness of dataset and simply corresponds to the fraction of malicious samples in a dataset over all the samples of the dataset.

We obtained our dataset by crawling the web from January 2013 to May 2013 via a 10-node cluster of custom crawlers running an adaptive fetch schedule with a recrawl delay of at least 15 minutes and an exponential back-off delay (multiplied by a constant factor of 10 if no change was observed in a recrawl and with a strict maximum of one week). The hourly seed of URLs for our crawler contained websites that were already present in our dataset and also Yandex's search engines results for Twitter's trending topics. Additionally, to counter the problem of low toxicity and prevent a bias toward benign websites, we injected a total of 2,979,942 URLs of websites into our crawl seed that the Wepawet online analyzer [90] had analyzed previously, starting with samples observed in the beginning of January 2013 and ending with samples observed at the end of April 2013. In total, after removing exact duplicates and restricting the number of pairs per unique URL to a maximum of 10, our dataset spans a size of 700GiB and 26,459,103 distinct website pairs from 12,464,920 unique URLs. A distinct pair denotes a pair where both versions are different from each other in respect to the SHA256 checksum of their normalized DOM tree.

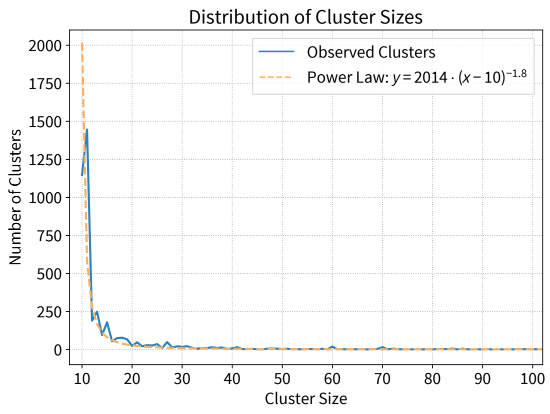
The time difference, before a change between base and current version of a website was observed, is shown in Figure 3.3a. The average time difference of our pairs is four weeks, with 80% of all pairs being 2 hours or more apart, 70% being 12 hours or more apart, 60% being 7 days or more apart, and 50% being 20 days or more apart (median). Since not all websites have been recrawled after exactly 15 minutes, we can only observe that a change happens in at least 16% of the websites in a 15-minute interval after a visit.

3.5.3 Case Studies: Identified Trends/Clusters

In our experiments, we identified a total of 67,038 different clusters, with the majority of clusters having 30 or less elements. Figure 3.3b depicts the final distribution of cluster sizes we observed in our experiments. Evidently, the observed distribution follows closely the power law function: $y = 2014 \cdot (x - 10)^{-1.8}$. In addition, we observe that the total sum over all cluster sizes is less than



(a) Overview of the difference in time between base and current version of a pair in the final dataset



(b) Overview of the number of different elements that are in each cluster

Figure 3.3: Time and distribution properties of the dataset

the number of distinct tags that we have analyzed. This is the case because any remaining tags are still considered to be outliers and do not constitute a trend yet. As a matter of fact, both the close resemblance to a power law function and a non-negligible amount of outliers are expected, because some changes are only made to a limited number of websites, for example, very similar articles might be posted to less websites than we require as a lower-bound to constitute a trend, and also because our view of changes is limited by the seed and link expansion of the web crawler, that is, it is possible that we only observed a subset of the true instances of each unique trend.

We feel that it is important to understand what a single cluster is actually describing, and we provide different examples about what tags have been clustered together. Therefore, we investigate two clusters in more detail. Although both clusters are low-count clusters, that is, relatively small, their small size actually illustrates that Delta does detect when a trend reaches a significant distribution and that it does not rely on an unreasonable large number of observations of a single trend.

The first example we discuss is an actual infection campaign that we have observed in the wild, an instance of a redirection to a Cool Exploit Kit installation. In contrast, the second example we discuss corresponds to a cluster describing the change in cross-site request forgery tokens.

We selected these two clusters manually by filtering clusters based on the generated signature with simple heuristics that suggest malicious behavior, such as external scripts that are included with a random component or JavaScript with a non-negligible ratio of digits over characters (suggesting obfuscation). Clearly, these and other heuristics can also be leveraged to order clusters according to "levels of interest" or to remove clusters that are likely uninteresting and should not be analyzed manually by an analyst.

Other trends we observed, but will not discuss in detail, include the modification of Facebook Like buttons (the backlink URL changes), a version update for the JQuery library served for blogs hosted on Wordpress.com, or the insertion of user-tracking tokens.

Cool Exploit Kit Infections of Discuz!X

One of the most interesting clusters, which shows the applicability of Delta in practice, describes an infection vector used to redirect to a specific infection campaign that uses the Cool Exploit Kit to distribute malware. This in-the-wild infection campaign was found at the beginning of April 2013 in a set of 15 different websites from the following 10 unique URLs:

- http://att.kafan.cn
- http://frozen-fs.net
- http://jses40813.ibbt.tw
- http://ppin888.com
- http://www.dv3.com.cn
- http://www.kxxwg.com
- http://www.ruadapalma.com

```
<script type="text/javascript" language="javascript">
  p=parseInt;
  ss=(123) ? String.fromCharCode : 0;
  asgq=" [4036 character obfuscated string] "
       .replace(/@/g,"9").split("!");
  try { document.body&=0.1 } catch(gdsgsdg) {
    zz=3; dbshre=79;
    if(dbshre) { vfvwe=0;
      try { document; }
      catch(agdsg) { vfvwe=1; }
      if(!vfvwe) { e=eval; }
      s="";
      if(zz) for(i=0;i—1374!=0;i++) {
        if(window.document)
          s+=ss(p(asgq[i],16)); }
      if(window.document) e(s); }}</script>
```

Listing 3.3: Cool Exploit Kit infection vector

- http://www.sdchina.cn
- http://www.wlanwifi.net
- http://www.yysyuan.com

Once we verified that the cluster was indeed malicious, we investigated the underlying commonalities between them. We found that all websites were using the discussion platform "Discuz!X" [111]. Discuz!X is an Internet forum software written in PHP and, according to the Chinese National Radio [112], the most popular Internet forum software used in China. Clearly, these infections are part of the same infection campaign. Additionally, such a strong common ground suggests that the infection is likely to be rooted in a security vulnerability in the Discuz!X software, and it provides support identifying the cause and a removal method.

Listing 3.3 shows the respective generated signature of the infection. For this infection campaign, we did not observe any differences in the tags that were clustered together.

```
<input
 name="csrfmiddlewaretoken"
 value="(JhD3IwCXcnnpRtvE42MN6r8dOBOWRoxG
        |hH4f6eOMCOTEYF0RYoXFRDaTLzym61O2
        [...]
        |DNczoWjeN1nK6nq3whXYpSSnZGdxx0Og
        |F9yLS0jNUXIURsXDRqxS5NVW7qXfWsgf)"/>
```

Listing 3.4: Cross-site request forgery token; | denotes an or

Beyond the inclusions of infection vectors pointing to an installation of the Cool Exploit Kit observed in all pairs, one website (http://frozen-fs.net) also included an infection vector that tried to infect visitors via an installation of the Blackhole exploit kit.

The domain that included the Cool Exploit Kit and the Blackhole exploit kit, "frozen-fs.net," was not cleaned up, and we observed that it was suspended by the provider 27 days after we detected the infection.

Cross-Site Request Forgery Tokens

A second interesting low-count cluster we found during our evaluation models variations in cross-site request forgery tokens in deployments of the Django web application framework. In total, we identified a similar modification among 17 different pairs of websites. Each website used form-based cross-site request forgery tokens and used the same identifier for a hidden form field, that is, "csrfmiddlewaretoken." For every pair, the attribute features did not diverge for the name attribute, while all were different for the value attribute. Nonetheless, Delta clustered them together, since the random entropy was nearly constant for the value attribute among all observed removed and inserted instances. The entropy was nearly constant for the normalized case as well as for the absolute entropy features. The exact identifying signature for that cluster is shown in Listing 3.4.

We feel that this observed trend constitutes a perfect example in which the limitations of the signature generation stick out and where Delta shows its robustness by clustering these changes correctly together. While the signature can detect all observed instances correctly, it is clear that when trying to match new versions of a website with the signature we would fail to identify the changed token value correctly since the value will change to a new, unobserved random value.

3.5.4 Performance

In order to judge the actual applicability of our system in practice, a performance analysis is necessary. We show in Figure 3.4 that the performance of Delta allows for deployments in real-world scenarios. However, corner cases exist that could impact an actual deployment, if the difference between the base and current version of a website is particularly large. We performed a manual in-depth analysis of the system that highlighted the actual performance bottleneck of our system: close to 80% of the time when analyzing the two versions was spent by the Python library BeautifulSoup to parse the HTML structure of a page. Although, the number of changes made to a website plays the most important role in analyzing the changes, pairs that took longer than three seconds to analyze were exclusively websites that sent data in an encoding different than specified. BeautifulSoup tries to take care of this and follows a code path that can get multiple thousand function calls deep, and easily hits the recursion limit of CPython. While we increased this limit in our evaluation to keep these pairs and prevent a dataset bias, the particular functions are actually tail-recursive and, therefore, can be expressed iteratively (thus, removing the necessity of allocating stackframes). However, the abstruseness of the involved functions prevented us from doing the very same in a reasonable amount of time. Regardless of these (still outstanding) engineering challenges for a general deployment, we could analyze a single pair in a median time of 0.340 seconds and in an average time of in 2.232 seconds. It is also evident that we finished each analysis in at most 20 seconds, regardless of the aforementioned problems encountered in BeautifulSoup.

These results, when taking into account that 60% of our data is seven days or more apart (see Figure 3.4), support our claim that Delta does not necessarily need to keep a base version locally, but

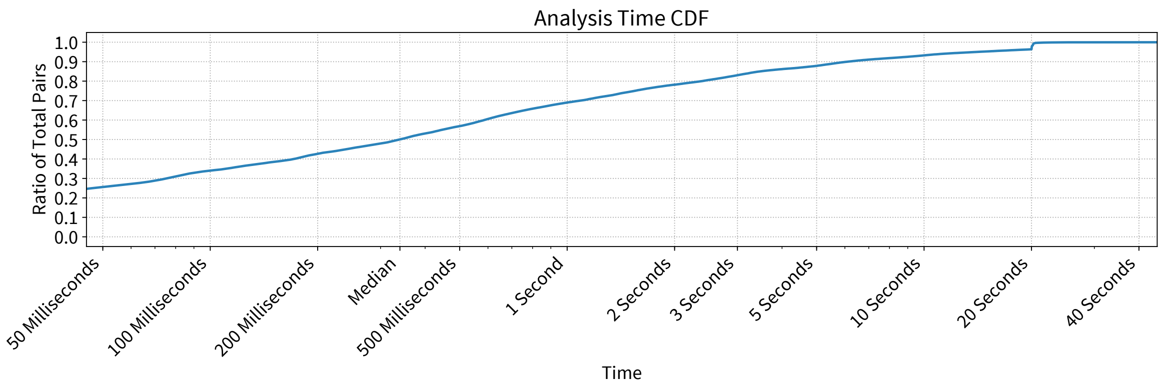


Figure 3.4: Overview of the ratio of website pairs that have been completely analyzed in our experiments in less than x seconds.

could rely on public archives like the Internet Archive or a web cache by a search engine. Nonetheless, we strongly recommend keeping a local version to prevent an additional delay in fetching the website and to prevent running into the problem of a potentially outdated or even non-existing version on the side of the public archive.

3.6 Limitations

Similar to other static analysis approaches leveraging machine learning, our approach has some limitations, which can be used to evade detection. This section discusses these limitations and how they could be managed in a real-world deployment of Delta. First, we introduce a limitation called *Step-by-step Injection*. Second, we will briefly discuss the *Evolution of Infection Vectors* as a major fundamental problem in detecting malicious code. Lastly, we discuss the trade-off between dynamic and static analysis and the limitations of either approach.

3.6.1 Step-by-step Injection

It is possible to circumvent Delta by adding malicious code in small steps, that is, in a series of modifications where each step on its own gets detected as being benign, while the aggregation is malicious. For example, an attacker could build an infection vector that delivers these small steps depending on a cookie or the visitor's IP address to keep track of the client's previous version. However, we argue that the scope of such an attack is heavily limited because: (a) it requires an attacker to be able to inject code that is executed on the server-side, otherwise detection is possible because it has to be done client-side, and the attack can also be impeded or even avoided by keeping the first version instead of updating the stored base version with every visit; or (b) the DOM tree will be modified online, for example through JavaScript. In the first case, an important and drastically scope-reducing factor is that the vulnerability needs to support such an iterative process, where, for example, memory regions are shared among browser tabs or between multiple visits to the same website, which is highly unlikely given the strict separation current browser sandboxes enforce. In the latter case, on the other hand, we suggest analyzing the website on every mutation event,[3] that is, by considering the website that was modified online as a new current version and comparing it to the stored base version.

3.6.2 Evolution of Infection Vectors

Detecting malicious code is an arms race and malicious websites are no exception. Malware developers are trying to evade detection systems to gain the upper hand, while detection system developers are trying to catch and prevent these evasions. Previous work on evasions motivated the search for better and different detection systems [113]. More advanced obfuscation [114], encryption, poly- and metamorphic code [115] and virtualized environments have become more common in response

[3]JavaScript events are called mutation events if they modify the DOM tree, for example, by changing attributes of a node, such as the src attribute of a script tag, or inserting or removing elements from the DOM tree.

to these improvements and impeded detection systems. With more approaches being able to handle these cases, it is to be expected that malware and infection vectors will evolve and successfully circumvent available detection systems. While retraining the machine learning algorithm on a more recent dataset is often a possible approach to counter the evasion problem, it is only a near-sighted solution to counter the dataset shift, as malware will deviate more severely, up to the point where the features will not model the underlying problem anymore. Even in cases where the features are not publicly known to an adversary, it is possible to partially derive these by probing the system carefully, which then will either allow for successful evasion of the system or (on re-training) increase the false positive and false negative rate because of misclassification due to minuscule differences between legitimate and malicious code in the feature space. Both cases are obviously not desired for a detection system, however, it is a general problem of all approaches employing machine learning [92, 23, 116, 117, 118, 110, 91, 33], and it is generally only countered reliably by adapting to a new feature space, which we leave for future work.

3.6.3 Dynamic vs. Static Analysis

Delta in its current form is a purely static analysis system, while, at the same time, the Internet is becoming more and more dynamic. While one might think that static analysis is inferior to dynamic analysis here, this is not the case. Instead, our system complements dynamic analysis systems: it detects trends/infections statically and can forward the interesting trends/infections to dynamic analysis systems that extract further information.

Our motivation to rely on a purely static analysis is based on multiple reasons. First, dynamic analysis is not necessarily useful at the early stage in which our system operates, that is, trends that change the behavior and are interesting to us show themselves first with static content changes, rendering dynamic analysis (currently) unnecessary. A second argument against dynamic analysis for Delta is that it, for instance by instrumenting embedded or included JavaScript to modify the DOM tree to retrieve a “final” version of the website, under-approximates the behavior of the website to this specific execution environment and might yield a potentially incomplete or untrue representation of the DOM tree. It also poses the questions of when to consider the DOM tree “final,” that is, when to take a snapshot. Consequently, it might then be possible to evade the trend detection step in the first place. Additionally, we might also miss infection campaigns that are statically present, but which are removed dynamically or are inactive (for us). For example, servers could be unavailable (for us) or code might not be loaded (for us), we could be fingerprinted, the IP address of our analysis system might be in a region of the world that is not affected or simply because the user-agent of our browser does not match a (unknown) regular expression. The third argument in favor of static analysis is that it can be considerably faster than dynamic analysis, which, in turn, allows us to leverage more computationally-expensive features to increase trend detection accuracy.

Lastly, while the trend detection step is purely static, to detect malicious behavior, Delta relies on an external analysis system that might very well use dynamic analysis. Generally, we do not impose any limitations on this detection engine but that it can detect malicious behavior.

3.7 Conclusion

In this chapter, we introduced Delta, a novel, light-weight system to identify changes associated with malicious and benign behavior in websites. The system leverages clustering of modification-motivated features, which are extracted based on two versions of a website, rather than analyzing the website in its entirety. To extract the important modifications accurately, we introduced a fuzzy tree difference algorithm that extracts DOM tree nodes that were more heavily modified, discarding changes in single characters or words, or legitimate evolutions. Beyond detecting if a change made to a website is associated with malicious behavior or not, we showed that Delta supports the detection of previously-unknown infection campaigns by analyzing, unknown trends and measuring the similarity to previous, known infection campaigns. Furthermore, we showed that the system can generate an identifying signature of observed infection campaigns, which can then be leveraged to protect users via content-based detection systems or as test-cases for online analyzer systems. Ultimately, the system's ability to identify specific infections is helpful in identifying the reason why the website was infected by a specific campaign in the first place, such as a distinct version of the web application among all infections; additionally, it facilitates the removal of malicious code and the mitigation of additional infections in the future.

Chapter 4

Mitigating the Risks of Takeover Attacks and Domain-Validated Certificates

Infrastructure-as-a-Service (IaaS), and more generally the "cloud," like Amazon Web Services (AWS) or Microsoft Azure, have changed the landscape of system operations on the Internet. Their elasticity allows operators to rapidly allocate and use resources as needed, from virtual machines, to storage, to bandwidth, and even to IP addresses, which is what made them popular and spurred innovation.

In this chapter, we show that the dynamic component paired with recent developments in trust-based ecosystems (e.g., TLS certificates) creates so far unknown attack vectors. Specifically, we discover a substantial number of stale DNS records that point to available IP addresses in clouds, yet, are still actively attempted to be accessed. Often, these records belong to discontinued services that were previously hosted in the cloud. We demonstrate that it is practical, and time and cost-efficient for attackers to allocate IP addresses to which stale DNS records point. Considering the ubiquity of domain validation in trust ecosystems, like TLS certificates, an attacker can impersonate the service using a valid certificate trusted by all major operating systems and browsers. The attacker can then also exploit residual trust in the domain name for phishing, receiving and sending emails, or possibly distribute code to clients that load remote code from the domain (e.g., loading of native code by mobile apps, or JavaScript libraries by websites).

Even worse, an aggressive attacker could execute the attack in less than 70 seconds, well below common time-to-live (TTL) for DNS records. In turn, it means an attacker could exploit normal service migrations in the cloud to obtain a valid TLS certificate for domains owned and managed by others, and, worse, that she might not actually be bound by DNS records being (temporarily) stale, but that she can exploit caching instead. We introduce a new authentication method for trust-based domain validation that mitigates staleness issues without incurring additional certificate requester effort by incorporating existing trust of a name into the validation process. Furthermore, we provide recommendations for domain name owners and cloud operators to reduce their and their clients' exposure to DNS staleness issues and the resulting domain takeover attacks.

4.1 Motivation and Contributions

Over the past ten years, cloud services have grown tremendously. Generally, clouds are comprised of hundreds to thousands of commodity servers, which make up pools of computing resources that are shared by different users. One of the main drivers behind the clouds' rise in popularity is their elasticity: Users can acquire and use resources as needed, on demand, and at scale, all while requiring almost no upfront investment. In fact, Amazon Web Services (AWS), Amazon's public cloud, serves over one million active users worldwide [119], Microsoft Azure is gaining 120,000 new customers each month [120], and the global cloud IP traffic has reached 3.9 zettabytes (3.9 billion terabytes) in 2015 already [121]. Unfortunately, as the recent years have shown, the resource pooling and increased popularity of cloud-based deployments also pose severe security issues to the clouds' tenants [122, 34].

With the clouds' increase in popularity and their commoditization, website operators have been empowered to deploy their website themselves instead of relying on more traditional web hosting. At the same time, HTTPS has become basically a requirement for any website operator, not only for dynamic websites trying to protect login credentials, but also for static websites. Unprotected websites are being ranked lower by search engines [123], they are limited in browser features that they can use [124], and they risk having content and advertisements injected, for example, by wireless access point operators or Internet Service Providers [125, 126]. For HTTP/2, it has become practically mandatory because all major browsers support HTTP/2 over TLS only [127]. Website operators now typically deploy TLS certificates for their domains and use HTTPS to ensure integrity and confidentiality for any communication with their website. For certificates to be trusted by the websites' visitors' browsers, however, they need to be issued by trusted certificate authorities (CAs). Traditional verification approaches involve identity documents, like verifying passports, which incurred high processing overhead. To cope with the high-volume demand for digital certificates, CAs adopted automated approaches to verify and issue certificates, and now heavily rely on domain-validation. Having launched only in April 2016, Let's Encrypt has since been dominating the domain-validation part of the certificate authority ecosystem through openly available and well-designed tooling that uses the Automatic Certificate Management Environment protocol (ACME) [128] to validate domain ownership and issue certificates almost transparently for users. Today, Let's Encrypt has issued over 100 million certificates in less than 15 months and their certificates account for 80% of all publicly trusted certificates [129, 130].

Unfortunately, combining the elasticity of cloud infrastructure and the automation of certificate issuance introduces new security vulnerabilities. We discover that stale and abandoned DNS entries pointing to cloud IP addresses can be exploited by attackers to deceive domain-based certificate validation and obtain certificates for the victim domains. The problem stems from the ephemeral nature of the cloud resources. More specifically, if a user releases a cloud IP address, but does not remove the corresponding DNS entry before releasing the IP address, an attacker can allocate the same IP address, impersonate ownership of the domain, and request trusted certificates from a CA, like Let's Encrypt. We call them IP address use-after-free vulnerabilities, which can enable a variety of attacks and cause harm. Adversaries can leverage the acquired valid certificates for man-in-the-middle attacks, for example, to intercept the HTTPS traffic to the victim domain on a wireless network. Worse, if an attacker obtains a wild-card certificate, her attack capabilities are significantly enhanced, possibly allowing her to impersonate any sub-domain, including non-existing ones. The obtained certificates can be abused for phishing attacks, by impersonating the legitimate website, including TLS verification and its "trustworthy green lock." Attackers can deface the website, and they might even be able to launch remote code execution attacks, for example, if JavaScript or native code is being loaded from the domain that was taken over [6, 102, 131].

To better understand the prevalence of IP address use-after-free vulnerabilities in the wild, we conduct a large-scale analysis. From passive DNS traffic, we extract over 130 million domains that point to IP addresses of cloud networks. On these domains, we perform regular liveness probes to determine whether their cloud IP addresses are allocated and in use. Our results indicate that over 700,000 domains point to cloud IP addresses that are free, and which are susceptible to domain takeover attacks due to use-after-free vulnerabilities. We further investigate the feasibility of obtaining particularly interesting target IP addresses from cloud services, and we estimate that it would cost attackers less than $1 (USD) to cycle through the necessary unique IP addresses, which renders the attack economically viable for adversaries. Based on our in-depth analysis, we propose to extend the ACME protocol version 2 by including our new trust-based identifier validation challenge, and we provide practical recommendations for domain owners and cloud operators to protect themselves from domain takeover attacks.

We make the following contributions:

- We conduct a comprehensive study of IP address use-after-free vulnerabilities, and the domain takeover attacks that these vulnerabilities enable. We show that the scale of the vulnerabilities is considerable: Over 700,000 unique domains point to IP addresses that are free and can be abused to take over the respective domains.

- We discover that even well maintained DNS zones can be vulnerable to domain takeover attacks: After releasing cloud IP address resources, an adversary might be able to exploit now outdated zone information in DNS caches to launch attacks.

- We examine the feasibility of launching domain takeover attacks in the real world through cloud IP address re-use, by analyzing their allocation cycles, and we show that it is practical, time-efficient, and cost-efficient for an attacker to launch such attacks.

- We propose a new domain-validation method for automated certificate management environments (ACME) CAs that leverages the existing trust of a name to mitigate domain takeover attacks.

The remainder of this chapter is organized as follows: First, we provide background detail on DNS, operation of Infrastructure-as-a-Service clouds, and domain validation (Section 4.2). Next, we analyze and evaluate to what degree IP address use-after-free vulnerabilities pose a security threat (Section 4.3). Then, we present our mitigation technique, which retains almost all usability benefits of automated domain-validation, yet protects against IP address use-after-free attacks (Section 4.4). Finally, we conclude (Section 4.5).

4.2 Background

We provide a basic introduction to the Domain Name System (DNS), to different operational models in cloud setups, and to the use of domain-validation for TLS certificate issuance.

4.2.1 Domain Name System and DNSSEC

The Domain Name System (DNS) is a core protocol of the current Internet architecture. It facilitates to use easily identifiable hierarchically organized names instead of IP addresses to access services online. Although the fundamental idea of DNS is straightforward [132], we describe IPv4 and IPv6 resource records (RRs) and DNSSEC as they are essential.

Resolving names to IP addresses via DNS is done by requesting an A RR to resolve a name to an IPv4 address, or an AAAA RR to resolve to an IPv6 address. The information for a RR is stored in the so-called parent zone. Each record is served by (at least one) DNS server, which is authoritative for that zone. There is, however, no automatic aspect within the DNS ecosystem that guarantees that DNS entries remain *fresh,* that is, a method that ensures that a given RR never becomes *stale,* but that it always points to the correct IP address or that it is removed if it should point nowhere.

DNS by itself does not provide authentication, which brings security issues due to response spoofing, and spoofing can allow domain takeover attacks. DNSSEC is one method to provide integrity for the unencrypted DNS ecosystem. Authenticating existing records is a straightforward extension of DNS through a signature record type (RRSIG) for each original resource record set (RRset), which is signed with a zone-signing key (ZSK). The public key portion of the ZSK is hosted in the zone, while the parent zone provides a hash of the ZSK in a DS RR. The problem of distributing public keys in a trustworthy manner is solved through DNS' hierarchical nature and its existing chain of trust from the root zone to the queried zone. Crucial is that DNSSEC discourages the use of online signing to prevent denial of service attacks against the nameserver and chosen-plaintext attacks against the zone-signing key, as well as deploying the ZSK to (hidden) master nameservers to automate signing of updated zone information online [133, Section 5]. Instead, it strongly encourages to publish only zone information that was signed offline in a secure manner, and then deployed to (hidden) masters [134, Section 3.1, Section 9, and Section 12][135, Section 3.4.3]. Furthermore, the current state of the DNSSEC ecosystem shows significant deployment issues, for example, not publishing all required records for validation, incorrectly rolling-over keys, or not rolling keys over in the first place, which indicates a lack of care or tooling when deploying DNSSEC in practice [136].

4.2.2 Cloud Models

Cloud Computing has become a widely used concept in Computer Science. Following, we employ the National Institute of Standards and Technology's (NIST) definition of Cloud Computing [137].

Clouds are hardware and software bundles to provide users with five basic characteristics: *on-demand self-service*, *broad network access*, *resource pooling*, *rapid elasticity*, and, *measured services*. Specifically, it means that a cloud must provide services at its users' demand, without requiring any further manual interaction by the cloud operator, it must allow customers to (ideally) automatically scale their resource usage based on their needs, and all operations must be metered precisely and billed accordingly.

Cloud infrastructures generally have different deployment models, depending on their use case and users: *public* for the general public, *private* for large operators or higher security requirements (e.g., businesses or the government), or *community* for private clouds shared among multiple organizations for cost-savings or security. We focus on IP address re-use vulnerabilities in public clouds.

The most distinguishing technical difference for clouds is their respective service model:

Software as a Service (SaaS).

The SaaS model is the most abstract setup. Customers interface with software provided by the operator, either via their web-browser or a standardized program interface (API). Customers do not have access "*the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities [...]*" [137]. Examples include Microsoft Office 365 and the SalesForce Platform.

Platform as a Service (PaaS).

For PaaS clouds, users deploy their own code and applications to run on the cloud. Although the executed code is under the users' control, access to the underlying cloud infrastructure, like network and disk, is similarly restricted as in the case of SaaS clouds. Examples include Heroku and Google App Engine.

Infrastructure as a Service (IaaS).

IaaS clouds, on the other hand, give more control to cloud users. Here, a user can freely request storage, network, memory, processing, and other resources as needed. Commonly, these resources are provided to the user in form of a virtual machine (VM), on which the user can install any operating system and software. Popular examples of IaaS clouds are Amazon Web Services (AWS) EC2 and Microsoft Azure.

We investigate IaaS clouds because they allow us to freely and rapidly allocate IP addresses as part of their resource pooling characteristic. Depending on the external interfaces of PaaS clouds, they may also be vulnerable to re-use attacks, which are related to the IP address use-after-free vulnerabilities that we describe.

4.2.3 Domain-Validated Certificates

The HTTPS ecosystem is based on certificate authorities (CAs), which are trusted by operating system and browser vendors. These vendors include the CAs' certificates in their products, and certificates that are presented to clients have to demonstrate a chain of signatures to a certificate of a trusted

CA. The job of a CA is to verify that the entity that requests a certificate to be issued is authorized to obtain a signed certificate for the specific domain(s) that the certificate is supposed to be valid for.

Various methods to assert authority over a domain exist. Classical and more expensive methods of identification require a CA to verify that a requesting party conforms to the domain-owning party by checking identity documents, for example, passports, or company incorporation forms. However, such processes incur significant overhead.

Nowadays, more cost-effective methods of validating domain ownership, or rather establishing that the requesting party is currently controlling the domain, exist, and they have been adopted by all major CAs, mainly to combat operating costs. These methods are generally referred to as issuance of a domain-validated certificate, because only authority over the domain is established. The three most common validation methods are:

DNS Validation.

To validate ownership of a domain via DNS, the certificate requester must set a nonce that she received from the CA in a DNS record, usually a TXT record, which the CA will attempt to query and validate. Requiring the requester to change a DNS entry implies that she controls the domain's DNS zone, which is considered a strong indicator for authority over a domain.

Email Validation.

Similarly, to validate a domain via email, the CA sends an email to (a) one of the mail addresses listed in the domain's WHOIS data, or, (b) to one of the common administrative email accounts, like "postmaster," "webmaster," or "sslmaster." The email includes a unique token that must be send to the CA, or a unique link that needs to be visited to verify ownership of the email address, and, in turn, the domain.

Web-based Validation.

For web-based validation the certificate requester receives a token from the CA that she must make available via HTTP at a CA-specified path on the domain for which the certificate was requested. Once made available, the CA verifies that the token is accessible and contains the correct value, and only then attests ownership of the domain and issues certificate.

Traditionally, CAs were dominated by an enclosed and business-oriented community. CAcert was among the earliest and most prominent approaches to introduce a community driven CA effort [138]. Unfortunately, due to insufficient support by browser and operating system vendors, it never reached widespread adoption. Furthermore, the recent rise of TLS related incidents, for example, DigiNotar [139] and CAs issuing illegitimate certificates [140], lead to two new developments trying to disrupt the established CA ecosystem: the wide-spread introduction and requirement of certificate transparency and the Let's Encrypt CA.

Certificate transparency is a framework that specifies that a CA must publish to a tamper-proof, append-only log, which can be audited by authorized parties [141, 142]. Its purpose is to allow potentially affected parties, for example, domain owners, to verify that a CA has not issued a certificate for a given domain to an unauthorized party. In an ideal world, all CAs would participate in this scheme and publish certificate transparency logs, but, unfortunately, not all CAs do currently participate. However, some individual CAs have been forced to publish transparency logs by browser vendors, most notably Google, who threatened to void their trust in the CAs and to remove the CAs' certificate from their products if the CA does not comply with Google's request. Without a doubt, the removal of a CA from a major browser, such as Google Chrome, would have severe business and financial consequences for a CA, as it might have to refund cost for already issued certificates and it would likely have difficulty acquiring new customers, which is what forces a CA into compliance and why it is willing to participate in the certificate transparency scheme. One example of such an occurrence is Symantec, who has been required to publish certificate transparency logs after they issued certificates for google.com without Google's authorization [140].

Let's Encrypt, on the other hand, is an effort to make TLS encryption more prevalent on the Internet. They practice a leaner and completely automatic identity verification process, and they only issue certificates with short lifetimes of 90 days, to limit the potential damage of key compromise and mis-issuance, as well as to encourage automation [143]. Contrary to the most other CAs, Let's Encrypt issues certificates free of charge, and identity is verified exclusively via web-based validation and through DNS validation. Thanks to a combination of a browser-trusted certificate, being free of charge, and software tooling openly available to reduce system administrator effort, it has led to a significant increase in the number of systems on the Internet which use validly signed certificates, as well as it increased Let's Encrypt's popularity and market share [144].

4.3 Problem Analysis

Mitigations to protect from security problems can be implemented with varying degree of complexity, and for problems of varying degree of complexity. However, in practice, these security measures bear performance overhead and have usability drawbacks, which might not be acceptable. In turn, their actual real-world deployment depends on security risk evaluations, operational costs, and human costs. Therefore, before trying to mitigate a non-issue, it is necessary to justify them with supporting data instead of recommending absolutes.

Following, we first discuss the different security issues in respect to use-after-free vulnerabilities for IP addresses in respect to DNS-based domain validation. We then evaluate to what degree those security issues are practical to exploit. Finally, we estimate how many domains might be susceptible to takeovers and whether protecting them is worthwhile.

For our problem analysis, we investigate and interact with systems that are online and in-use by third parties. Naturally, those systems are outside of our control. In turn, our analysis poses ethical challenges to not affect or impact the legitimate users of such systems in any way. We discuss the considerations we undertook for an ethical and appropriate, yet realistic, analysis separately for each experiment in their respective sections.

4.3.1 Impact

Domain takeovers bear serious consequences, even temporary takeovers can provide ample opportunity for an attacker (Section 4.1). Naturally, the way an attacker might cause harm to the legitimate domain operator and domain users varies from case to case and the space of attacks is vast, which is why we only discuss a subset of possible attacks:

Malicious and Remote Code Loading.

Likely the most straight-forward way for an attacker to turn a profit through a domain she took over is by serving malicious code, serving advertisements, or including affiliate marketing [1, 102, 131]. Although considered easier to launch for websites, the attack is not restricted to websites. Instead, an attack could also be launched on mobile or desktop applications, for example, through remote code loading [145, 146]. Unfortunately, HTTPS and HSTS themselves do not mitigate such an attack.

TLS Certificates.

Another way for an attacker to leverage a domain takeover attack or to increase its success chance is by requesting a TLS certificate that is trusted by operating systems and browsers. Requesting a trusted TLS certificate has become practically feasible because of domain-validated certificates, such as Let's Encrypt. Once she has obtained the certificate, she has increased capabilities for remote code loading attacks over HTTPS, even including HSTS.

Nameservers.

A domain might also point to a nameserver, where the domain server can be for the same domain or different ones. In practice, these cases occur because DNS demands multiple nameservers for redundancy, and if a nameserver does not respond, a client automatically and, transparent to the user, retries queries with fail-over nameservers. Therefore, a domain pointing to a free IP address for a nameserver only incurs a latency penalty and is barely noticeable by the user. However, an attacker could take over the entire domain and even create additional domains. For a domain owner, taking over a domain that is being used as nameserver equates to the worst case scenario. Unfortunately, even entire top-level domains have been vulnerable to nameserver domain takeover attacks [147].

Email Servers.

Similarly, after gaining control over a domain, an attacker might be able to send and receive emails. Importantly, a DNS MX record is not required: if a domain has no MX record set, then its respective A record is being used. Acquiring the capability to send or receive email allows an attacker to abuse a domain for spear-phishing and phishing campaigns, such as CEO email scams, or to recruit victims for fraudulent schemes [148, 7].

Sub-domain Attacks.

Finally, top-level domains are not the only worthwhile takeover targets for an attacker. Sub-domains are at least similarly interesting for attacks, even sub-domains that might have never been used in production, as they could still be abused for authentication bypass vulnerabilities, for example, like it recently happened to the ride-sharing company Uber [149].

Regarding TLS certificate related attacks, it is sufficient for an attacker to request an ordinary certificate. She does not require a wild-card certificate to launch successful attacks. However, if an attacker can obtain a wild-card certificate, her capabilities are significantly extended. For example, if she can receive a wild-card certificate for “support.example.com,” she would then be able to impersonate, intercept traffic to any sub-domain of “support.example.com,” and even launch sub-domain related attacks at the main domain “example.com” [149]. Although, currently, wild-card certificates are not supported by free domain-validated certificate authorities, like Let's Encrypt or StartCom, at least Let's Encrypt is planning to support them as early as January 2018 [150]. Furthermore, wild-card certificates are already supported by other mainstream CAs, such as Comodo. While they charge a fee, they allow significantly longer validity periods of up to three years, which can make attacks even more disastrous.

4.3.2 Taxonomy

For a precise classification of how IP address use-after-free vulnerabilities are being rendered possible, we distinguish four different cases in which a domain points to a free IP address (i.e., the domain is stale) through the following taxonomy:[1]

Early Migration.

A domain-IP mapping is *migrating early* if the domain is in use by the operator, and the records at the authoritative nameserver have been updated to point to the new IP address before the old IP address is being released and available for others to request and use.

Delayed Migration.

Similarly, a domain-IP mapping is *migrating with delay* if the domain is in use by the operator, and the records at the authoritative nameserver have *not* been updated yet, that is, they point to a released IP address.

Auxiliary.

Differently, a domain-IP mapping is *auxiliary* if the domain is used by the operator, and the domain has multiple records, which point to both current and old IP address, possibly in a way so that the old and free IP address would only be used as in a fail-over scenario and has otherwise no practical impact.

[1]Our study focuses on TLS certificates, web servers, domain-validation through HTTP, and type A DNS records. However, our findings also apply to other record types, for example, MX or NS.

Abandoned.

We define a domain-IP mapping as *abandoned* if the domain is not used legitimately anymore. For example, a company might become defunct and is not operating the service anymore that was previously offered at the domain, but it retains ownership of the domain until its expiration.

Depending on how the domain becomes stale, the length of the window of opportunity differs. In case of an early migration, an attacker has the shortest window of exploitation: the cache lifetime of the domain IP mapping. Note, however, that the time a domain IP mapping might be cached is not strictly its time to live (TTL) as set by the authoritative nameserver. The mapping can be purged from the cache before its expiration, and a caching nameserver might ignore the TTL entirely and cache entries longer, for example, for performance reasons, though in violation of the DNS RFC [151]. Theoretically, early migration could prevent IP address use-after-free attacks under the assumption that no intermediate nameservers cache entries longer and that the IP address is released only after the TTL has expired. Practically, unfortunately, human error results in domains not always migrating early and intermediate nameservers might ignore the TTL. Therefore, even those domains migrating early can be at risk of temporary domain takeovers.

From a security standpoint, the remaining three classes are more worrisome. The easiest case to launch a successful attack against is an abandoned domain: the attacker is not rushed by the legitimate operator, and she can wait until an opportunity arises. Fortunately, it is also the least interesting case for an attacker because users are not expected to contact the service at the domain regularly anymore but only sporadically (e.g., through an outdated bookmark for a website), thus, the number of potential victims is generally low.

For domains that migrate with delay, the window of opportunity to validate ownership of a domain is fixed in time and often short. While an attacker could miss the window, she can lurk and wait for a target domain migrating with delay by repeatedly trying to allocate the target IP address, which we later show is practical (Section 4.3.3). More important, once the window of opportunity has passed, the successfully validated domain is not useless to the attacker even though she has no control over the host with the IP address behind the domain-IP mapping anymore (it is now a new IP address, which is not under the attacker's control). For instance, in case of domain-validated TLS certificates, once an attacker validated that she owns the domain, she can later leverage the obtained certificate for man-in-the-middle attacks, for example, for a wireless network at a coffee shop, because the certificate is trusted by major operating systems and browsers. Here, the number of victims is larger than in the case of abandoned domains, but seldom substantial. The core problem with domain-IP mappings that are migrated with delay lies in the long-term capabilities granted to the attacker.

Auxiliary domain-IP mappings are the most troublesome case: they provide a constant window of opportunity and can cause the most havoc. First, an attacker can remain stealthy as a "fail-over" until a viable opportunity arises. During normal operation, the attacker's machine does not respond or it redirects all traffic to a legitimate host. Second, an attacker can force a fail-over to the IP address under his control by launching a denial of service (DoS) attack against the legitimate hosts. However, even without forcing a fail-over, an attacker will see a subset of traffic due to implicit round-robin in DNS, which occurs because DNS records have no implied order. Upon forcing fail-over, the attacker forces a domain-validation service to connect to the host under the control of the attacker, as no other hosts are responsive. Correspondingly, without forcing a fail-over, the attacker might need to try multiple times until the domain-validation service connects to the address under her control and, in turn, validates her ownership of the domain. The attacker can verify ownership of the domain successfully in both cases, for example, to request a certificate, and a significant number of users will connect to the attacker's machine (all or a subset due to DNS' round-robin). Overall, auxiliary domain-IP mappings can affect the most victims and it can provide ample opportunity to cause harm, for example, to visitors of a website by injecting malicious code.

After we classified the reasons for why IP address use-after-free vulnerabilities exist and what their impact can be, the immediate next question becomes: can an attacker actually exploit these vulnerabilities in practice, by allocating the same IP address the victim has freed?

4.3.3 IP Address Churn

An attacker can successfully exploit an IP address use-after-free vulnerability in practice if she can get a cloud provider to assign the recently freed IP address to herself within the window of opportunity. Following, we determine whether it is practical for Amazon Web Services (AWS) and Microsoft Azure, the two largest cloud providers today [152].

Specifically, we repeatedly allocate and free IP addresses in succession. To prevent starvation, we are using a slow allocation cycle to not interfere with the clouds' operations: We request five IP addresses per region, freeing them immediately, and then sleeping for 10 seconds, that is, effectively allocating 1 IP address every two seconds. We performed our IP address churn experiment from April 29, 2017 01:03 UTC to June 6, 2017 23:27 UTC spanning all regions of the cloud providers at the time for a total cost of $31.06 (USD). Over the course of our measurements, we cycled through a total of 14,159,705 allocations of 1,613,082 unique IP addresses. As we always first released addresses before allocating the next batch, we cannot cause address starvation. This is highlighted by us always receiving an IP address upon issuing an API request.

The success of our technique depends on how fast we can iterate through the pool of free IPv4 addresses for a given availability zone. This depends on the overall size of the pool, and its variance, that is, how fast addresses are allocated by other users. To illustrate these characteristics for each availability zone, we investigate the churn (see Figure 4.2) and time between allocation of the same address (see Figure 4.1). We show only AWS specific plots in the pursuit of brevity and comprehensibility, as Azure is not behaving significantly different.

Using the churn plots in Figure 4.2, we get an overview of change in the IaaS cloud's IP pools. Figure 4.2 shows the churn in allocated addresses for AWS, that is, for each day we allocated addresses we plot the fraction of addresses we previously allocated and the fraction of addresses we did not previously receive as an allocation. Dates without data relate to dates where either the IaaS provider conducted maintenance operations, or our measurement scripts were not yet running for that zone.

The natural pattern we expect for the churn plots is an initially high share of new addresses while the pool is being initially explored. This pattern should then slowly approach a stable socket, which corresponds to those addresses that are handed back to the pool by other tenants. Indeed, we find this pattern in our data. For example, Figure 4.2a and Figure 4.2g show this expected pattern. However, these zones have a relatively large pool of addresses that are free at any given time. Zones like eu-west-2 (see Figure 4.2i) are significantly smaller, hence converge more quickly. This furthermore underlines that the allocation algorithm must, in some form, iterate through the whole pool of addresses, instead of just allocating the (same) first free addresses.

In addition, we also find a couple of interesting events: Zone ap-southeast-2 (see Figure 4.2e) started off similar to eu-west-2. However, at the beginning of week 20 in 2017, a large batch of free addresses was added to the pool, leading to a "restart" of the churn pattern. In eu-west-1 (see Figure 4.2i) and us-east-1 (see Figure 4.2k) we see the effect if several days of not iterating through the pool: As soon as we restart our allocation script, we observe a slight rise in new addresses, which have accumulated during the time we did not perform measurements. We find the last notable pattern in us-west-2 (see Figure 4.2n). Here, a substantial amount of so far unseen addresses is released to the pool in the middle of each week.

Next, we take a look on how long it takes to iterate through the whole pool, that is, how fast an attacker could obtain a specific address. For this, we look at how much time passes on average, until an address is allocated for the second time. Given our earlier observation that we do indeed circle through the IP pool, we expect the mean to correspond to the point where we iterated through the

IP address pool. This is summarized with boxplots (without outliers) in Figure 4.1. We find that with our ethically restricted approach most pools are exhausted within under a day. Only the largest, like eu-west-1 and us-east-1 reach means significantly over a day.

Although we used a slow allocation cycle to not interfere with the clouds' operations, an attacker is not bound by the same ethical standard. Practically, the attacker will be bound only by the response time of the IP address allocation API endpoint and her network latency to it. Therefore, an attacker can cycle through available IP addresses much more rapidly. In fact, considering the AWS API limit (10,000 requests per second [153]) and the number of requests needed to exhaust pools in our experiments, an attacker would only need between two and 61 seconds to acquire a target IP once the victim has freed it, using a rapid allocation cycle of 5,000 IP allocations per second. In practice, this theoretical limit is not necessary for an attacker to launch a successful attack. For example, DNS cache times are almost always five minutes, and often much longer with 60 minutes to multiple hours, thus, allowing an attacker to be successful by exploiting caching effects with rates of less than 50 IP address allocations per second.

4.3.4 Affected Domain Names

Considering the worrying high-rate of IP address churn for major cloud providers and low opportunity cost for an attacker to launch an attack, the only question that remains unanswered before we can determine whether temporary stale domains pointing to readily available IP addresses are a problem in practice is whether a significant number of domains are affected?

For a better understanding of how many domains are affected by IP address churn, we observe DNS traffic through Farsight's passive DNS measurements [154]. The Farsight passive DNS dataset is provided through a continuous data feed. For our collection and DNS data analysis, we follow established best practices for collecting and handling Internet measurement data [155], we anonymize

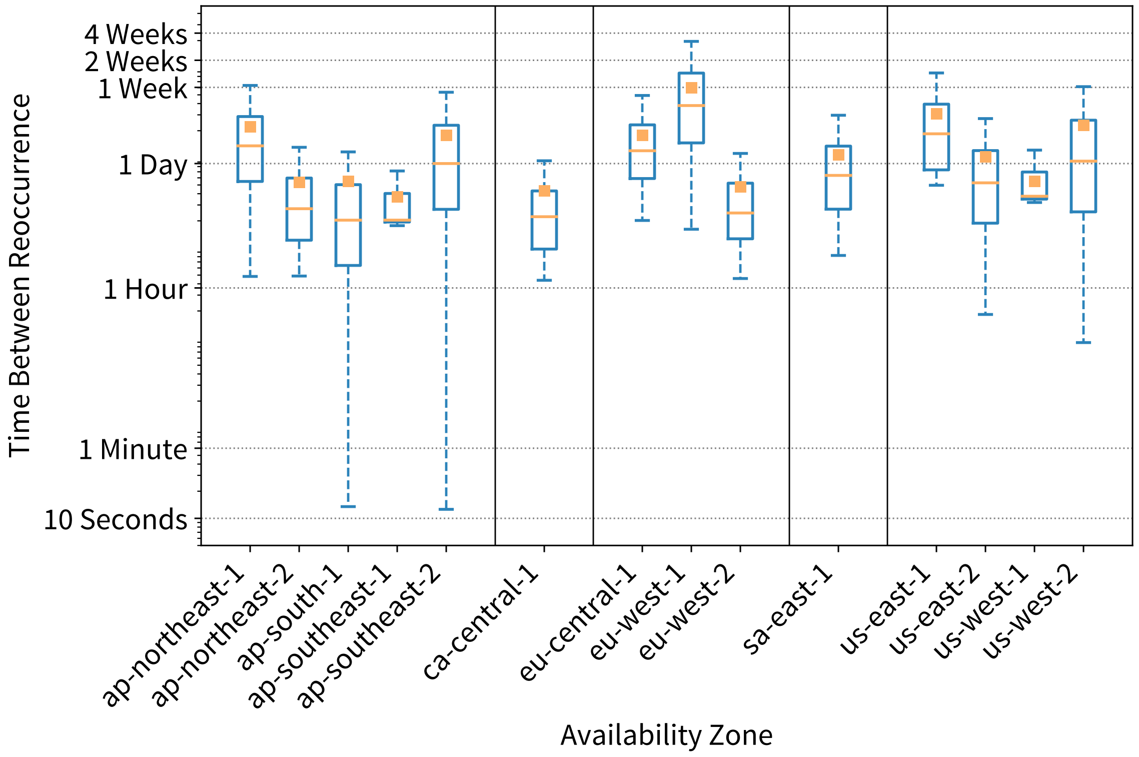


Figure 4.1: Time between allocations of the same IP address per AWS EC2 availability zone

all incoming data immediately by removing any resolver information, and we only retain successful DNS responses.

Specifically, we collect all DNS responses containing A records pointing to the Amazon Web Services (AWS) EC2 cloud, the Microsoft Azure cloud, and the Digital Ocean cloud spanning exactly 120 days from April 11, 2017 0:00 UTC to August 9, 2017 0:00 UTC. Overall, we extract and analyze 130,274,722 unique domains with 767,108,850 unique domain-IP mappings, counting also sub-domains. Including sub-domains is important for completeness, however, it makes an accurate comparison to top domain lists (e.g., Alexa), to estimate the domains' popularity, difficult, because they do not include sub-domains. Matching at the second-level of a domain is similarly problematic due to potentially over-estimating the impact of ephemeral sub-domains and the loss of information

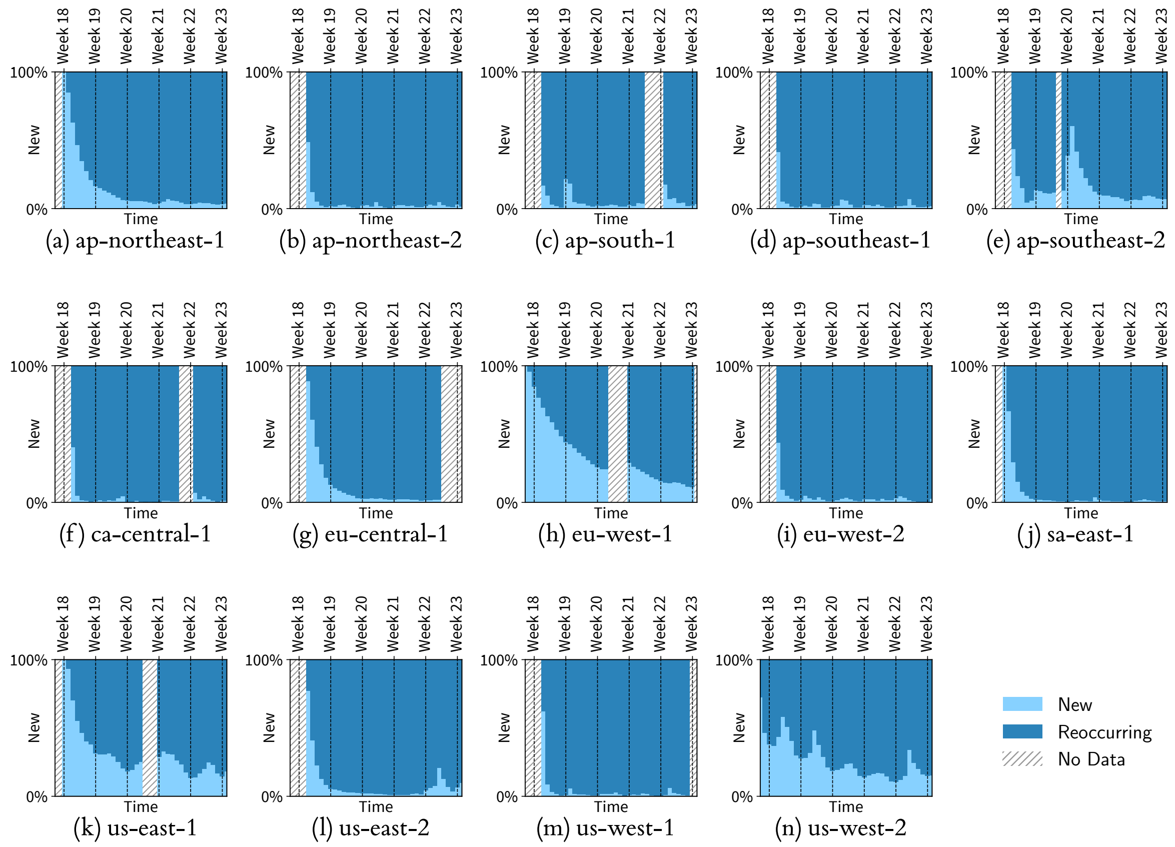


Figure 4.2: Share of newly-observed IP addresses (churn) per AWS EC2 availability zone

on sub-domains of special second-level domains, such as .ac.nz or .co.uk. It remains for future work to evaluate the distribution of DNS zone staleness in regard to domain popularity.

We perform our evaluation on a Kubernetes cluster comprised of 656 processor cores and 3,020 GiB memory, and which is connected at a dedicated 10 Gbps Internet uplink.[2] For each domain, we

[2]The cluster is on a network separated from the main network of the institution at which the experiments are performed. The network traffic generated for our evaluation is not subject to packet introspection, which would have had a negative impact on our measurements.

test every six hours[3] from June 10, 2017 0:00 UTC to August 9, 2017 0:00 UTC (60 days) whether the IP address is still in use or if it might be freed and available:

1. We resolve the domain and check if the IP address the domain points belongs to a network of a cloud provider.[4] If the domain points to a cloud IP, we test if the IP address is responsive and whether it might be free and available to others. If it does not point to a cloud provider or does not exist anymore, we do not perform any further tests.

2. We test if the IP address responds to ICMP ping requests, or responds to any packet sent on 36 of the most frequently used TCP and UDP ports (see Table 4.1) [156] within a two seconds timeout.[5] If we receive a response to any of our requests, we mark the IP address as online and allocated. Correspondingly, if we receive no response until the timeout is reached, we mark the IP address as offline and freed.

Naturally, ingress firewall rules could prevent our test from succeeding and, thus, our estimation is an upper-bound. One might expect it to be a gross over-approximation because cloud virtual machines instances have traditionally received public IP addresses. Nowadays, however, this is not necessarily the case: cloud instances that do not need a public IP address can and generally do live in cloud-only internal networks. Furthermore, by default, many machines respond to ICMP ping requests or allow for secure shell (SSH) access via TCP on port 22. Additionally, a public IP address associated with an instance is freed and can be reused by others if the instance is shutdown, even if it is later powered on again (it receives a new IP address at this point). In turn, it means that we only misclassify machines as offline with heavy ingress filtering that do not provide a service on the

[3]Some tests were up to twelve hours apart because of scheduling delay.

[4]We exclude networks of cloud providers that are used for services other than cloud virtual machine instances, for example, Load-Balancing-as-a-Service.

[5]We chose a two seconds timeout after we experimented with higher timeouts of five to ten seconds and did not notice any difference in results. A shorter timeouf of one second resulted in a high misclassification rate due to network and system load. The cut-off for no misclassifications was close to 1.4 seconds in our tests. Out of carefulness, we chose a two-second timeout.

Protocol (Common)	TCP	UDP	Port(s) ▲
FTP	✓	✓	21
SSH	✓	✓	22, 2222, 22022
Telnet	✓	✓	23
SMTP	✓	✓	25, 587
WHOIS	✓		43
DNS	✓	✓	53
HTTP	✓		80, 8000, 8080
Kerberos	✓	✓	88
POP3	✓	✓	110
IMAP	✓	✓	143
LDAP	✓	✓	389
HTTP (Secure)	✓		443, 8443
SMTP (Secure)	✓	✓	465
LDAP (Secure)	✓	✓	636
Telnet (Secure)	✓	✓	992
IMAP (Secure)	✓	✓	993
POP3 (Secure)	✓	✓	995
MS SQL	✓	✓	1433
CPanel	✓		2082
CPanel (Secure)	✓		2083
CPanel WHM	✓		2086
CPanel WHM (Secure)	✓		2087
MySQL	✓	✓	3306
2Wire RPC	✓	✓	3479
Virtuosso	✓		4643
Postgres	✓	✓	5432
CWMP	✓	✓	7547
Plesk	✓		8087
Webmin	✓		10000
ENSIM	✓		19638

Table 4.1: Ports and protocols used for IP address liveness checking

top 36 ports (see Table 4.1), and which have not been migrated to an internal network yet, which is becoming scarcer. Therefore, although our estimate remains an upper-bound, we are confident that it is a close estimate.

Over the course of our measurements, we classify 702,180 unique domains (0.539%) as pointing to available and freed IP addresses. Therefore, these domains, most likely, have been vulnerable to a (temporary) domain takeover attack at some point in time. In fact, while the majority of domains migrated delayed (80.31%), a non-negligible amount of domain-IP mappings are abandoned (17.24%)

and, fortunately, only a small number of domain-IP mappings are auxiliary (2.45%). Note that we only determine that the domain could be taken over, but its prior purpose remains unknown. Further investigation by future work is required to determine how many of the vulnerable domains have been actively used in the past and what the impact of an attack on them would be, for example, on a website that is protected through HTTPS and requires a TLS certificate, or a domain that is used to load remote code for a mobile application (see Section 4.3.1). Although the amount of vulnerable domains appears small relatively speaking, in absolute terms, the number of stale domains is quite large. Additionally, due to the nature of our dataset, we only observe domains that are actively being attempted to be accessed: the estimated number of cases that might be vulnerable to domain takeover attacks and could be abused for phishing or scams, but which were not being accessed during our observation period, might be significantly larger.

4.3.5 Proof of Concept Domain Takeover

Finally, we show the practicality of domain takeover attacks through a proof of concept certificate request to Let's Encrypt. Certainly, we face the largest ethical challenges with this experiment, as disrupting or having any impact on legitimate users raises ethical concerns. For example, it is impossible to guarantee that we do not interfere with any third party operation that might rely on the domain, or that we do not accidentally receive Personally Identifiable Information (PII) or other confidential data. Therefore, we perform a domain takeover attack for a domain under our control. After obtaining the certificate from Let's Encrypt and verifying that it has been published to certificate transparency logs, we revoke it, and publish the revocation to Let's Encrypt. The time until these actions appear in CT logs serves as an indication of the time that passes before the legitimate owner would be able to notice the attack by monitoring CT logs.

For our proof of concept experiment, we gained temporary control over the domain "cloudstrife.seclab.cs.ucsb.edu" by attempting to re-allocate the IP address to which the domain points to

(34.215.255.68). Note, that the IP address is located in the availability zone us-west-2, which has a high churn that makes takeover attackers more difficult. While this may seem contradictory, as a high churn means that an attacker can allocate more addresses per time-unit, a high churn also indicates a larger IP address pool. Ultimately, we were able to successfully re-allocate the IP address within 27 minutes and 55 seconds with a slow allocation cycle of two IP addresses per second. While anecdotal, it serves as an estimate of the time needed to launch an attack successfully under unfavorable conditions for an attacker (high churn, low allocation rate). We requested a TLS certificate from Let's Encrypt, it appeared in different certificate transparency logs between 34 minutes and 61 minutes later, and we revoked the certificate immediately after certificate transparency log entries had been propagated. Our certificate request was published at the Certificate Search (crt.sh) web-interface under id 250959196; it can be viewed at `https://crt.sh/?id=250959196`. The certificate that we obtained from Let's Encrypt and a message signed by the respective private key is contained in Appendix A.1.

Although the incorrect migration of domain-IP mappings is comparatively small on a relative scale, we believe that the absolute numbers speak volumes paired with the practicality of takeovers. Together, they justify looking closer at mitigating IP address use-after-free at its core, however, with a strong requirement to incur as little additional overhead on usability or performance as possible.

4.4 Mitigation

We address the issue of IP re-use attacks abusing stale DNS records, particular for IP addresses belonging to cloud networks, a topic that has received little attention so far. To be more specific, we investigate IP address use-after-free vulnerabilities, which can pose severe security threats, and which can be made even more dangerous through domain-validated TLS certificates (see Section 4.3). Current automated domain-validation-based certificate issuance systems are also threatened to be exploited through man-in-the-middle attacks discussed by Gavrichenkov et al. [157]. Existing defenses rely on certificate revocation, which is severely fragmented and cannot be relied on in practice [158, 159]. It became only recently more tractable, for example, through CRLite [160], but these solutions have not been adopted yet. One core problem is that revocation checks in browsers are not comprehensive: Chrome generally does not verify revocations, its CRLSet is limited to emergency revocations by design [161], and Mozilla's Firefox similarly limits revocation checks through OneCRL to CA intermediate certificates [162]. Certificate revocations in other software and libraries, which rely on the same certificate issuance processes and would also be required to adopt the new revocation checks, are rarely checked in practice [163]. Furthermore, revocations are reactive by nature, and they provide a window of opportunity to an attacker by design: the time until the revocation has propagated plus the time until the attacker's certificate has been revoked by the issuing CA on request of the legitimate party, the latter of which is generally a manual process as additional verification is required. We believe that the first line of defense should be with domain-validation-based CAs and it should be preventive. Therefore, we propose an additional layer of protection for domain-validation-based CAs, such as Let's Encrypt, that can efficiently and with negligible overhead prevent these attacks. Our mitigation technique builds on the ACME protocol version 2 [128] and it is complimentary to the certificate transparency framework [141].

4.4.1 General Concept and Threat Model

The underlying problem of IP address re-use attacks is that a domain-validated certificate can be requested as soon as an attacker controls the IP address to which a domain points to, and that requesting and receiving a trusted certificate is fully automatic and only a matter of seconds nowadays. An attacker might be able to obtain the IP address legitimately, because the domain record was left stale. To obtain a certificate, she might also be able to perform man-in-the-middle attacks between the authenticating CA and the target system. A similar issue occurs, if she can (temporarily) compromise the DNS (delegation or authoritative servers) for a domain. Then, she can simply change the IP address a record points to, as well as potential CAA or DANE TLSA records [164, 165]. Technically, attacks involving DNS-based attacks should be prevented by DNSSEC [134]. However, if key signing is performed online on the authoritative servers itself (against DNSSEC best practices) [166], and she compromises one of these servers, then she regains full control over the domain. Although, domain takeovers rarely tend to last for extended periods of time, TLS certificate for the domain can later be used by the attacker until the certificate's expiration date, possibly involving other man-in-the-middle attacks.

For all certificate requests that a CA receives, one of the following four cases applies:

1. No certificate has been requested for this domain in the past.
2. A certificate for the domain has been requested in the past, and the domain still points to the same IP address.
3. A certificate for the domain has been requested in the past, but the domain now points to a different IP address.
4. A certificate for the domain has been requested in the past, but it was verified in a more strict manner, possibly using extended validation (EV).

The first case is relatively frequent, and it is indistinguishable from the legitimate first use of domain-validated certificate issuance, which it is impossible to protect against without extended validation, which is itself often deemed too costly or impractical. We also acknowledge that an attacker who has compromised the system to which this domain points to will, in any case, be able to issue a new certificate for the domain, or steal the existing one.[6] Hence, a full system compromise is outside of

[6]Certificate theft can be protected through hardware security modules and may further become a commodity through methods like Intel SGX or ARM's TrustZone, which can be used to entrench certificate handling in a secured enclave.

the scope of our work. What our mitigation technique has to ensure is that a domain-validated certificate is only issued if the CA can verify that there has been no non-cooperative change of authority over either the system the domain points to or DNS zone for the domain.

Concerning our threat model, the attacker does not control a trusted CA, and she has average resources and skills, that is, she is not a state-level actor and cannot expend significant resources for a successful attack. Her overall objective is to obtain a domain-validated TLS certificate for a target domain that already uses a valid TLS certificate issued by a third party CA. However, she has no administrative access to the machine that the target domain currently points to, she cannot steal the current certificate or factors its keys in a reasonable amount of time, but, instead, she must request a new certificate. Taken into account the current operational model for domain-validating CAs, to achieve her goal, the attacker can: (a) obtain access to an IP address to which a stale A record for the domain points to, (b) perform a man-in-the-middle attack somewhere on the path between the issuing CA and the system to which the target domain points to, or, (c) illegitimately take over authority over the DNS zone for some amount of time.

4.4.2 Pre-Signature Certificate Consistency Checks

To ensure that an attacker within our threat model cannot request a new certificate, we must ensure that she cannot show that there has been a cooperative change for: (a) the IP address to which the domain points to, or (b) the DNS zone of the domain. One way to accomplish this task is by requiring each subsequent certificate request for a domain for which a certificate has been issued in the past by trusted CA, or which was covered by a similarly issued wild-card certificate, to be signed with a preexisting certificate.

Pre-Signed Domains

A challenge for a CA receiving a domain-validation certificate request is to determine whether a TLS certificate has been issued to this domain in the past, either by itself, or possibly by another trusted CA. Fortunately, two approaches to implement these requirements exist that are viable:

Federated Approach.

In case of the federated approach, each trusted CA is required to publish its issued certificates in multiple certificate transparency logs, which do not need to be run by the CA itself [141]. This approach has the strong advantage that it utilizes established technology, meaning that the required functionality is readily available and no additional service needs to be deployed and managed. Although certificate transparency logs are not yet required for every CA or every certificate, and not all CAs are publishing certificate logs, Google Chrome is already requiring CT logs to some certificates: for all certificates issued by Symantec, WoSign, and StartCom, as well as for all extended validation certificates (since January 2015). Furthermore, enforcing the requirement for all trusted CAs is expected within the next years [167]. Thus, expected development and policy changes would further empower this approach.

From an algorithmic point of view, a naïve existence check requires lookups for each trusted CA in an aggregated database. Fortunately, by leveraging CAA records via DNS combined with DNSSEC, one can limit lookups to a small set of CAs, for example, only one or two CAs. Specifically, it is more likely that one of the authorized CAs has issued a certificate for the domain in the past. Once a previously issued certificate has been found that is still valid, then the search can be terminated early, which reduces lookup time. Additionally, CAA records have become mandatory to be honored by CAs in September 2017 [168]. Therefore, due to the increasing adoption and availability of CT and CAA, we consider this approach the most practical and promising one.

Centralized Approach.

Alternatively, a centralized approach is possible. Here, a single authority, possibly IANA, would provide an oracle service. The service would return a boolean answer when queried, confirming whether any CA ever issued a certificate for a specified domain. Before issuing a new certificate, CAs would have to check if a certificate has been issued in the past. Furthermore, they must notify the authority of newly issued certificates. Unfortunately, the centralized approach bears potential trust issues and poses a single point of failure.

4.4.3 Domain Takeover Resistant Identifier Validation Challenge

Next, we develop a practical identifier validation challenge that is resistant to domain takeover attacks. Specifically, we target the ACME protocol, which is used by Let's Encrypt and others to automate the process of issuing certificates. To do so, we introduce an additional challenge to the ACMEv2 RFC [128]. No other changes to the RFC are necessary. In turn, it allows our validation challenge to be minimally invasive to the protocol and its subsequent implementations, yet, at the same time, it significantly improves security by mitigating the attacks that we present in this chapter. The core idea of our proposed challenge is to leverage existing certificates to form a chain of trust. Implementing a solution that uses existing certificates to sign responses to identification validation challenges triggers various issues with the handling of key material. For example, private keys should not be used outside of the context for which their respective certificate has been issued, which would happen if we naïvely sign a challenge response with a key, for which the respective certificate was issued for handling TLS server connections. Fortunately for us, retrieving the challenge response through over HTTPS eliminates the problem, and verifying the used certificate satisfies all requirements we put forth in the previous sections.

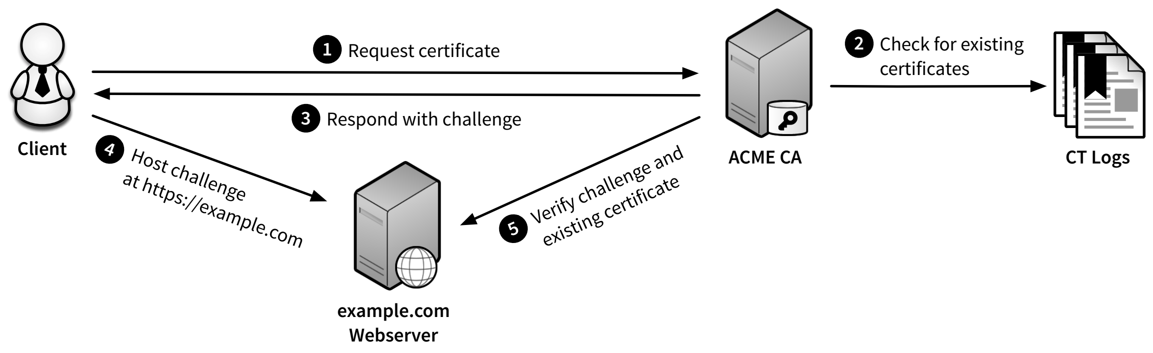

1 Request certificate
3 Respond with challenge
2 Check for existing certificates
Client
ACME CA
CT Logs
4 Host challenge at https://example.com
5 Verify challenge and existing certificate
example.com Webserver

Figure 4.3: Certificate request process that mitigates domain takeover attacks

Our challenge works as follows (see Figure 4.3):

❶ The client sends a new certificate request for her domain, for example, "example.com," to a domain-validating ACME CA.

❷ The CA checks whether a certificate for the domain "example.com" exists, that is, that one has been issued by a trusted CA in the past. The CA is free to include expired certificates in the check or ignore them according to an agreed-on policy (see Section 4.4.4).

❸ The CA issues a challenge to the client, which she needs to fulfill to validate ownership of the domain. If a prior certificate exists, the CA sends two challenges: first, our challenge, which is similar to the original HTTP challenge, and which includes a token to be hosted at a specified path at the domain of the requested certificate, and, second, a challenge that is considered more trustworthy than the HTTP challenge, such as a whois-based challenge or a DNS-based challenge. Following the ACMEv2 RFC, a client needs to satisfy only one of the two challenges. If she fails our challenge, which might happen in some cases (see Section 4.4.4), the more trustworthy challenge must be completed. For more details on how challenges are implemented, we refer to Section 8 "Identifier Validation Challenges"'of the ACME v2 RFC.

Alternatively, if no prior certificate exists, the CA is free to send any challenges as defined by the RFC.

❹ Once the client receives our challenge, she will host the nonce from it at the URL specified by the challenge to serve as the verification resource.

❺ The CA will attempt to access the verification resource, and, in turn, verify that the challenge has been completed by the client. Verification requires that the nonce has been placed at the resource, as well as that the HTTPS response is signed with the private key for a certificate of the domain that was previously issued by a trusted CA (see certificate existence check; ❷).

4.4.4 Failure Cases

There exist some possible failure scenarios of our challenge, which must be handled gracefully to preserve security of domain-validation. However, the simple failure of the process does not (yet) indicate an attack. Furthermore, as soon as a failure has been resolved, the above process can be used to regularly renew certificates automatically because the HTTPS challenge will not fail again for the same reason.

Lost Access to Old Certificate or Private Key

Among the most likely non-malicious scenarios for failure is the case of an operator who has lost access to her prior certificate or private key. Here, the HTTPS response cannot be signed and the challenge will fail. From a security standpoint, this case must be treated like a potential attack by the CA because it is impossible to automatically distinguish between a legitimate lost key, and an attacker not having access to the key in the first place. Instead, the operator should use a DNS-based challenge or whois-based challenge. Note that no additional certificate request is required, but the same certificate request will be used. In fact, instead of issuing the certificate, first, a prompt that additional verification is needed will be shown to the operator, and once she passes the additional challenge (sent along with the first challenge; ❸), only then the certificate will be issued.

Expired Certificate

Another common case in which the HTTPS challenge might fail are expired certificates. Operators may simply forget to renew their certificates in time, or a service may be shut down for a longer period, preventing certificate from being renewed. Whether expired certificates should be accepted, and if so, whether their expiration should be limited by a grace period, is a policy decision rather than a technical decision. Basically, two options exist:

1. Accept an expired certificate.

2. Treat it like an attack.

Relaxing the requirement and allowing expired certificates could increase the usability of our approach. However, relaxing requirements for corner cases introduces additional sources for potential errors, and thereby, security issues. Ultimately, we err on the side of caution and default to strong security and treating it as an attack, as also recommended by Fiebig et al. [169].

Legitimate Change of Authority

A third legitimate case that might fail is a legitimate change of domain ownership, possibly without the consent of the previous owner. Such cases include but are not limited to seizures because of copyright violations, or court orders, or a simple lapse in renewing the domain itself. Again, such a change in ownership cannot be recognized as legitimate by an automated system, simply because an attack has exactly the same properties. Therefore, similar to the lost private key access, the CA fails the HTTPS challenge and it requests a second challenge to be completed by the client, which any legitimate client can complete easily.

Possible Attacks

Following our earlier reasoning, attacks are cases in which the requesting client cannot prove a continuity in authority using previously issued valid certificates, which are considered rare, particular as you renew certificates in the validity period of your current period, and often automatically. Considering prior work (Chapter 6) and the attacks that we present in this chapter, a large portion of attacks are time critical. Therefore, the first aspect in the process of resolving a potential attack should be time. By increasing the time requirement, we increase the likelihood of the enabling attack to be detected. Nonetheless, potential for stale DNS attacks remains. Yet, we can approach this issue by designing an extended process for validating ownership of a domain and the correct delegation to an IP address. Unsurprisingly, CAs already commonly offer such extended validation processes. In addition, this service could also be offered by official institutions or NGOs with a sufficient trust level and the resources to do this. The certificates issued in this process would not even have to be valid for an extended time period. In fact, they can be used as simple seeds to re-initiate the continuous process of retrieving domain validated certificates.

4.4.5 Transitioning Techniques

One of the biggest problems when introducing new technique is the transitioning phase. However, for the adoption of our challenge, this is not an issue. The certificate ecosystem already makes extensive use of validity periods, generally certificates are set to expire within 1-3 years, and even as early as three months in case of Let's Encrypt. If our challenge would be adopted, we can also make use of extensive CT logs, which contain over hundreds of millions of domains already. For domains for which no entry exist in CT logs, we realize that our challenge is based upon *trust on first use* [170]. However, this does not leave domains for which certificates are already issued with less security than today, but it strictly increases security. Furthermore, CAs may add domains for which they previously signed certificates to certificate transparency logs voluntarily. Therefore, we believe that our system provides a robust and painless transition toward an increase of security for domain-validated certificates within the diverse certificate ecosystem.

4.4.6 Best Practices

Beyond directly addressing the root cause of the presented problems in the certificate issuing process, we suggest that cloud providers deploy mitigations as well. These mitigation techniques aim to prevent attackers from allocating specific addresses, for example, by rate-limiting IP address allocation and release operations, using disjoint sets of IP addresses for different tenants to reduce attack surface, and perhaps even by monitoring their networks for (non-scanning) inbound traffic to unallocated addresses to warn previous users of those addresses.[7] Finally, for cloud tenants, we strongly suggest keeping old addresses allocated when migrating IP addresses, at least until the TTL of the record has expired out, preferably until one can be reasonable sure that it is not cached anymore (preferably from a day to a week). Furthermore, we can only stress the importance of maintaining DNS zones properly and to remove obsolete records as quickly as possible to not fall victim to domain takeover attacks.

4.5 Conclusion

We have shown that it is practical, time-efficient, and cost-efficient for an attacker to (temporarily) takeover domains by exploiting so-called IP address use-after-free vulnerabilities on, currently, the two largest Infrastructure-as-a-Service clouds (Amazon AWS and Microsoft Azure).

In our study, we discovered that attacks are practical on public clouds because of their instances' ephemeral nature and the "throw-away culture" of development operations concerning immutable

[7]The noise-to-signal ratio might impractical for monitoring because of Internet-wide scanning efforts, and filtering scanning traffic from other traffic might be too costly for a supplemental warning service.

instances and service migration. In turn, it is not necessary to takeover the IP address to which a domain points to, but IP address migration occurs regularly and sometimes is outside of the control of the cloud user (e.g., reboot or shutdown of the hypervisor because of an update), thus freeing the previously assigned IP address and making it available for re-use by others. Here, a slightly incorrect DNS domain record migration strategy can immediately render domains vulnerable to IP address use-after-free attacks. In fact, the problem is even further amplified for so-called spot instances, which are significantly cheaper instances, but which can be terminated at any point and without notice to the cloud user, and for which he cannot protect himself from temporary domain takeovers.

We have examined the reasons of why and how IP address re-use domain takeover attacks can occur in practice, and we classify them according to what their potential impact in practice is. Particularly, we investigated their impact on domain-validated TLS certificate issuance, such as through automatic certificate management environments (ACME), for example, Let's Encrypt. Based on our findings, we then developed best practice recommendations for cloud operators as well as domain owners and cloud users, which can reduce vulnerability to the aforementioned attacks.

Finally, we introduced a new mitigation techniques that addresses the issue of domain takeover attacks for trust-based domain-validation services, focusing on the real-world case of automatic certificate issuance. Our mitigation technique protects against IP address use-after-free attacks with negligible operational overhead and only requires manual intervention in disaster-recovery scenarios, thus, rendering it practical for real-world deployment even under strict performance and usability requirements of services like Let's Encrypt.

Chapter 5

Enumerating IPv6 Hosts

Security research has made extensive use of exhaustive Internet-wide scans over the recent years, as they can provide significant insights into the overall state of security of the Internet, and ZMap made scanning the entire IPv4 address space practical. However, the IPv4 address space is exhausted, and a switch to IPv6, the only accepted long-term solution, is inevitable. In turn, to better understand the security of devices connected to the Internet, including in particular Internet of Things devices, it is imperative to include IPv6 addresses in security evaluations and scans. Unfortunately, it is practically infeasible to iterate through the entire IPv6 address space, as it is 2^{96} *times* larger than the IPv4 address space. Therefore, enumeration of active hosts prior to scanning is necessary. Without it, we will be unable to investigate the overall security of Internet-connected devices in the future.

In this chapter, we introduce a novel technique to enumerate an active part of the IPv6 address space by walking DNSSEC-signed IPv6 reverse zones. Subsequently, by scanning the enumerated addresses, we uncover significant security problems: the exposure of sensitive data, and incorrectly controlled access to hosts, such as access to routing infrastructure via administrative interfaces, all of which were accessible via IPv6. Furthermore, from our analysis of the differences between accessing dual-stack hosts via IPv6 and IPv4, we hypothesize that the root cause is that machines automatically and by default take on globally routable IPv6 addresses. This is a practice that the affected system administrators appear unaware of, as the respective services are almost always properly protected from unauthorized access via IPv4.

Our findings indicate *(i)* that enumerating active IPv6 hosts is practical without a preferential network position contrary to common belief, *(ii)* that the security of active IPv6 hosts is currently still lagging behind the security state of IPv4 hosts, and *(iii)* that unintended IPv6 connectivity is a major security issue for unaware system administrators.

5.1 Motivation and Contributions

There has been a multitude of Internet-wide security challenges of varying severity over the recent years. Heartbleed [171] and SSL related vulnerabilities [172, 173], common misconfigurations of database systems [174], and other issues like protocol amplifiers [175, 176] have been investigated closely. Studying these issues methodologically has only been possible because exhaustive security scans of the Internet Protocol version 4 (IPv4) address space became practical through ZMap in late 2013 [177]. Since then, Internet-wide IPv4 security scans have become an integral part of modern security research.

The total number of IPv4 addresses is, however, limited. For many of those addresses, their use is further restricted through special use arrangements, and because of large allocations to institutions that were early adopters of the Internet. In fact, all addresses managed by the Internet Assigned Numbers Authority (IANA) have been allocated as of September 24, 2015 when the American Registry for Internet Numbers (ARIN) allocated its last IPv4 address [178].

The accepted long-term solution to the IPv4 address exhaustion problem is considered to be the Internet Protocol version 6 (IPv6) [179, 180]. Contrary to the 32-bit wide addresses of IPv4, IPv6 uses 128-bit wide addresses (7.9×10^{28} as many as IPv4) and its adoption would eliminate the need for further address resources for the foreseeable future.

Indeed, IPv6 has gained significant traction in recent years: In August 2016, Google reported that almost 13% of their users accessed their services via IPv6. This number increased by an order of magnitude in just three years from 1.3% as of July 2013 [181]. Similarly, the Internet Society reports that "global IPv6 traffic has grown more than 500% since June 6, 2012." Many other network operators have deployed IPv6 to significant parts of their network since then [182]. In fact, for some networks, up to 97% of all devices use IPv6 (see Table 5.1).

Unfortunately, the vast address space of IPv6 threatens to take the important tool of Internet-wide scans away from the security community. Theoretically, for IPv6, up to 2^{128} addresses (approximately 3.4×10^{38}) can be allocated. While scanning all reachable devices is considered to be a solved problem for the IPv4 address space [177], it is practically infeasible to scan the entire IPv6 address space, because it is larger than the IPv4 address space by 2^{96} (28 orders of magnitude). In fact, a sweep over the entire IPv6 address space would take 7.532×10^{23} years with state-of-art tools for Internet-wide scanning.

Due to the Internet's continuing growth and its increasing dependence on IPv6 globally, it is critical to include IPv6-connected devices in future Internet-wide security evaluations, in addition to IPv4. This need is further amplified by the fact that IPv6 traffic is commonly enabled (by default). Often no standard security mechanisms, such as firewalls, have been put in place for IPv6, even though they are already in place for IPv4. In turn, it exposes the respective hosts to attacks from miscreants via IPv6 [183, 37].

At the same time, it remains difficult to perform Internet-wide IPv6 security scans, which leaves a dangerous blind spot. To address this issue, authors have started to suggest various techniques to perform Internet-wide IPv6 security scans, which leverage IPv6 seed sets to scan IPv6 hosts. The most recent of these, 6gen, has been presented by Murdock et al. [184]. However, most existing approaches to collect active IPv6 addresses as seed sets require network vantage points or leverage older, possibly stale, public datasets. For example, some techniques require access to content delivery networks or traffic brokers to observe IPv6 traffic and collect addresses [185, 186]. Others extract IPv6 addresses from historical forward DNS records, in the hope that they are still active [37]. Unfortunately, some techniques to collect these records, such as ANY queries, have since been deprecated by the operators of major nameservers to protect from denial of service attacks [187], which renders them impractical for IPv6 address collection. Fiebig et al. [9] recently introduced a different methodology to enumerate IPv6 hosts, namely by exploiting the NXDOMAIN semantics in the DNS ecosystem. However, their technique can be mitigated comparatively easily, as they demonstrated on an industry conference in 2016 [188]. Therefore, it is necessary to identify new seed-set collection techniques that allow researchers, who might not have access to network vantage points, to include IPv6-connected devices at scale in Internet-wide security evaluations.

To retain the capabilities of security researchers to conduct Internet-wide scans, we introduce a novel IPv6 address enumeration technique that leverages DNSSEC-signed IPv6 reverse zones. We show that our approach enumerates classes of active IPv6 addresses that existing techniques miss, and that prior work has not evaluated. Furthermore, our technique does not depend on implementation-specific behavior and it is resilient against the mitigation techniques that have been put in place to protect against the enumeration techniques of prior work. Instead, to prevent our enumeration technique, significant changes to the DNSSEC standard are required.

In our evaluation, we discovered that IPv6-connected hosts expose a variety of critical security issues: exposed file sharing, access to interior and exterior routing protocols, remote access to switches and routers, remote monitoring, hosts that can be exploited to launch reflected and amplified denial of service attacks, and, alarmingly, remote system management ports vulnerable to attacks that allow full machine takeover (e.g., IPMI, which provides practically physical access through remote keyboard and video).

Category	Network Operator	Percentage
Wireless Carrier	Digicel Trinidad & Tobago	97.04%
	Verizon Wireless	77.65%
	T-Mobile USA	71.09%
University	University of Twente	79.17%
	Virginia Tech	70.06%
Organization	United States Space and Naval Warfare Systems Command (SPAWAR)	74.52%
Broadband Provider	Google Fiber	64.96%
	xs4all (Netherlands)	61.75%

Table 5.1: IPv6 penetration of real-world networks [189]

We make the following contributions:

- We introduce a practical enumeration technique that effectively exploits DNSSEC zone walking to identify active IPv6 hosts by utilizing unique features and the well-structured format of the IPv6 reverse DNS tree. We focus on reverse zones that have deployed NSEC3 to thwart existing zone-walking attacks. Specifically, we exploit intricacies of how the IPv6 reverse zone is organized to make enumerating active IPv6 addresses in the face of NSEC3 practical.

- Our methodology is resilient against mitigations, including techniques effective against earlier enumeration approaches, and, to mitigate it, modifications to the DNSSEC standard are required. In fact, we enumerate hosts that have been hidden from established methodology using existing mitigations already.

- Using our methodology, we identify several vulnerabilities and misconfigurations of hosts reachable via IPv6 that were hidden from scans using methodology of prior work. Our results indicate that the exposed IPv6 addresses can cause additional and significant security risks, and network operators are required to take precautions when adding IPv6 addresses into the DNSSEC-signed reverse zones, as it inevitably leaks information about the presence of those hosts to potential attackers.

In the remainder of this chapter, we provide the necessary background information (Section 5.2), detail our enumeration technique (Section 5.3), discuss ethical considerations for active measurements (Section 5.4), evaluate our technique (Section 5.5), consider potential mitigations (Section 5.6), and, ultimately, conclude (Section 5.7).

5.2 Background

Some background information on the Domain Name System (DNS), DNSSEC, denial of existence records, and the way the IPv6 reverse zone is organized is required for our enumeration technique.

5.2.1 Domain Name System and DNSSEC

DNS is a core protocol of the current Internet architecture. It allows using easily identifiable hierarchically organized names instead of IP addresses to access services online. While the basic idea of the DNS is straightforward [132], denials of existence (NXDOMAIN) require some attention, as our approach builds upon their equivalent in the scope of DNSSEC.

In a simplified schema (see Figure 5.1), a client talks to a nameserver to inquire about whether a specific name for a specific resource record (RR) type exists within a zone. If the record does exist, then the nameserver responds with the respective answer (e.g., in case of an A record, with the IPv4 address mapping for a name). If the record does not exist, the nameserver generates a NXDOMAIN response (NX signifying “non-existing”).

Unfortunately, however, the DNS protocol does not provide authenticity and it is susceptible to a variety of attacks, including man-in-the-middle attacks, like filtering, redirection, and response spoofing [190, 191]. An intermediate nameserver could (maliciously) hijack NXDOMAIN responses and replace them with a record that points to an advertisement website [192, 193]. While

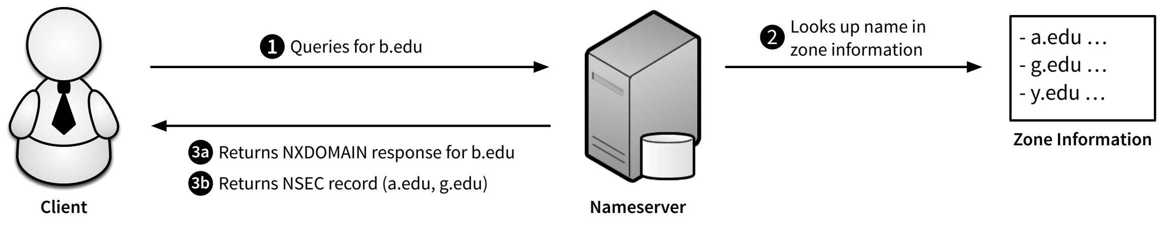

1 Queries for b.edu
2 Looks up name in zone information
- a.edu …
- g.edu …
- y.edu …
Zone Information
3a Returns NXDOMAIN response for b.edu
3b Returns NSEC record (a.edu, g.edu)
Client
Nameserver

Figure 5.1: Example DNS interaction between a client querying a nameserver without and with DNSSEC. The client queries the nameserver for a record of the domain “b.edu” (1). The nameserver looks up the resource record (RR) in the zone information (2). Here, the queried resource does not exist in the zone file. If DNSSEC is not present, then the nameserver responds with a single NXDOMAIN response that is generated online (3a). If DNSSEC is present, then the nameserver responds with an authenticated response. Since DNSSEC discourages online signing, a pre-signed entry must exist. However, pre-signed denials of existence for any possible query are impractical from a space and computational perspective. Therefore, DNSSEC returns pre-signed denials of existence for an entire name range: the previous existing entry with an associated record is “a.edu,” the next existing entry with a record is “g.edu” (3b), effectively leaking the existence of those names within the zone.

the intermediate nameserver is intentionally violating the standard, it is technically able to return bogus responses because they are not authenticated.

DNSSEC aims to solve these authentication problems via cryptographic signatures for records contained as part of a zone. Authenticating existing records is an extension of DNS through a signature record type (RRSIG) for each original record, which is signed with a zone-signing key (ZSK). The public key portion of the ZSK is hosted in the zone, while the parent zone provides a hash of the ZSK in a DS RR. In turn, it solves the problem of distributing public keys in a trustworthy manner through DNS’ hierarchical nature and its existing chain of trust from the root zone to the queried zone. Intuitively, signing NXDOMAIN RRs would be possible if the zone-signing key is available at the nameserver, so that the generated records can be signed online. However, DNSSEC discourages the use of online signing to prevent denial of service attacks against the nameserver and chosen-plaintext attacks against the zone-signing key [133]. Instead, it strongly encourages to serve zone information that was signed offline. Consequently, authenticating denials of existence naïvely is impractical: all non-existing names would have to be signed and it would require operators to create zones of practically unbounded size. As a solution, a single NSEC RR is used to deny the existence of a range of records: it describes the previous existing name and the next existing name. For example, an NSEC record might point to "a.edu" as the previous existing name and "g.edu" as the next existing record. Then, any query for a name that is lexically between "a.edu" and "g.edu," for example, "c.edu" or "foo.edu," would result in the same authenticated NSEC response. This is an efficient authenticated denial of existence, satisfying the requirements of DNSSEC.

5.2.2 IPv6 and Reverse IPv6 Zones

Contrary to IPv4's quad-dotted decimal representation, IPv6 addresses are instead represented as 32 hexadecimal digits, which are divided into eight groups of four digits to ease readability, for example, `2001:0db8:0000:0bad:f00d:feed:cafe:0001`. For convenience, addresses can be abbreviated by removing leading zeroes and replacing the largest consecutive group of zeroes with a double colon, for example, the above address can be abbreviated to the shorter `2001:db8::bad:f00d:feed:cafe:1`.

Conceptually, reverse zones are like any other standard DNS zone, but they have a specific meaning: They are used to map an address or resource, such as an IPv4 or IPv6 address, to a name instead of the other way around. For IPv6, the designated reverse zone is ip6.arpa and it is hierarchically organized at nibble (a nibble is a single hexadecimal digit) boundaries in reverse order. Listing 5.1 depicts an example reverse zone for `2001:db8::/32` with two entries, one for `2001:db8::bad:f00d:`

`feed:cafe:2` pointing to "h.a.edu" and one for `2001:db8::bad:f00d:feed:cafe:9` pointing to "s.a.edu."

In practice, reverse address zones are used for a variety of reasons. Initially devised for troubleshooting, reverse lookups for forward-confirmed reverse DNS names are nowadays its main use case and considered best operational practice [194]. A forward-confirmed reverse DNS lookup corresponds to looking up the domain name with an address and then looking up the address for that domain name, if they are the same, then the lookup is considered confirmed. Today, most mail transfer agents (MTA) rely on confirming reverse DNS lookups to reduce spam and might reject or bounce incoming mail if the lookup is not forward-confirmed [195]. Consequently, network operators are essentially forced to deploy reverse zones to not degrade the quality of service for the hosts in their network. In practice, reverse zones are regularly populated automatically via DHCP and IPv6 node information queries and the reverse zone information accurately represents an active part of the network [196, 197, 12].

Due to DNS' inherent hierarchical design and the IPv6 address space being split into a significant number of sub-networks, it is not possible to simply download the entire reverse zone for IPv6 to enumerate hosts. In fact, the sub-networks are delegated to thousands of different nameservers worldwide, which do not allow to download the respective reverse zones directly. Hence, it motivates the need for an effective IPv6 address enumeration technique.

Fortunately, the IPv6 reverse zone (ip6.arpa) supports DNSSEC since April 2010, which enables our enumeration approach if the respective delegate reverse zones are also DNSSEC-signed. Currently, as of January 2018, already 51 out of 59 delegate IPv6 reverse zones (i.e., zones below ip6.arpa) are signed via DNSSEC [198], and, thus, this allows our approach to enumerate IPv6 hosts within those zones, that is, within those networks. Interestingly, the (still) unsigned reverse zones include the 6-to-4 zone (`2002::/16`), which is an IPv6 transition mechanism and which can be enumerated through traditional IPv4 enumeration techniques.

```
$TTL 1h
@   IN SOA ns1.a.edu. admin.a.edu. (
    2018010101    ; serial
    1h 15m 1w 1h) ; refresh retry copy cache

@   IN  NS  ns1.a.edu.

; IPv6 PTR Entries
2.0.0.0.e.f.a.c.d.e.e.f.d.0.0.f.d.a.b.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.  IN  \
PTR  h.a.edu.
9.0.0.0.e.f.a.c.d.e.e.f.d.0.0.f.d.a.b.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa.  IN  \
PTR  s.a.edu.
```

Listing 5.1: Example IPv6 reverse zone for 2001:db8::/32

5.3 Approach

Following, we describe our approach, which enumerates active IPv6 addresses by walking an IPv6 network's reverse zone. The network can be the entire IPv6 address space or any sub-network that might be of particular interest, for example as part of a security evaluation. Consequently, our approach can be targeted and can be faster than state-of-the-art techniques.

Our enumeration technique requires that the reverse zone for the network is signed via DNSSEC because it relies on NSEC or NSEC3 responses for non-existing addresses. Nevertheless, it is already practical because over 86% of the top-level delegations in the IPv6 reverse zone are already DNSSEC signed and it is expected that all zones will support DNSSEC soon [199]. In fact, NIST recommends deploying DNSSEC since September 2013 [200] and adoption has been ever increasing since then [199]. If the records are not signed yet, for example because a large network is partitioned into smaller networks and only some of the zones employ DNSSEC then we can still enumerate the hosts within networks for which the reverse zones are signed (regardless of whether intermediate zones are signed).

A fundamental difference of our approach to existing techniques that determine an active part of the IPv6 address space through network vantage points or datasets is that our approach can enumerate hosts that do not actively initiate connections, nor does it require that IPv6 addresses appear in a forward zone. At the same time, conventional "brute-force" enumeration attacks known from IPv4 [177] do not scale to the vast IPv6 address space, while our approach can enumerate sparsely populated IPv6 networks without problems.

5.3.1 Reverse Zones with NSEC

It is an understood problem that NSEC denials of existence allow zone-walking attacks on signed zones because they leak the previous and next existing name of that zone. In case of the IPv6 reverse zone, those leaks correspond to the previous and next IPv6 name pointer (PTR) for an address in that reverse zone, or a nameserver (NS), if a subdomain (sub-network) is delegated to another nameserver [201]. We modify the existing NSEC-based approach and exploit the organization of the IPv6 reverse zone to enumerate addresses more efficiently.

Starting from a target IPv6 reverse zone, for example, the root zone for the entire IPv6 address space, the steps to enumerate the reverse zone for NSEC-based denials of existence records are:

1. *Bootstrapping*: We query for a random string below the target zone, like foobar.ip6.arpa, to determine a starting point for address enumeration (seed). Based on the organization of the IPv6 reverse zone (as specified by RFC 5855 [202]), it is guaranteed that a random string that is not a single hexadecimal digit will result in a NSEC response. In turn, it removes the requirement to identify a non-existing address in the address space prior enumeration.

2. *Zone Walking*: Starting from the seed, we follow the chain by iteratively querying the next addresses incremented by one, that is, the next address that might not exist and could yield a denial of existence.

   If we do not receive a NSEC response, then we discovered an active address, and we keep incrementing the address until we receive a NSEC response. Once we receive a NSEC response, based on the organization of the IPv6 reverse zone, we can immediately identify if the next entry of a NSEC record is an address or a sub-network: if it is not a full-length IPv6 address (32 nibbles), then this sub-part of the reverse zone is delegated, possibly to another DNS server.

If we encounter a zone delegation, we optionally identify via a random seed whether it is signed at all, and if so, if we can immediately descend into it (NSEC) or if it requires further processing (NSEC3). If we can descend into it, we optionally add it to a sub-zone queue (i.e., we perform breadth-first search).

We terminate the zone-walking step if the next address in the returned NSEC record points to the seed (we have closed the chain and formed a circle).

3. *Sub-zone Enumeration (optional)*: For each sub-zone that we added to our queue, we may descend into it and recursively apply the same enumeration strategy.

Intuitively, the runtime of our approach to enumerate IPv6 addresses for NSEC-based reverse zones is linear and requires $O(n + m)$ DNS queries to nameservers for the reverse zone where n is the number of addresses within the networks and m is the number of zone delegations.

5.3.2 Reverse Zones with NSEC3

In an attempt to mitigate the side effect of zone-walking attacks on DNSSEC-signed zones, Laurie et al. proposed NSEC3 [201]. Instead of listing the previous and next existing name in clear, NSEC3 uses a cryptographic hash for the names in the zone, sorts the hash values in alphabetical order, and then uses each pair of consecutive hash values in the zone to indicate the denials of existence through a NSEC3 record.

If the zone is using NSEC3, then the nameserver responds to a query for a non-existing name n as follows: it computes its hash value $h(n)$ where h is the cryptographic hash function as specified for the zone, and it then returns the NSEC3 record with the pre-computed hashes of the existing names n_1 and n_2, such that $h(n_1) < h(n) < h(n_2)$. Note that $n_1 < n < n_2$ does generally not hold because h is not order-preserving. In fact, since the names are ordered by their hash value, and since h is not order-preserving, only the cryptographic hashes of two existing names are exposed, which are considered computationally difficult to reverse.

Given a NSEC3 response, the client can verify herself that the name does indeed not exist in the zone. She verifies that the NSEC3 response is authentic and then verifies that the queried name, when hashed, falls into the range specified by the NSEC3 record. To hash the queried name, she uses the parameters specified in the authenticated NSEC3 record, that is, hash algorithm (only SHA1 is currently supported), salt, and the number of iterations, which are valid for the entire zone.

Nevertheless, NSEC3 records still leak two existing records from the zone, even though their names are cryptographically hashed. Therefore, they are technically still vulnerable to zone enumeration through brute-force and dictionary attacks [203, 204]. In fact, the attacks identified by prior work inspired our research. However, existing approaches for forward zones are ineffective for the IPv6 reverse zone because of the reverse zone's organization: *(i)* existing dictionary attacks, such as nsec3walker, are inefficient due to the small alphabet (0-f, one character maximum) and the large height of the zone's hierarchical tree; and *(ii)* uninformed brute-force attacks are computationally expensive and considerable computational resources are required to successfully launch them, particularly considering the size of the IPv6 address space. Following, for our case, we show the contrary: enumerating IPv6 addresses for NSEC3-protected reverse zones is practical and effectively computationally less complex than uninformed brute-force attacks.

Different from NSEC-based address enumeration, in case of NSEC3, a two-phased approach is required. First, we need to collect the NSEC3 chain for a zone online by actively querying for names. Subsequently, we can unblind the IPv6 addresses offline. Note that the first phase does not necessarily have to be completed before we can launch the second phase. We can launch the second phase as early as the first NSEC3 record is being observed, which can reduce the time required to enumerate the target network's addresses significantly. Furthermore, even though a network operator could change hash parameters during the collection phase, such as the salt or the iteration count, previously collected NSEC3 records can still be unblinded and used to enumerate hosts within the zone. Following, we discuss how our approach can efficiently unblind NSEC3-protected IPv6 addresses in the reverse zone by exploiting intricate details of the specification and implementation of the IPv6 reverse zone.

5.3.3 Online Collection

The design of NSEC3 makes it computationally impractical to follow its chain to find the next hash. Instead, the core idea is to randomly query for names that do not exist until the full NSEC3 chain has been recovered. Similar to the NSEC case, a complete chain of NSEC3 records forms a closed circle and, thus, can be verified easily. During the sampling process, any not-yet-discovered NSEC3 records leave missing "gaps" on the circle. Eventually, the sampling process will fill all gaps (see Figure 5.2). The problem of discovering names whose hashes are inside one of the remaining gaps is similarly embarrassingly parallel as the offline unblinding step and can easily be sped up massively through graphical processing units.

For NSEC3-based reverse zones, online collection works as follows:

1. *Bootstrapping*: We query for a random string below the target zone, like foobar.ip6.arpa, to determine a starting point for online collection. As in the case for NSEC, it is guaranteed that a random string that is not a single hexadecimal digit will result in a NSEC3 response and it removes the requirement to identify a non-existing address in the address space prior enumeration.

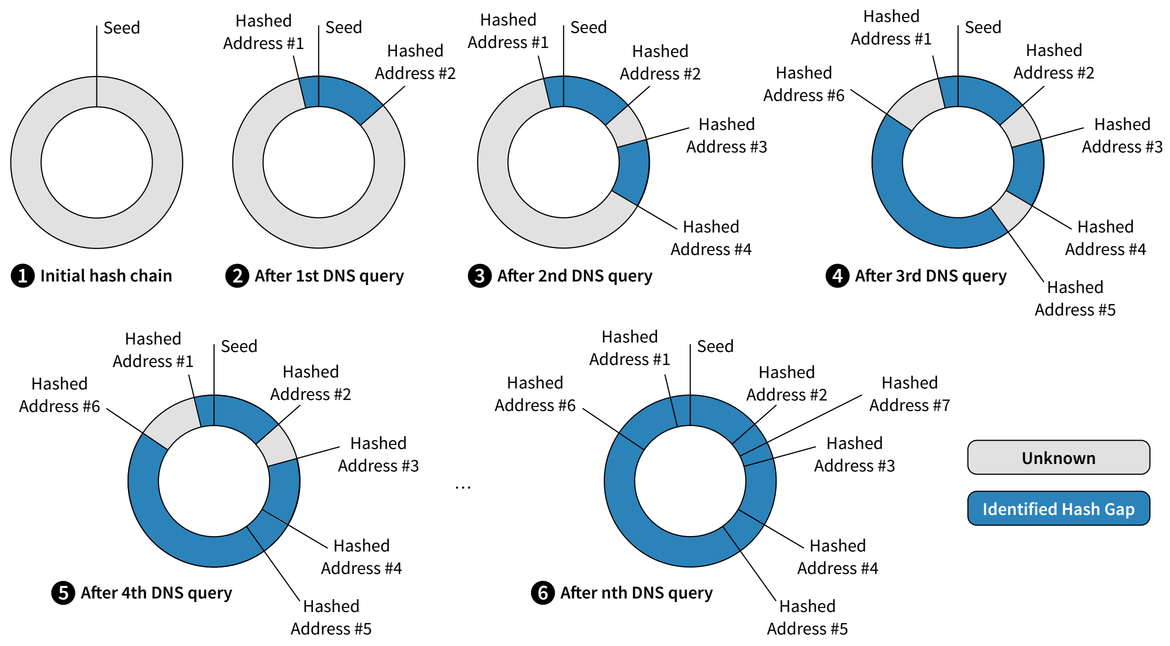


Figure 5.2: Online collection and NSEC3 hash gaps. During the online collection phase for NSEC3-protected zones, we first bootstrap by choosing a random seed that is guaranteed to result in a NSEC3 response for the zone, which exposes two hashed addresses. Following, we walk the zone randomly and iteratively fill hash gaps to discover more addresses until we have successfully identified all hash gaps.

In addition, we are also interested in the current hash algorithm, salt, and iteration count to fill hash gaps locally as to not query the nameserver unnecessarily or cause suspicion or incur unnecessary load.

2. *Zone-Walking*: We calculate the hash value for a random name under the zone based on salt and iteration count. If the hash value is covered already by a range uncovered from the previously collected NSEC3 records, then we repeatedly select random names until a hash falls into a gap and is guaranteed to reveal more information about the NSEC3 chain (see Figure 5.2).

Intuitively, with the number of hash gaps decreasing, the probability to hit one of the remaining ones decreases too, and the time requirement increases. The average number of required hash calculations is $O(r \log r)$ with r being the number of records in the zone (addresses plus delegated sub-zones).

Already during the collection phase we can determine whether a hash is a full IPv6 address or a zone delegation: a NSEC3 record leaks whether the next hashed value is a PTR record (full IPv6 address) or a NS record (sub-zone delegation) (see Listing 5.2). In fact, this detail allows us to separate addresses and networks into different buckets and unblind them separately later, which reduces computational cost significantly.

We retain all NSEC3 records for offline unblinding.

We repeat the zone-walking step until no more hash gaps exist or in case an exit condition is true, in which case parts of the address space remain unexplored. If we have filled all hash gaps within the NSEC3 circle, we have successfully collected all hashed IPv6 addresses and sub-zone prefixes.

The runtime of the online collection phase is $O(n+m)$ DNS queries to the nameservers where n is the number of addresses within the target network and m is the number of sub-zone delegations.

To probabilistically enumerate addresses within a zone, one may specify an exit condition that terminates the zone walking step. A trivial condition might be a timeout during which a new gap must be filled. However, a more intelligent solution is to fill in all gaps until at most x gaps of at most size y exist. At that point, at most $x \times y$ hashes of the entire zone will not be collected through our approach (effectively, missing at most $x \times y$ addresses or sub-zones). Here, x and y can be chosen to specific probabilistic requirements, such as “at least 95% of the zone must be enumerated.” Additionally, if hashes within those ranges are later discovered during unblinding, the gaps can be filled.

```
; Reverse IPv6 NSEC Entries
2.0.0.0.e.f.a.c.d.e.e.f.d.0.0.f.d.a.b.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. IN \
NSEC 9.0.0.0.e.f.a.c.d.e.e.f.d.0.0.f.d.a.b.0.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. \
PTR RRSIG

; Reverse IPv6 NSEC3 Entries
1PDJ9FP13S70NCFCJCV35B8LLVT68U5Q.8.b.d.0.1.0.0.2.ip6.arpa. IN NSEC3 1 0 10 86\
B3E6B74F0A2C23 G5AL6GMJ6ARLJ9M5F56LL48JPHJ1SGQK PTR RRSIG
```

Listing 5.2: Example NSEC and NSEC3 RRs for the reverse IPv6 zone for `2001:db8::/32`. A client querying for a name that is lexically between `2001:db8::bad:f00d:feed:cafe:2` and `2001:db8::bad:f00d:feed:cafe:9` will receive the NSEC (top) record from the nameserver. For NSEC3, if no record exists in the zone whose hash is lexically between `1PDJ9FP13S70NCFCJCV35B8LLVT68U5Q` and `G5AL6GMJ6ARLJ9M5F56LL48JPHJ1SGQK` (base32-encoded SHA1), then the NSEC3 record will be returned.

5.3.4 Offline Unblinding

Following online collection, the next step to enumerate IPv6 addresses is to unblind the collected hashes offline. Since DNSSEC leverages cryptographically secure hashes, the naïve choice falls to brute-force attacks. Brute-force attacks, however, are impractical because of the large search space for SHA1, which is the only supported hash of DNSSEC, at 2^{160} possible values.

Generally, domain names can be composed of letters, digits, and hyphens [205]. The IPv6 reverse zone, however, follows a well-defined structure: each subdomain is strictly a hexadecimal digit (see Section 5.2.2). Practically, by leveraging the organization of the IPv6 reverse zone, we can unblind hashed IPv6 addresses (which we identified as full addresses during online collection) significantly faster through directed search. We exploit the fact that addresses are almost never randomly assigned from a network's range, but instead follow observable patterns. First, addresses are often assigned incrementally through static assignment or via DHCPv6, possibly with gaps at earlier nibbles, such as `2001:db8::1/64`, `2001:db8::2/64`, or, with a gap, `2001:db8::1:1/64`. Second, addresses are also more likely to be assigned through stateless address autoconfiguration (SLAAC) than being randomly picked. With SLAAC, a host commonly assigns itself an IPv6 address based on its MAC address, in which case 12 nibbles (out of 32 nibbles) of the IPv6 address are based on the host's MAC address, which is vendor-based, and additional four nibbles are constant across all IPs assigned through SLAAC. For example, a host with MAC address `00:11:22:33:44:55` on the network `2001:db8::/32` would assign itself the IPv6 address `2001:db8::211:22ff:fe33:4455`. As of January 2018, only 24,434 vendor prefixes are officially in use [206], and combined with the constant nibbles, it reduces the search space by a factor of 2^{25}. Inherently, a MAC-based address assignment strategy allows Internet-wide equipment and user tracking, because the MAC is considered universally unique and remains constant across networks. To prevent such tracking, privacy extensions were added to SLAAC, for which temporary addresses may be used instead. These privacy extensions make the enumeration attack more difficult initially due to the addresses' ephemeral nature, however, their effectiveness degrades over time since addresses are generally not reused. Furthermore, their use is commonly limited to end users and they are not used by servers or network equipment.

Overall, we can reduce the search space from 2^{128} to as little as 2^{39} for full IPv6 addresses (although the SHA1 search space is 2^{160}, it is reduced to 2^{128} because IPv6 addresses are only 128-bit wide) depending on network prefix and address assignment strategies used. By guiding the address search intelligently, we can further speed up the unblinding process. Specifically, we can exploit that a hash gap (pair of NSEC3 records) leaks the type of the preceding and following resource record. The type of the resource record indicates the length of the unhashed value (PTR for full addresses, NS and SOA for network prefixes), which, in turn, significantly reduces the complexity of unblinding the hashed value. Practically, we can reduce the search space down to as little as 2^{39} for full addresses and 2^{33} for networks, which renders enumeration practical. Notably, we successfully unblinded various networks of different sizes (/32, /48, and /64) in mere hours, including for reverse zones with high hash iteration count (see Section 5.5).

Unblinding zone delegations is practical for similar reasons: First, we accurately identify them as delegated zones during online collection (since the NSEC3 record leaks whether the next hash is a PTR or NS record). Second, we exploit that sub-networks, a common cause for sub-zones being delegated, are commonly assigned and used incrementally rather than randomly from the vast address space. Third, we exploit that networks are allocated at specific nibble boundaries, effectively limiting the search space to $\sum_{0 \leq i \leq 8} 2^{4i}$ ($\leq 2^{33}$).

For example, for the hashes `g5al6gmj6arlj9m5f56ll48jphj1sgqk` and `1pdj9fp13s70ncfcjcv35b8llvt68u5q` (see Listing 5.2), we only need to attempt to unblind full addresses as they are PTR records. Combined with the salt `86b3e6b74f0a2c23`, we can then unblind the hashes to `2.0.0.0.e.f.a.c.d.e.e.f.d.0.0.f.d.a.b.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa` and `9.0.0.0.e.f.a.c.d.e.e.f.d.0.0.f.d.a.b.0.0.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa`, that is, they represent the IPv6 addresses `2001:db8:bad:f00d:feed:cafe:2` and `2001:db8:bad:f00d:feed:cafe:9` respectively.

In summary, our approach can quickly enumerate IPv6 hosts and networks, even for sparsely populated IPv6 networks, by exploiting the well-defined organization of IPv6 addresses and networks, and by leveraging the structure of the IPv6 reverse zone and the information (record type) leakage of DNSSEC-based denial of existence records (NSEC3).

5.4 Ethical Considerations

In our evaluation, we perform active measurements on the enumerated addresses to establish if they are actually active. We also establish a limited set of additional data points on the running software versions and possibly security-sensitive configuration settings. For our data acquisition, we adopt the high and well-accepted ethical standards of prior work conducting Internet-wide active measurements [177, 207, 9]. We further ensured that our measurements do not disrupt or harm evaluation targets, for example, through unintended resource or bandwidth consumption, and we put a process for a permanent opt-out of our measurements in place.

5.4.1 Preventing Disruption

In addition to standard ICMPv6 (Internet Control Message Protocol version 6) echo request to establish host reachability, we performed only basic service and version detection on open service ports. Misconfigurations, such as weak cryptographic keys, were only evaluated based on protocol handshake information. Similar to prior work, our independent evaluation of this measurement procedure yields that it is of negligible risk compared to the benefits provided to the community. This approach prevents misleading findings and reduces false positives, which would cast an incorrectly insecure picture of the evaluated hosts. Examples of such false positives are services listening on non-standard ports or services secured via tcpwrapper, which would also not be detected correctly by a standard port scan.

5.4.2 Subject Information and Opt-Out

A network administrator might misjudge our measurements for attacks, due to receiving alerts from an intrusion detection system deployed at the evaluated network. To inform the operators of the measured networks, we follow best practices [177] and provide a "usage notice" website reachable at both the IPv4 and IPv6 addresses of the measurement machine. The notice explains that the measurements are benign in nature, who is conducting them, how to contact the authors, and how to opt out of future measurements. We have not received any opt-out requests or related complaints.

5.4.3 Responsible Disclosure

We encountered several vulnerable systems and deployments during our evaluation. With the publication of our methodology, an attacker could use it to enumerate active IPv6 addresses and rediscover vulnerable devices and infrastructure. Therefore, we conducted a responsible disclosure process for our findings, having informed the affected parties. To prevent any possible harm, we contacted the individual parties and the responsible Computer Security Incident Response Teams (CSIRT). The responsible disclosure process has been completed for all our findings.

5.5 Evaluation

We first evaluate how our technique fares on an Internet-scale. We then look in-depth at various issues IPv6 networks exhibit in the wild, which prior studies have missed, possibly due to being unable to target and enumerate specific IPv6 networks or IPv6-only hosts. Our results underline the need for an active enumeration technique for future IPv6 security studies, instead of being able to rely on data collected at network vantage points.

5.5.1 Internet-wide Enumeration

First, we enumerate the entire IPv6 address space using our technique. To enumerate the address space more quickly, we seed our enumeration technique with IPv6 network prefixes that we obtained from aggregating a view on the global routing table (GRT). We aggregate this GRT from Border Gateway Protocol (BGP) dumps available from RIPE RIS [208] and Routeviews [209] following current best practices [210]. In addition, we leverage the enumeration technique of Fiebig et al. to establish a baseline [9].

We find that our technique performs favorably compared to the enumeration technique of Fiebig et al. (see Figure 5.3). Specifically, we perform better than the baseline for large prefixes. For instance, for network prefixes of size /32, the maximum allocation size for IPv6, we identify 3,770 more networks, while for networks of size /48, the general allocation size for IPv6, we find 2,649 more networks [211, 212]. Unfortunately, however, due to the delayed deployment of DNSSEC, our technique currently enumerates fewer different prefixes than Fiebig et al. for more specific nibbles in IPv6 addresses. We expect this behavior to change in the near future as the adoption of DNSSEC is increasing, which, in turn, allows our technique to enumerate even more addresses.

Interestingly, during our study we encounter 316 networks using DNSSEC that have an untrusted path from the root zone. In detail, of these 316 networks, 191 utilize NSEC and 125 have NSEC3 configured. This observation underlines that DNSSEC and DNS zones are not necessarily configured correctly in practice. Following the hierarchical concept of DNSSEC, there should not be a zone that is DNSSEC-signed that was not found by enumerating from the reverse zone root (ip6.arpa). DNS is strongly hierarchical by design, and following the tree-based key distribution and verification schema of DNSSEC, there should be no signed zone that is only reachable through intermediate unsigned zones. It further indicates that the seed-based approach we utilized not only reduces the overall runtime, but also discovers networks and enumerates hosts that would otherwise not be found by naïvely enumerating the reverse zone in a top-down fashion. In practice, the time to unblind IPv6 addresses is reduced further by exploiting the knowledge of total addresses in a reverse zone (the number of hashes that we collected online prior to unblinding) and address assignment strategies. Addresses are rarely assigned randomly, but instead follow incremental strategies or use stateless address auto-configuration, and allow us to direct our unblinding process in the search space and reduce its time further.

We find fewer records than Fiebig et al. While they enumerated 5.8M unique addresses using their technique in late September 2016 [9], and—with an improved version running on multiple hosts at the same time—over 10M in early 2017 [12], we merely found 2.2M addresses running the published toolchain on a single host (compared to 5.8M). We mostly attribute this to the significantly

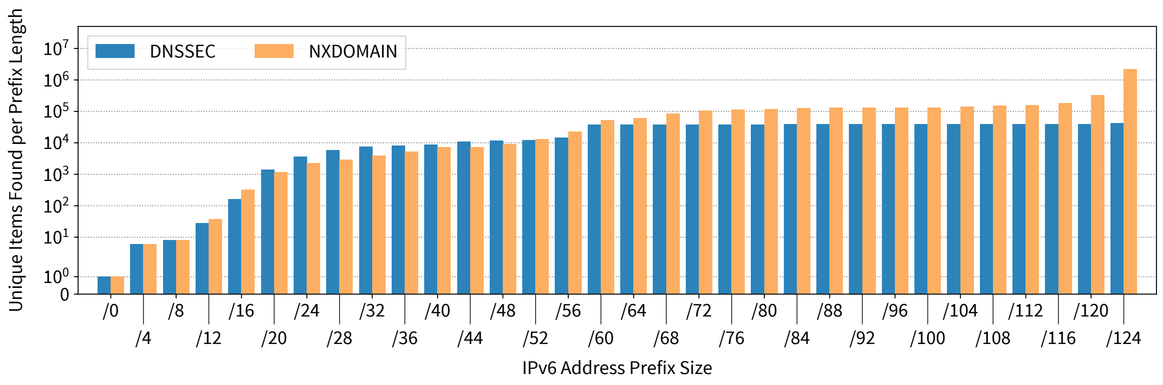


Figure 5.3: Records enumerated by our DNSSEC-based technique and the technique by Fiebig et al. [9]. Applied on a global scale, we identify more unique prefixes than the technique by Fiebig et al. for prefix lengths between /20 and /56, for example, 3,395 more networks with a prefix length of at least /44. For networks smaller than a prefix of /60, the number of discovered prefixes increases more slowly, because DNSSEC is not yet being frequently deployed at smaller leaf networks (compared to its wide-spread adoption for zones at the higher level). The deployment of DNSSEC for these smaller networks is expected to increase in the near future.

higher number of necessary requests of their approach compared to our more informed enumeration technique (Figure 5.2), leading to an increased impact of packet loss. Indeed, especially due to the higher number of requests, their technique can be detected and selectively mitigated with relative ease. Hence, our technique does not only provide more reliability by being harder to mitigate, but also puts less stress on networks. Both features are desirable when conducting large-scale active measurements.

In summary, we find that our technique shows great promise. We easily out-perform existing techniques for zones that are already DNSSEC signed. Our technique is only hampered by the current deployment state of DNSSEC for leaf zones. However, the adoption of DNSSEC is expected to increase even further in the future, as is the adoption of IPv6. Therefore, we expect our approach will be able to enumerate significantly more networks in the near future.

5.5.2 Observed Security Issues

Following the demonstration of the large-scale potential of our technique, we utilize it to survey network security issues in current IPv6 deployments around the globe. Specifically, we have scanned 338 different IPv6 networks, and we report detailed findings of the security posture of five different networks with different and diverse security requirements: *(i)* a French Internet service provider (ISP), *(ii)* a Ukrainian Local Internet Registry (LIR) and transfer broker (responsible to facilitate IP address space transfers), *(iii)* a European domain registry, *(iv)* a supercomputing facility in the United States, and *(v)* a large German university. The security issues we have uncovered in these networks illustrate that even experienced network operators from a variety of backgrounds might be unaware of the problems that a hasty IPv6 deployment can bring.

For each identified network of hosts, we perform the following two steps:

1. We look up the hostnames for the enumerated IPv6 addresses within the reverse zone, and then forward look up the hostnames to obtain the corresponding IPv4 addresses. If a hostname maps to a single IPv4 address, then we assume that IPv4 and IPv6 address point to the same physical host and compare open ports and available services through access via IPv4 and IPv6, respectively. If a hostname maps to multiple IPv4 addresses, we do not further evaluate its security as it would skew the comparative analysis because it is uncertain which IPv6 and IPv4 addresses correspond to each other.

2. We evaluate the enumerated hosts with nmap and we specify the command-line arguments *-Pn -O -sV –nsock-engine=epool -p1-10000 -sS -sU –max-retries 1* to identify potential security issues.

Network	Hosts			**Sub-Networks**
	IPv6-only	Dual-stack	Total	
French Internet Service Provider	2,069	66,545	70,818[*]	43
Ukrainian LIR	4,619	245[†]	4,864	611
European Domain Registry	130	119[‡]	249	0
United States Supercomputing Facility	28	1,343	1,371	1
German University	138	97[§]	235	0

Table 5.2: Number of IPv6 hosts enumerated and sub-networks identified

[*] We successfully unblinded 68,614 addresses within our timeout of 12 hours, and only 2,204 hosts remain blinded for a 96.90% success rate. [†] Two (2) hosts leak private IPv4 addresses via forward DNS lookups from two networks, and two (2) hosts point to IPv4 localhost addresses.

[‡] Five (5) hosts leak private IPv4 addresses via forward DNS lookups.

[§] Sixteen (16) hosts leak private IPv4 addresses via forward DNS lookups.

We responsibly disclosed our findings to the network operators for all networks that we have evaluated in the course of this chapter. We hope that our findings motivate network operators to evaluate the security of IPv6-connected devices on their network.

The different networks that we investigated in-depth vary in the way they deploy DNSSEC: one network deploys NSEC3 and the remaining four deploy NSEC. They also differ in size as the number of active hosts ranges from 235 to 70,818, with between 28 and 4,619 hosts classified as IPv6-only. We classify a host as IPv6-only if we were unable to confirm that its hostname, which we obtained from the reverse IPv6 zone, points to exactly a single IPv4 address in the forward zone.

An IPv6 network might be split into various sub-networks for specific purposes or regions. Unsurprisingly, the number of sub-networks differs quite a lot per network type: the French Internet service provider's network has 43 sub-networks, which likely correspond to different regions where they provide their services; the Ukrainian LIR delegates the most networks (611), most likely to its customers, some of which are government and law enforcement entities; the European domain registry and the German university do not have any sub-networks, possibly because of a central network operations center; and the United States supercomputing facility uses one sub-network, possibly for users of the computing cluster or the cluster itself. While we did not include the sub-networks in our evaluation, our technique can enumerate them readily as they are also DNSSEC-signed.

We are focusing our efforts on the following problems and discuss them separately: *(i)* for IPv4 and IPv6 dual-stack hosts, we look at all ports accessible via IPv6 but not via IPv4 and vice-versa; *(ii)* for IPv6-only hosts, we look at all services that can be accessed externally and which could be a security risk; and *(iii)* potential privacy concerns for names in the reverse zone. Particularly, we investigate more closely:

- **Remote access protocols:** Secure Shell (SSH), Telnet, and remote desktop sharing.
- **File sharing:** Apple Filing Protocol (AFP), FTP, HTTP, Server Message Block (SMB), and WebDAV.
- **Monitoring and system management:** Nagios Remote Plugin Executor (NRPE), Simple Network Management Protocol (SNMP), Intelligent Platform Management Interface (IPMI), and management interfaces for machine virtualization (Hyper-V, VMware).
- **Network management via routing protocols:** Open Shortest Path First (OSPF) as an interior gateway protocol, and the Border Gateway Protocol (both iBGP and BGP).

5.5.3 Dual-stack Analysis: IPv4 vs. IPv6

We contrast the security deployment of IPv4 and IPv6 by taking an in-depth look into some existing networks. In total, we investigate more closely the differences in security measures of accessing 68,349 hosts through IPv6 compared to through IPv4. The hosts are part of the networks of five different institutions with varying security requirements.

The infrastructure network of the French Internet service provider is the most populous network, of the ones we have investigated more closely, with 66,545 dual-stack hosts. Fortunately, most hosts are secured appropriately. In fact, much to our surprise, hosts following incremental IPv6 address assignment pattern exhibit the same or better security, that is, the same or less exposed ports through IPv6 than via IPv4. This might be the case because the services are configured to listen on their respective IPv4 address only, instead of the default to listen on all available addresses (IPv4 and IPv6) or interfaces, and, thus, no access via IPv6 is possible. On the other hand, hosts who have taken on globally routable addresses via stateless address autoconfiguration (SLAAC) do exhibit worse security. Alongside world-readable Apple file sharing we discovered open ports for access to management interfaces of Cisco switches via Telnet, access to Hewlett Packard StoreFabric network storage devices (both client and management interface ports), as well as read-only SNMP access for various networking devices (access might not be restricted to read-only, but without potentially disrupting infrastructure, we are unable to confirm whether access is read-write; therefore, we report all SNMP access as read-only).

Different is the network of the supercomputing facility in the United States, for which we enumerated 1,371 IPv6-capable hosts, with 1,343 of them being dual-stack. Although a significant amount of services, like HTTP(s), FTP(s), IMAP(s), SMTP(s), POP3(s), are available on the network, almost all of them are accessible via IPv4 and IPv6 and we consider them as intentionally open and without additional security risk. Of all 1,371 hosts, 828 hosts assigned themselves IPv6 addresses through SLAAC, while the remaining 543 hosts have IPv6 addresses assigned incrementally with gaps due to jumps at earlier nibble boundaries, confirming that guided search for enumeration has substantial benefits. There was no difference in security for incrementally assigned addresses and automatically assigned address through SLAAC, but hosts remained more open to attackers via IPv6 than IPv4. Specifically, we still encountered services accessible via IPv6 that are likely unintentionally accessible as they are security-sensitive, including, but not limited, to BGP (secured via tcpwrapper for some hosts only), Telnet access to Cisco routers, and access to Microsoft's Active Directory.

Similar to the supercomputing facility, a variety of IPv6 hosts on the German University's network expose SSH, HTTP, and FTP. Again, we observed the same ports being publicly accessible via IPv4. Since universities often provide HTTP and FTP mirrors of open-source software, and SSH is generally considered secure, we do not consider them potential security problems. Alarmingly, however, we still determined a plethora of potentially critical security problems. In particular, publicly accessible via IPv6 but not IPv4 are: interior BGP and exterior BGP for 57 hosts, old SSH versions on two Cisco switches, SNMP on 35 hosts, Nagios Remote Plugin Executor for 38 hosts, a portmap version on 38 hosts that can be exploited to launch reflected and amplified denial of service attacks [213], and fingerd on one host. Especially concerning are the exposure of BGP, portmap, and SSH access on the two Cisco switches, which used weak host keys (512-bit RSA).

We observed no significant differences in security for dual-stack hosts for the European domain registry or the Ukrainian LIR. Yet, over all networks, the security of hosts whose addresses appear assigned through SLAAC, that is, automatically based on the hosts' MAC addresses, is worse than those for which the address is assigned incrementally.

5.5.4 Security Posture of IPv6-only Hosts

We also enumerated hosts that are single-stack and thus are only reachable via IPv6. Interestingly, some early proponents of IPv6 without prior experience operating IPv4 networks exhibited the worst security measures and exposed administrative, infrastructure, and network management interfaces through IPv6 to the world. Most likely, they assume more secure defaults and might not know better given a lack of experience.

Unfortunately, although experience helps to mitigate some issues, it is not a silver bullet. An example is the infrastructure network of a major LIR in the Ukraine of which almost all hosts (4,619 of 4,864 hosts) are reachable only through IPv6. However, since the network operator has extensive experience operating an IPv4 network, we were expecting a relatively secure network. Regardless of prior operating experience, we discovered critical security issues on two IPv6-only hosts, both of which do not have an entry in the forward zone. Both hosts expose the Quagga routing software's management port as well as BGP via IPv6 and could be used to control routing for all the LIR's sub-networks, which include law enforcement and government entities. Although already concerning, we detected an old version of Quagga (0.99.22.1) at a core network router, which is potentially vulnerable to a remote code execution and a denial-of-service attack [214, 215]. Unfortunately, the critical security issues did not stop there, and, even more alarming, we discovered a vulnerable version of SuperMicro IPMI at an IPv6 address that was assigned automatically (via stateless address autoconfiguration), which not only allows full remote execution, but it allows an attacker to gain practically physical access to the machine remotely.

We manually confirmed that all vulnerable hosts were not part of any public dataset used by Czyz et al. [37], which further emphasizes the need for practical IPv6 address enumeration techniques, and it illustrates that existing datasets might in fact cast a skewed result on the security state of IPv6-connected devices. Considering that Czyz et al. collected their dataset from ANY records on the forward zone, it is clear why prior work did not include it: the hosts' IPv6 addresses do not appear in the forward zone at all, but only appear in the reverse zone.

As in the case for dual-stack hosts, we reach the conclusion that the security posture of IPv6-only hosts varies in the way addresses are assigned. For devices who leverage SLAAC security is worse than for those who have addresses assigned manually or via DHCPv6.

5.5.5 Privacy Issues

A possible security and fundamental privacy issue we discovered is the leakage of meaningful host-names through the automatic population of the reverse zone.

In case of the European NIC, regardless of the deployed security measures at those hosts, the respective hostnames leaked information about their use case: configuration management and deployment, system and network monitoring, logging, version control, bug tracking, as well as registry internal infrastructure (authentication, transfers, validation). Although not a security issue necessarily, it opens an avenue for reconnaissance for attackers and it might provide the extra information that is necessary to circumvent security measures that have been put in place.

Similarly, for the French ISP, stateless autoconfigured IPv6 addresses leaked that Apple, Cisco, and Hewlett Packard devices are on the network. From reverse DNS entries, we further determined that the Apple devices are laptops and based on a combination of reverse DNS, MAC address, and service and version detection on open ports, we can determine that the Hewlett Packard devices are HP StoreFabric storage devices, while the Cisco devices are top-of-rack switches. Additionally, based on hostnames themselves and routes taken to hosts, we believe that we have enumerated hosts in four data centers or office buildings: two in Paris, one in Lyon, and one in Toulouse.

Significantly more concerning is the case of the United States supercomputing facility though. The way the reverse zone is used and populated allows us to track employees' devices and even their location. Specifically, we were able to track 13 phones and 10 laptops of employees over time and we correlated their working hours, and their presence across two buildings. Of the ten laptops, three laptops are connected via Ethernet and Wi-Fi, allowing higher fidelity tracking, and one person is using two laptops. From reverse zone information, we can also determine that four people work in the main complex, while another nine work in an adjacent and affiliated research center. We manually verified this to be true through its website.

Tracking is made possible due to the automatic populating of the reverse zone. To track working hours, regular liveness probes are sufficient (e.g., via ICMPv6). On the other hand, tracking users across buildings is possible in two different ways. First, through liveness probes over multiple network prefixes, since the remaining nibbles of the address stay constant (due to SLAAC), and, second, through forward DNS lookups on the hostname under a different subdomain (the subdomain used for Wi-Fi access in the buildings is different). More fine-grained location tracking, up to floors and even rooms, is sometimes possible through tracing the route to the host and investigating intermediate router hostnames more closely. The privacy implications of automatically populating the reverse zone are further amplified by host and node information, such as names in "jane-iphone" or "doe-notebook" (with only one person with the first name Jane or last name Doe working at the facility).

5.5.6 Discussion

From our evaluation it is apparent that IPv6 hosts can be, and sometimes are, secured in the same manner and to the same level as IPv4 hosts. However, as of today, IPv6-connected hosts still lag behind in regard of security when compared to IPv4 hosts, and their improvement progress must be monitored and evaluated closely as to not relive the "Wild West" days of the Internet from the 1990s.

Furthermore, we discovered that stateless address configuration can be a significant security problem if network-based firewalls are not deployed. Our findings show that devices take on global IPv6 addresses automatically if they are advertised an IPv6 route, regardless of whether they are secured appropriately. Since some networks are secured appropriately and since the self-assigned IPv6 addresses do not fit into the networks' address assignment pattern, we suspect that the devices with self-assigned addresses have worse security because the network operators are unaware of their behavior and might assume that they do not support IPv6 yet, possibly because support might have been added with a software update after deployment. We believe that we encountered these cases because IPv6 is sometimes enabled by default in newer firmware versions of switches and routers, which might be installed for part of a data center only, for example, through a staggered deployment, and because laptops might normally connect to IPv4-only networks exclusively, but sometimes connect to a network where an IPv6 route is advertised. For them, host-based firewall rules might not be configured for IPv6 yet, thus exposing the machine completely to the rest of the Internet.

5.6 Mitigation

In response to zone-walking attacks against DNSSEC, a variety of defenses have been proposed. Some of these approaches would also prevent enumerating IPv6 addresses from the reverse zone. However, the proposed defenses have significant shortcomings and some require to fully trust the nameserver with the authoritative zone-signing keys, a practice that DNSSEC strongly discourages. We discuss how those techniques would impact our approach and, if adopted, what other issues they bear.

5.6.1 Reverse Zone Modifications

A straightforward solution to prevent IPv6 addresses from being enumerated via DNSSEC on the reverse zone is to drop the reverse zone completely or to not deploy DNSSEC on it. Not keeping any reverse zone information for IPv6 addresses has significant problems though, which would render the affected IPv6 addresses almost entirely useless in practice. Nowadays, reverse zones are used to protect against spam and other inconveniences and the lack of a reverse entry for an address is considered a lack of trust and "sign of trouble." For instance, almost all incoming email servers (SMTP) are configured to look up the reverse name and reject incoming mail from IP addresses that do not hold a valid reverse DNS record. Therefore, not keeping a specific IP address in a reverse zone immediately limits the use of that address. For instance, in the case of a hosting or access Internet service provider, it would effectively prevent its customers from sending email.

Alternatively to dropping the reverse zone entirely, one could choose not to deploy DNSSEC for it. However, similarly as to verifying that an IP address has a reverse entry, some SMTP servers are trusting signed and valid reverse entries more and service them quicker (e.g., no greylisting). In turn, the decision to not sign the reverse zone can degrade the overall quality of service but it would not prevent the service to be used at all. In addition, this technique exposes the reverse zone to the known problems of DNS that have been solved by DNSSEC. For example, by effectively removing any authenticity on a zone one enables malicious nameservers to return bogus responses (again).

In both cases, the respective authority for the reverse zone needs to decide on the trade-off: whether she prefers to degrade quality of service, or whether she wants to prevent zone-walking and protect the privacy of addresses on her network. It is understandable that network operators prefer to guarantee a high quality of service over preventing zone-walking attacks, particularly considering that IP addresses will become public during communication with other hosts anyways. Thus, hiding them is merely a misguided attempt at security through obscurity. Furthermore, security management of the hosts that could be enumerated is often outside of the responsibilities of the network operator herself (instead, a system administrator is often responsible) while the quality of service is her métier.

5.6.2 Minimally Covering NSEC Records

An alternative approach to preventing zone-walking attacks via already existing DNSSEC record types, such as NSEC3, was proposed by Weiler et al. [133]. Instead of signing the zone offline and thus, by requirement, introducing large spans for NSEC3 records, Weiler et al. suggest to sign records online and to return *minimally covering NSEC3 records* on demand. For instance, a minimal covering NSEC3 record for a non-existing domain n with hashed name h_n would fake the previous existing hash as $h_n - 1$ and next existing hash as $h_n + 1$. For proving the denial of existence for n, it is irrelevant whether $h_n \pm 1$ actually exist, if they do not exist the denial record is considered a "white lie."

Minimally covering NSEC3 records prevent zone-walking attacks effectively. However, this approach requires online signing and thus requires the full zone-signing secret key to be available at the nameserver. If the zone-signing key is deployed to the authoritative nameservers, then any single compromised authoritative nameserver results in a complete zone compromise, and any bogus and possibly malicious responses can be signed and returned. This would be a direct contradiction to the goals of DNSSEC and its operational practices [216]. Given the computational overhead of online signing DNS responses and its potential security risks, minimally covering NSEC records have so far been adopted only hesitantly.

5.6.3 NSEC4

Another attempt to revolutionize DNSSEC's denial of existence records was the proposal of NSEC4 by Gieben et al. [217]. However, the respective Internet-Draft does not propose any techniques that would prevent zone-walking, and thus cannot be considered a mitigation technique. Instead, it introduces performance optimizations for denials of existence of wildcard records and the opt-out flag. The draft has expired in January 2013 and has not been renewed. The optimizations have been integrated into NSEC5.

5.6.4 NSEC5

Goldberg et al. [218] introduce NSEC5 as a solution to provably preventing zone enumeration attacks. The adoption of NSEC5 would prevent enumeration of active IPv6 addresses through the reverse zone, but, it comes at the significant cost of requiring additional online asymmetric cryptography operations. In fact, the additionally incurred cost for online signing when deploying DNSSEC renders nameservers subject to denial of service attacks and chosen-plaintext attacks [133], which is why it might have been rejected by industry leaders in favor of signing zones offline. Specifically, denial of service attacks due to asymmetric cryptography can be abused in many more ways for DNSSEC over similarly authenticated protocols, like TLS, because it uses UDP for the transport protocol instead of TCP. The latter are less impacted because they normally do not perform any cryptographic operations prior completion of the TCP handshake, which acts as a way to ensure that the connection between server and client is intended. On the contrary, in the case of DNSSEC, no such protection exists and cryptographic operations must be performed when receiving the first and only packet. Furthermore, it is more prone to abuse because of reflection and spoofed addresses. Nonetheless, we support the authors' effort to have NSEC5 become an Internet standard. The additional computational cost incurred on the nameserver and the increased risk of denial of service attacks might be a reason why the Internet-Draft remains a work in progress, and had to be renewed by the authors prior to expiration five times already [219]. Without sufficient industry interest and without an implementation except for the reference implementation for Knot DNS being available (although NSEC5 solves a known problem and was proposed in mid 2014 [220], no implementation for the BIND nameserver exists), wide adoption of NSEC5 in the (near) future appears highly unlikely, allowing our approach to be used in practice.

If NSEC5 would be deployed for a zone, an attacker who is trying to enumerate that zone would need to obtain the NSEC5-signing-key. Once the attacker has obtained the key, she can degrade NSEC5's security guarantees to those of NSEC3, walk the zone, and, in turn, enumerate IPv6 addresses.

5.7 Conclusion

We introduced a technique to enumerate part of the active IPv6 address space as a starting point to evaluate the security state of IPv6-connected hosts. Our approach leverages DNSSEC-signed reverse DNS zones to enumerate active IPv6 addresses that can later be scanned through readily available tools, such as nmap. Although NSEC3 should protect from zone-walking attacks, the combination of the well-defined structure of IPv6 addresses in the reverse zone, and the implications of the disclosure of the record types for the previous and next hashes in the NSEC3 chain counteract its protective impact. In turn, it reduces the search space needed to break the hashed addresses to as little as 2^{64}, with additional reductions in practice through intelligent search due to incremental (e.g., manual or via DHCPv6) and MAC address-based (stateless address autoconfiguration) address assignment schemes. Exploiting these intricacies, we successfully demonstrate that it is practical to enumerate active IPv6 addresses at scale in the face of NSEC3. Furthermore, to the best of our knowledge, we are the first to introduce systematic and practical methodology to enumerate IPv6 addresses through NSEC and NSEC3 based DNSSEC-signed reverse zones by exploiting previously ignored subtleties in the interplay of reverse zones and DNSSEC.

Based on the enumerated address set, we evaluated the state of security of IPv6 hosts and we have shown that many are insufficiently secured. Specifically, IPv6-enabled systems often expose critical infrastructure or sensitive and privacy-concerning information to the outside. For instance, we discovered various routers exposing unsecured Telnet access, or internal file shares being exposed via IPv6, and that the analysis of hostnames in the reverse zone can leak employees' working hours and locations. Furthermore, from our comparative analysis of scanning dual-stack hosts via IPv6 and IPv4, we conclude that one main cause is that globally routable IPv6 addresses are assigned automatically to the machines. It appears that hosts assigning themselves a globally routable IPv6 address is a practice some system administrators are unaware of, as the respective hosts are almost always properly protected from unauthorized access via IPv4.

Finally, we discussed mitigation mechanisms that could protect against zone-walking in the presence of DNSSEC and, in turn, could prevent IPv6 address enumeration attacks through DNSSEC-signed reverse zones. Ultimately, we reach the conclusion that the proposed defenses suffer from shortcomings that will prevent them from being adopted in practice in the (near) future. Therefore, we expect our approach to continue being a viable IPv6 address enumeration technique and to further improve with the continued deployment of DNSSEC.

This page intentionally left blank

Chapter 6

Related Work

A large amount of prior work has been carried out to address Internet abuse of varying scale. Hereinafter, we first discuss related work on tackling web-based threats, from detecting website defacement, to defeating phishing through image-based analyses, to detecting malicious code, and, finally, to leveraging and understanding the dynamic nature of the web to recognize intrusions. Following, we discuss related work in the areas of DNS security and measurements, IP address squatting and takeover attacks, the security of domain-based certificate and trust validation, and cloud security, specifically issues grounded in resource sharing. Finally, we discuss prior work in the areas of IPv4-wide security scanning to measure abuse and abuse potential, enumerating active IPv6 addresses, and privacy issues with respect to DNSSEC and zone enumeration.

6.1 Website Defacement Detection

Similar to Meerkat, Davanzo et al. [76] introduce a system that acts a monitoring service for website defacements. Their system utilizes the website's HTML source code for detection, and its features were selected manually based on domain knowledge acquired a priori, making the system prone to concept drift. On their, comparatively, very small dataset containing only 300 legitimate websites and 320 defacements, they achieve false positive rates ranging from 3.56% to 100% (depending on the machine learning algorithm used; suggesting extreme underfitting and overfitting with some algorithms), and true positive rates between 70.07% to 100% (in the case of simply classifying everything as a defacement; i.e., extreme underfitting). Overall, these results are significantly worse than MEERKAT, both in terms of false positives (1.012%) and true positives (97.878%).

Bartoli et al. [77] propose Goldrake, a website defacement monitoring tool that is very similar to the tool proposed by Davanzo et al. and leverages a superset of their features. To learn an accurate model, Goldrake requires knowledge about the monitored website to learn website-specific parameters. However, it is unclear how well Goldrake detects defacements in practice because it is evaluated on a small and (likely) non-diverse dataset comprised of only 11 legitimate websites and 20 defacements, on which it performs poorly with a high false negative rate (27%).

Medvet et al. [221] introduce a defacement detection system based on work by Bartoli et al. and Davanzo et al., but the detection engine is replaced by a set of functions that are learned through genetic programming. The learned functions take the features by Bartoli et al. and Davanzo et al. as input, but classification is more accurate on a dataset comprised of 15 websites (between 0.71% and 23.38% false positives, and about 97.52% true positives). It is, again, unclear how the system would fare in a real-world deployment because of the small and (likely) non-diverse dataset.

Note that all text-based approaches have major weaknesses, similar as those encountered in spam and phishing detection, such as using images to show text to evade detection. MEERKAT does not suffer from these shortcomings.

Lastly, most commercial products that detect website defacements are built upon host-based intrusion detection systems to monitor modifications of the files on the web server, for example, via file integrity checks (checksums) [222, 223]. Therefore, those approaches bear the major drawback that they can only detect the subset of defacements that modify files on disk, and that they cannot detect other defacement attacks, such as through SQL injections, even when the defacements look exactly the same to the website's visitors. Meerkat does not have this blind spot and detects these stealthier attacks.

6.2 Image-based Detection in Security

No prior work, to the best of our knowledge, applies image-based methods to detect defacements, which is why we compare Meerkat to prior work that visually detects phishing pages, or leverages image-based techniques as part of a larger system.

Medvet et al. [224] propose a system to detect if a potential phishing page is similar to a legitimate website. The system leverages features such as parts of the visible text, the images embedded on the website, and the overall appearance of the website as rendered by the browser for detection. Similarity is measured by comparing the 2-dimensional Haar wavelet transformations of the screenshots. Their system achieves a 92.6% true positive rate and a 0% false positive rate on a dataset comprised of 41 real-world phishing pages.

Similarly, Liu et al. [225] present an anti-phishing solution that is deployed at an email server and detects linked phishing pages by assessing the visual similarity to the legitimate page, but only when analysis is triggered on keyword detection. The system identifies phishing pages by comparing the suspicious website to the legitimate website, which it does by measuring similarity between text and image properties, like the font size and font face used, or the source of an image.

Although detecting phishing pages by comparing the similarity of two websites is sensible, for defacements the difference between them is more interesting. Instead of creating a visually-similar page to trick users into disclosing their credentials, a defacer wants to promote his message. Adopting existing phishing detection systems to detect defacements instead, that is, by comparing if the website looks different from its usual representation, however, bears two problems: *(i)* the usual representation must be known and stored for comparison, and *(ii)* false positives are much more likely if the website is dynamic or if it shows regularly-changing ads.

Anderson et al. [60] introduce image shingling, a technique similar to w-shingling, to cluster screenshots of scams into campaigns. However, in its current form, image shingling cannot be used to detect defacements as it is trivial to evade the clustering step with only minor modifications that are invisible to the human eye, and, thus, the technique is unsuitable for a detection system in an adversarial context.[1]

Nappa et al. [226] leverage perceptual hashing to group visually similar icons of malicious executables, assuming that a similar icon suggests that the two executables are part of the same malware distribution campaign. While it is theoretically possible to detect defacements through perceptual hashing-based techniques and comparing the distance of the hashes, it is impractical to do so on a large scale and in an adversarial context. For once, one must have a ground-truth screenshot that is close enough to the screenshot that one wants to classify; if ground-truth is not available or slightly too different, a system based on perceptual hashing will be unable to detect the defacement. Furthermore, classification is not constant in the number of defacements the system has seen in the past: For each new screenshot we would want to classify, we would need to compute the distance to the hashes of at least some (or all) of the previously-seen defacements.[2]

Grier et al. [227] introduce their own image similarity measure to cluster malicious executables that have similar looking user-interface components after being executed in a dynamic analysis environment. Two images are considered similar if the root mean squared deviation between the images' histograms is below some manually-determined threshold. Clearly, a defacement system based on this technique is not suitable in an adversarial context: An attacker can (and eventually will) simply

[1]The authors acknowledge the shortcomings in an adversarial context in Section 4.2, but they do not discuss any remediation techniques.

[2]Therefore, detection time increases with each observed defacement; it is at best in $O(\log n)$ and at worst in $O(n)$, with n being all observed defacements.

change the colors slightly or add dynamic content, so that the root mean squared deviation is above the threshold, but remains visually similar to the human eye. Furthermore, as for Nappa et al. [226], one needs to pair-wise compare the histogram of the screenshot one wants to classify to some or all of the already-seen defacements.

Meerkat does not suffer from any of these shortcomings: First, it learns high-level features on the defacements' general *look and feel* to detect also previously unseen defacements, and, second, its classification time is constant in the number of already-seen defacements.

6.3 Detection of Malicious Code

Numerous papers have been published on detecting malicious activity in websites. To the extent of our knowledge, no prior work exists that actively searches and finds previously unknown malware infection campaigns. The majority of prior work focus on dynamic analysis of JavaScript in instrumented environments or on rendering websites in high-interaction client honeypots. It is important to recall that Delta is complementary and provides additional information: the infection campaign and the responsible node of the DOM tree.

Eshete et al. [228] discuss the effectiveness and efficacy issues of malicious website detection techniques. Approaches from blacklists, to static heuristics, to dynamic analysis are compared in their detection accuracy and time spent analyzing the website. A major argument on the weaknesses of previous work is their missing discussion on the necessity of episodic re-training or online learning capabilities, to keep up with the ongoing evolution of web-based malware, and to prevent the evasion of deployed detection systems.

Cova et al. [90] introduce the system JSAND, to detect and analyze drive-by download attacks and malicious JavaScript in an instrumented environment. The system leverages a comprehensive dynamic analysis approach by instrumenting JavaScript to extract a variety of different features from redirection and cloaking, to deobfuscation, to observing heap exploitation. They compare JSAND to client honeypots, such as Capture-HPC and PhoneyC, as well as the anti-virus engine ClamAV. It shows a much lower false positive (0%) and false negative rate (0.2%) than all other approaches (5.2% to 80.6%), while taking an average of 16.05 seconds to analyze a website. CaptureHPC, the closest system in terms of accuracy takes 20 seconds per sample.

Canali et al. [91] extend the dynamic analysis system JSAND by implementing a pre-filtering step. The main goal is to prevent the submission of certainly benign websites to the dynamic analysis system and, in turn, reduce the time spent on analyzing benign samples, that is, the system assigns to a false negative a much higher cost than it does to a false positive. The filter method leverages a C4.5 (J48) decision tree and a diverse set of features spanning from the HTML content, to the JavaScript code, to information about the host, to uniform resource location (URL) patterns. They evaluate their filter on a dataset of 15,000 websites and compare it to similar methods by Seifert et al. [110] and Ma et al. [118]. Both other methods yield more false positives and false negatives, but process up to 10 times more samples in the same time.

Provos et al. [23, 92] introduce a system to detect URLs to malicious websites. However, they are not considering legitimate infected websites in general, as their approach is restricted to detecting the inclusion of exploit pages, and hence their approach is complementary to our system's capabilities. Their system uses a proprietary machine learning algorithm to classify URLs based on features like their use in "out of place" inline frames, obfuscated JavaScript, or links to known malware distribution sites. Besides detecting 90% of all malicious landing pages with 0.1% false positives, they validate previous work by Moshchuk et al. [229] that infection vectors are inserted into legitimate websites through exploiting vulnerabilities, advertisement networks, and third party widgets.

Delta complements prior work by being able to search and find known and unknown infection vectors throughout the Internet, which prior work can then leverage to train their system for better detection accuracy and increased user protection.

6.4 Web Dynamics in Security

Maggi et al. [33] introduce a web application intrusion detection system, which is able to learn about changes made to the web application. The problem of web application concept drift is addressed by learning how the web application is accessed by a legitimate user and employing an unsupervised classification algorithm. Features include, for example, a sequence corresponding to the order in which websites are accessed or how web page parameters are distributed. The presented technique is orthogonal to Delta: The main goal is not to find new infection campaigns or to protect the visitor of a website, but rather to protect the integrity of the web application. Protecting a normal, wandering user would require intrusion detection and protection of all websites the user visits, since the access pattern, on which the system is based, depend on the underlying architecture of the website. A global deployment to protect users, although possible theoretically, is practically impossible.

Davanzi et al. [76] study a similar approach for detecting the impact of web dynamics. They introduce a system to detect if changes made to a website are defacements, which might cause serious harm to the organization, monetary or reputation-wise, or if they are legitimate, officially approved content changes. However, they explicitly point out that their approach does not work with malicious modifications because their approach detects changes that are visible to the end-user, which is the exact opposite of how malicious infection vectors are placed in practice. In detail, they employ anomaly detection to regularly visit and monitor a set of 300 websites actively and detect if changes made to the website constitute a defacement or not.

Delta, on the other hand, leverages web dynamics to derive additional information from observed changes across a large number of websites, primarily a change in behavior (i.e., introducing maliciousness), but also the impact and scale of changes, and commonalities in the operation of the websites. This information then guides a human analyst or analysis tool to more quickly determine the root cause for a website being attacked and turning malicious, and, thus, facilitates removal of the malicious behavior.

6.5 DNS Security

The domain name system (DNS) is a critical service in the Internet ecosystem and prior work has studied DNS security extensively. Bell and Britton hold a patent in which they describe how a host can be taken over by assigning the same IP address to a virtual interface on another system [230]. Yadav et al. report on domain-flux practices in botnets, a technique in which a domain generation algorithm is used to generate many domains, of which the operator only needs to control one to remain in control of her botnet [231]. However, to some degree as the dual of exploiting stale DNS records, one can register a single or multiple of those domains to take over a botnet, and it has been done successfully by Stone et al. [232]. Liu et al. conducted a study similar to our problem analysis for Cloud Strife [233]. However, methodological challenges and limitations of their datasets lead them to an under-estimation of the impact of stale DNS records in cloud scenarios. Indeed, contrary to them, we find that the problem of stale DNS records is amplified by multiple orders of magnitude. We further systematically analyze the practicality of acquiring the previously-used cloud IP addresses, discover use-after-free attacks based on DNS caches, and we propose a usable mitigation technique to automatically validate certificate issuance.

Instead of relying on correct DNS responses, bit-squatting exploits random bit-flips in DNS requests to lure clients to malicious or phishing websites [234]. Different from our attack, bit-squatting relies on integrity errors that occur at random and thus is not as targeted as our attack. Furthermore, exploiting integrity errors, it can be mitigated easily via hardware and software, for example, by adopting DNSSEC and leveraging its integrity guarantees. Similar to our technique, typo squatting can be used to lure clients on malicious websites [235, 236, 237]. It remains important to note that in a typo-squatting attack, the attacker needs to register a new domain and hope that users visit that domain. For our attack, although the window of opportunity might be shorter, the attack is significantly more severe: It is impossible for users to tell whether they are in fact being attacked, as domains and IP addresses have residual trust, and any connection might be marked trusted by the browser due to domain-validated TLS certificates. Indeed, Zdrnja et al. demonstrated an approach to detect typo-squatting attacks from mined DNS data [238]. Different from prior work, our study focuses on the vulnerabilities of stale DNS records pointing to cloud IP addresses, we conduct comprehensive measurements, and we propose a mitigation to retain the convenience of domain-validation for certificate issuance.

6.6 IP Address Squatting and Takeover Attacks

Taking over IP addresses is a well-known security problem. The most common and well discussed attack method aims to take over entire network prefixes using BGP, which can be easily observed and will be scrutinized quickly [239]. Wählisch et al. demonstrated a method to detect such takeovers using RPKI [240]. Ballani et al. conducted a study investigating prefix hijacking in 2007 [241], while Zhang et al. developed first defense methods against such attacks [242]. In 2015, Gavrichenkov demonstrated that modern domain-validated TLS certificates (and thereby HTTPS) can be broken using prefix hijacking [157]. Attackers with more powerful capabilities on the network path between a client requesting a certificate and a CA do not even have to perform prefix hijacking, but instead can easily exploit IP address squatting, as they are already on the path. Cloud Strife, on the other hand, details a new attack vector to conduct IP address squatting, which is practical, and time- and cost-efficient to launch.

6.7 Certificate Validation Security

The security threats we studied in Chapter 4 tie in with modern, domain-based, certificate authorities and their surrounding security challenges. Various efforts currently track the adoption of Let's Encrypt [243, 244]. In general, the security implications of domain-based certificate validation are widely accepted. In their analysis of the HTTPS/TLS trust ecosystem, Clark et al. [245] place great trust in DANE [165] to mitigate this issue. Apart from DANE, Certificate Transparency [141, 142] is considered the ideal reactive mitigation for maliciously and wrongfully obtained certificates and has received significant attention recently. The DNS certificate authority authorization (CAA) record might reduce the impact of IP use-after-free attacks to some degree [164], as it limits the CAs that are allowed to issue a certificate for a specific domain, and, thus, force an attacker to request a certificate from these CAs. However, our analysis shows that current domain validation in trust ecosystem is susceptible to use-after-free attacks regardless of CAA records. In fact, the only way to defend against use-after-free attacks through CAA is to restrict certificate issuance in its entirety, which then raises problems when the certificate expires while also relying on automatic certificate renewal setups, such as those recommended by Let's Encrypt, in which case automatic DNS zone updates are required (which become difficult in the presence of DNSSEC). Overall, relying on CAA would require numerous compromises in terms of certificate lifetime management and DNS zone maintenance, while still providing a potential (small) window of opportunity for an attacker whenever the CAA record needs to be relaxed to allow certificate renewal. We introduced a mitigation that incorporates existing trust of a name into the validation process and can protect against these attacks.

6.8 Cloud Security

Concurrent with the increasing adopting of cloud services, cloud security has drawn more research attention. Chen et al. provided a contemporary summary and analysis of cloud security issues [246], and indicated problems of shared resources. Similarly, Subashini and Kavitha provided a comprehensive analysis of security challenges in cloud scenarios [247]. Their analysis of IaaS platforms only includes similar issues to those approached by Ristenpart et al. [122]. Specifically, Ristenpart et al. exploit shared resources in IaaS environments to facilitate cross-VM side-channel attacks. However, they focus on physical computing resources and they do not investigate issues induced by logical resource sharing, for example, access to the same IP address pool. Jensen et al. focus on classical web attacks, especially in SaaS (Software-as-a-Service) scenarios [248]. Takabi et al. discuss the overall issue of IP squatting that is related to secure handling of provisioning and multi-domain cloud platforms with shared resource pools [249]. Zhang et al. investigate access control and trust management in the context of multi-tenant environments [250]. Cloud Strife is orthogonal to prior cloud security research, and it focuses on the certificate ecosystem vulnerabilities as it is being used in combination with cloud services.

6.9 IPv4 Security Scanning

Internet-wide scans have become an important tool for applied security research. They are imperative to identify and understand the impact of new vulnerabilities or common misconfigurations, like Heartbleed or DROWN. Heninger et al. scanned the IPv4 address space for weak cryptographic keys used by TLS and SSH servers [207]. Alarmingly, they discovered shared secret keys due to a lack of entropy during key generation, and they were even able to recover secret keys. Aviram et al. discovered DROWN, a new attack that exploits flaws in SSLv2. To determine its practical impact, they scanned the entire IPv4 address space and identified that 33% of all HTTPS servers were vulnerable [251].

These discoveries have been made possible by various advances around Internet-wide scanning. Heidemann et al. performed one of the first Internet-wide scans by sending ICMP messages to all allocated IPv4 addresses to identify reachable hosts [252]. Although enumerating all reachable hosts took multiple months to complete, the study clearly indicated the potential and benefits of large-scale probing. In 2013, Durumeric et al. developed ZMap [177], a fast scanning tool that can scan the entire IPv4 address space in under five minutes given the right conditions. They further discuss guidelines and best practices in using this tool to perform Internet-wide scans. We support their guidelines and took similar precautions to minimize the impact of our measurements.

6.10 Enumerating and Scanning IPv6 Addresses

While Internet-wide scans have become a common tool in the IPv4 world, measurements for IPv6 are still lagging behind. Specifically, three distinct research directions have been pursued: prefix-based measurements, studies based on client-centric vantage points, and, the most neglected, server-centric and security motivated studies.

Monitoring and measuring the IPv6 deployment has been of growing interest ever since the IPv6 standard was introduced. Large service providers and vendors, such as Cisco or Google, have since been tracking the use of IPv6 [253, 181, 189]. Similarly, Dhamdhere et al. analyzed historical BGP data to determine IPv6 deployment at the autonomous system (AS) level, for which they were able to determine that it was lagging behind at edge networks [254]. While some publicly accessible resources exist about the allocated IPv6 prefixes, for example, prefix assignments from IANA [255], those resources only provide a high-level view and do not allow exact measurements. Furthermore, considering that the smallest recommended end-user allocation for IPv6 networks is a /64 network (2^{32} *times* the size of the entire IPv4 address space), it is impossible to tell which part of an announced prefix is allocated or in active use. Therefore, it is impossible to provide insights into IPv6 address utilization from prefix information alone, and efficiently enumerating active IPv6 addresses remains a challenge.

To characterize IPv6 adoption by end-user systems, Colitti et al. included web resources from a dual-stack host and from an IPv4-only host on the Google landing page, so that its visitors' browsers would attempt to access the dual-stack hosted resources via IPv6 first [256]. Due to possible browser or DNS incompatibilities in respect to IPv6 however, the reported numbers are lower bounds.

Plonka and Berger passively measured which and how clients connected to a large content delivery network's IPv6-capable servers and inferred patterns from it, like the stability and density of active IPv6 addresses [186]. Foremski et al. develop Entropy/IP, which is an approach that leverages machine learning to predict likely active IPv6 addresses, based on a seed set of active addresses observed in the past [257]. Murdock et al. introduced a more generic approach (6Gen) to determine potential IPv6 addresses from seed sets [184]. In both cases, addresses that are not in use might be generated, and, hence, the generated addresses are subjected to subsequent liveness verification. Unfortunately, these prior studies depend on existing and comprehensive seed sets, which are difficult to collect without the visibility that a network vantage point provides, such as a large Internet service provider or network operator. However, due to their inherent privacy concerns, these vantage points are heavily guarded and generally not accessible to third parties, such as academic researchers. In contrast, in Chapter 5, we introduced an approach to enumerate an assigned part of the IPv6 Internet that does not depend on a privileged network position. Furthermore, network vantage points can miss certain hosts. For example, for the content delivery networks hosts that do not initiate any connections to it, for example, servers, are missed. These hosts, however, are still discovered by our approach (see Section 5.5). Ultimately, the dataset that our approach collects can be readily used as input for generative algorithms, such as Entropy/IP [257] or 6Gen [184].

Czyz et al. aim to evaluate the general filtering policy applied to dual-stack servers (IPv6 and IPv4; less than 20 ports) [37]. As a source for dual-stack hosts they rely on hostnames with both A and AAAA records in the Rapid7 DNS ANY dataset [258]. Consequently, the security posture of IPv6-only hosts is not evaluated, a gap we fill in Chapter 5. As our findings confirm, their results indicate that dual-stack enabled servers have more permissive IPv6 firewall policies compared to IPv4, for example, SSH, Telnet, and SNMP are more than twice as open for IPv6-capable routers as they are for their IPv4 counterparts. However, their work exhibits limitations that our technique does not have. Specifically, due to their focus on dual-stack hosts Czyz et al. have missed IPv6-only hosts as well as systems lacking forward-zone A or AAAA records. We overcome these limitations by presenting a technique to identify active IPv6 hosts in specific networks instead of relying on network vantage points or public, possibly stale, datasets. Hence, we can survey so-far neglected IPv6-only systems, which exhibit critical security issues. Furthermore, contrary to Czyz et al., we pinpoint a possible root cause of the differences in firewall policing between IPv4 and IPv6: Stateless address autoconfiguration (SLAAC).

Fiebig et al. also utilize reverse DNS entries to obtain a view on assigned IPv6 addresses [9]. Specifically, they exploit semantic differences in the type of the response of a nameserver [259] to enumerate reverse zones. However, their work does not include a security evaluation of the identified hosts. We leverage their work as a baseline for our evaluation and we find that our technique performs better for large prefixes, due to the already high deployment rate of DNSSEC in their respective reverse zones. Furthermore, we find that the usefulness of their technique has limitations. After Fiebig et al. presented their findings in late 2016 [188], mitigation technique have been adopted by network operators. Furthermore, their technique generates a significant request volume, which can be mitigated similarly. In contrast, mitigations for our enumeration technique require significant changes to the DNSSEC standard, which we hypothesize industry is unlikely going to adopt in the near future due to deployment concerns (see Section 5.6). Furthermore, our technique is more economical in generated requests, putting less strain on networks and rendering network-based detection more difficult.

6.11 DNSSEC Privacy Issues

DNSSEC-signed zones that leverage NSEC-based denial of existence are known to be vulnerable to zone enumeration attacks [201]. Although NSEC3 renders it more difficult, as a hash-based approach, it remains possible to enumerate the zone through a brute-force attack. Goldberg et al. presented variants of NSEC3 and showed that the modified schemes would still be vulnerable to zone enumeration through brute-force attacks [203]. To break the hashed names, Wander et al. leveraged a GPU to launch a dictionary attack against the ".com" zone and successfully unblinded 64% of the zone [260]. We discussed prior work related to preventing zone-walking attacks on DNSSEC in Section 5.6, which is why we omit it here in the pursuit of brevity.

Previous work hints at the potential of information leakage through reverse DNS zones [261, 262, 263]. However, they only provide preliminary insight, and do not discuss or leverage any information leaks (e.g., resource record types and their meaning for IPv6 reverse zones) nor do they conduct any empirical study on the real-world significance of such leaks. Contrary to prior work, our approach transfers the challenge of unblinding NSEC3 into a new domain. There, we leverage various intricate details, which have not yet received any attention, to considerably reduce the effort to unblind IPv6 addresses from the NSEC3 chain. Specifically, we utilize the way reverse zones are organized, the well-defined structure of IPv6, and the insight that NSEC3 still leaks the record types, which have a specific meaning for reverse zones.

This page intentionally left blank

Chapter 7

Summary

In this dissertation, we investigated abuse on the Internet, which has become an ever-present and ever-evolving threat for users with the increasing reliance on Internet-based services in daily life, because the software and protocols implementing these services' functionality are often vulnerable to attacks. Without carefully navigating these threats, miscreants can violate the users' security, privacy, and trust by stealing and monetizing their private data, or by scamming, defrauding, or misleading them. However, to enable users to steer clear of such abuse, we need to be able to analyze and understand it first. We contributed to conquering the problem of Internet abuse from three interconnected angles: *identification*, to address occurring abuse (Chapter 2 and Chapter 3); *prevention*, to mitigate existing abuse and abuse potential (Chapter 4); and *discovery*, to reduce future abuse potential (Chapter 5).

To *identify* large-scale abuse that is occurring on the Internet, we examined two current web-based threats in depth: website defacements and malware infection campaigns.

To tackle defacements attacks, we introduced MEERKAT, which is an automated detection system that utilizes a novel approach based on the look and feel of a website to determine if the website has been defaced. It leverages stacked autoencoders to automatically learn features of how website defacements look, and it uses a feed-forward neural network with dropout to classify them accurately. We have demonstrated that Meerkat significantly outperforms state-of-the-art defacement detection systems on the largest dataset to date, spanning over 10 million defacements across 16 years. Contrary to prior work, Meerkat does not require feature engineering based on a priori domain knowledge, and, thus, it can tackle web evolution and evasions more gracefully. Since its inception in 2015, Meerkat has already inspired companies and similar systems are now being offered as online services against defacement attacks.

We then developed Delta, a light-weight system that extracts and analyzes the changes made to websites, to determine if they introduce a change in behavior, such as a website being compromised by an attacker and turning malicious. Delta leverages a novel fuzzy tree difference algorithm to extract only relevant changes, which it then clusters to distinguish stand-alone changes from groups of changes. Based on the identified groups, it establishes the scale and impact of malware infection campaigns, generates an identifying signature for easier detection, and facilitates root cause analysis by establishing commonalities across a campaign's websites. Aiding root cause analysis is crucial in defending against Internet abuse, because we must not only detect the presence of web-based malware, but we must also identify known and unknown campaigns, to understand how they are being launched. Indeed, Delta finds previously unknown web-based malware infection campaigns at scale, which we have shown by example of a campaign redirecting to the Cool Exploit Kit. Furthermore, as a result of its design, Delta can identify malicious behavior even if it is currently inactive.

In order to *prevent* existing abuse and abuse potential, we presented Cloud Strife, a new mitigation for IP address takeover attacks for TLS-based services. Although these attacks were thought to be difficult to launch and easy to detect, we discovered a new variant, called IP address use-after-free, that we confirm is practical, time-efficient, and cost-efficient for attackers to launch. This is the case because of how current online services are deployed, that is, their cloud-based ephemeral character and the cloud's underlying elasticity. Worse yet, attacks based on this new variant are imperceptible to its victim, as they are indistinguishable from normal system maintenance or service migration. Specifically, to mitigate these attacks against TLS-based services, Cloud Strife increases the security of the HTTP-based TLS certificate issuance process at the certificate authority level, by enforcing a chain of trust to the legitimate owner and preventing certificates to be issued without a strong proof of ownership. In turn, an attacker who is launching an IP address takeover attack, including but not limited to IP address use-after-free vulnerabilities, will not be able to obtain a valid certificate, which severely reduces her capabilities, and which downgrades her attack from a full impersonation attack to a simple denial-of-service attack. Finally, our mitigation is practical and can be deployed readily under strict real-world performance and usability requirements, because it has negligible operational overhead and only requires intervention in disaster-recovery scenarios.

In our fourth contribution, we introduce a novel technique to *discover* future abuse potential by enabling Internet-wide security measurements for IPv6. Specifically, our technique enumerates active IPv6 addresses from the IPv6 reverse zone. To do so, it leverages the DNSSEC zone enumeration attack, which we make practical even in the face of the zone enumeration defense NSEC3 by exploiting previously ignored subtleties and intricate details in the organization of the IPv6 reverse zone. Contrary to prior work, our technique discovers active IPv6-reachable hosts without requiring a network vantage point to observe IPv6 traffic. It also discovers hosts that previous approaches miss because they do not initiate connections that could be observed, such as those of routers or servers, and it can be directed at specific networks more easily. Based on our technique, we have then shown that IPv6 security posture is lagging behind its IPv4 counterpart for five networks of varying scale and managed by various operators, which we conjecture is the case because of unintended IPv6 connectivity, possibly through stateless address autoconfiguration (SLAAC).

In summary, in this dissertation, we made contributions to *identifying*, *preventing*, and *discovering* current Internet abuse and future abuse potential. Due the adversarial nature of Internet abuse, however, some issues remain, and future work should investigate more closely how we can address them at their core. For example, we need to better understand how IPv6 deployment impacts security posture, what the abuse potential of newly IPv6-reachable devices is, and how evolution in protocols and usage pattern impacts security.

Appendix A

Mitigating the Risks of Takeover Attacks and Domain-Validated Certificates

A.1 Takeover Attack Proof of Concept

For our proof of concept experiment (see Section 4.3.5), we obtained a valid certificate for the domain "cloudstrife.seclab.cs.ucsb.edu." The obtained certificate is shown in Listing A.2. The respective entry in the certificate transparency log can be found at `https://crt.sh/?id=250959196`. We revoked the certificate after it has propagated to certificate transparency logs, that is, shortly after issuance. In face of often ignored revocation checks, we opt not to publish the private key. Instead, we prove ownership of the certificate by signing a unique message (see Listing A.3 and Listing A.4). We did not use the certificate for any purpose besides signing the message. It can be verified as follows (lines starting with # denote comments, and lines starting with $ denote commands):

```
# Copy Listing A.2 to certificate.pem and Listing A.4 to message.txt.dgst.b64

# Create message.txt
$ echo —n "Cloud Strife: Mitigating the Security Risks of Domain Validated \
Certificates" > message.txt

# Convert the full certificate to raw PEM
$ openssl x509 —pubkey —noout —in certificate.pem > certificate_raw.pem

# Decode the signature
$ base64 —d message.txt.dgst.b64 > message.txt.dgst

# Verify the message
$ openssl dgst —sha256 —verify certificate_raw.pem —signature message.txt.dsgt \
message.txt
```

Listing A.1: Instructions to verify the signature

```
——————BEGIN CERTIFICATE——————
MIIFHzCCBAegAwIBAgISA3XAEcaykugGaCy9tCoCdJWKMA0GCSqGSIb3DQEBCwUA
MEoxCzAJBgNVBAYTAlVTMRYwFAYDVQQKEw1MZXQncyBFbmNyeXB0MSMwIQYDVQQD
ExpMZXQncyBFbmNyeXB0IEF1dGhvcml0eSBYMzAeFw0xNzExMDkyMzA4NTVaFw0x
ODAyMDcyMzA4NTVaMCkxJzAlBgNVBAMTHmNsb3Vkc3RyaWZlLnNlY2xhYi5jcy51
Y3NiLmVkdTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAONF0TzeAA6N
q5Li7e9h6+Y//d8Zy2gbWN465t3MPVlz1lSLqCZvT4e3IDjuyQ/gx+yWnndtQrhs
zHt+GigQbBcAFM5YohIVrTr7M8ozZVZhu1x11xmPZYJ9hAi8NO6p2uoZMNwiHh35
XVFQs5LFG6QpPGBWoNtu1t5zwLYF01STlMS/hNn0P/KlrnAzs2tSX//OxxaY+jos
KQCl9LrXKhOXcmaZMXFe7t8uglFsjbEvM9TRFqeENROik/TLjRlyb3BM5HtKVnno
tDh6078qCgwMzZyh5YRy2uOGHCp13TdZQtOELq0qfGNjVClwRENo+AW1K8fPnw9L
S49OpBwzx2MCAwEAAaOCAh4wggIaMA4GA1UdDwEB/wQEAwIFoDAdBgNVHSUEFjAU
BggrBgEFBQcDAQYIKwYBBQUHAwIwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQUKnFO
hVGO9fXAoSDpoRiztZhSYo4wHwYDVR0jBBgwFoAUqEpqYwR93brm0Tm3pkVl7/Oo
7KEwbwYIKwYBBQUHAQEEYzBhMC4GCCsGAQUFBzABhiJodHRwOi8vb2NzcC5pbnQt
eDMubGV0c2VuY3J5cHQub3JnMC8GCCsGAQUFBzAChiNodHRwOi8vY2VydC5pbnQt
eDMubGV0c2VuY3J5cHQub3JnLzApBgNVHREEIjAggh5jbG91ZHN0cmlmZS5zZWNs
YWIuY3MudWNzYi5lZHUwgf4GA1UdIASB9jCB8zAIBgZngQwBAgEwgeYGCysGAQQB
gt8TAQEBMIHWMCYGCCsGAQUFBwIBFhpodHRwOi8vY3BzLmxldHNlbmNyeXB0Lm9y
ZzCBqwYIKwYBBQUHAgIwgZ4MgZtUaGlzIENlcnRpZmljYXRlIG1heSBvbmx5IGJl
IHJlbGllZCB1cG9uIGJ5IFJlbHlpbmcgUGFydGllcyBhbmQgb25seSBpbiBhY2Nv
cmRhbmNlIHdpdGggdGhlIENlcnRpZmljYXRlIFBvbGljeSBmb3VuZCBhdCBodHRw
czovL2xldHNlbmNyeXB0Lm9yZy9yZXBvc2l0b3J5LzANBgkqhkiG9w0BAQsFAAOC
AQEAIj1W4ZzHlsaj6ccWccGyVahfk9JDhImMQLDUR02FYqtHLPjyM1JIIyYHP9xE
S2JZBbzMlrr2SjfxC3IQhDkUIjyPEeLv6WVT0hFbbzu3QAYjW5yigctpuggx/v7c
rhbWpmY9TJRU2QAsADF9NIeSXo+3zp15QAvrss2l+qtEK3uLgQ12+antYaI85wkc
P6MGHVV52asshcjy+v2wHxJDONmtzCHQbYXA7nhSUfspnVax8EfraGWF5XobZyLw
p91BZjOB1D+HD3ubtbk2PjlW/Eld7jgv2pCEM0iXk5suidCnG47jmZQA892iUVVf
tx4z5/ntnkiw7Gwwzm+o34fMmQ==
——————END CERTIFICATE——————
```

Listing A.2: Proof of concept certificate, signed by Let's Encrypt

```
Cloud Strife: Mitigating the Security Risks of Domain Validated Certificates
```

Listing A.3: Proof of concept message

```
Bc99Sl5FwjqYLJl/jS1gPC9fyI9XiS/ex7QVg+zIFZpJ+aPCYcsGm4fGkJxathte
w4i0p3q3lSmnkukRoRNVSvMJdfJRm5QvRQr43HsC6iT+N2xZI/QLcH0nMGUftpR2
HuEiY8LwIalNuxOOjTZJwfTTSRM+NdCjSa39RDpqQLU5LGKjBpSTT/jfg0RwrX0w
MhDnq+iqqrW0kDg08bxARWUfY7tHUAvPpiyyEhnfyThliHFkrKUjAGtH6f+6fKFe
8pZO0XJHRoMuhq4OXMjOWKJZYu7XwQXn3GDoo1bwIwykwmIpUu9wGAjlimtTY5eW
uM0tg2PkmbuZi3JaGsczuQ==
```

Listing A.4: Signature for the proof of concept message

Appendix B

Copyright

This page intentionally left blank

Bibliography

[1] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. “Delta: Automatic Identification of Unknown Web-based Infection Campaigns”. In: *Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Virgil D. Gligor and Moti Yung. Berlin, Germany: Association for Computing Machinery (ACM), Nov. 2013, pp. 109–120. ISBN: 978-1-4503-2477-9. DOI: `10.1145/2508859.2516725`.

[2] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. “Relevant Change Detection: Framework for the Precise Extraction of Modified and Novel Web-based Content as a Filtering Technique for Analysis Engines”. In: *Proceedings of the 23rd World Wide Web Conference (WWW)*. Ed. by Andrei Z. Broder, Kyuseok Shim, and Torsten Suel. WWW Companion. Developers’ Track. Seoul, Republic of Korea: International World Wide Web Conference Committee (IW3C2), Apr. 2014, pp. 595–598. ISBN: 978-1-4503-2745-9. DOI: `10.1145/2567948.2578039`.

[3] Giovanni Vigna, Kevin Borgolte, Jacopo Corbetta, Adam Doupé, Yanick Fratantonio, Luca Invernizzi, Dhilung Kirat, and Yan Shoshitaishvili. “Ten Years of iCTF: The Good, The Bad, and The Ugly”. In: *Proceedings of the 1st USENIX Summit on Gaming, Games and Gamification in Security Education (3GSE)*. Ed. by Zachary N. J. Peterson. San Diego, CA: USENIX Association, Aug. 2014. URL: `https://www.usenix.org/conference/3gse14/summit-program/presentation/vigna` (visited on 08/20/2018).

[4] Yinzhi Cao, Yan Shoshitaishvili, Kevin Borgolte, Christopher Kruegel, Giovanni Vigna, and Yan Chen. “Protecting Web Single Sign-on against Relying Party Impersonation Attacks through a Bi-directional Secure Channel with Authentication”. In: *Proceedings of the 17th International Symposium on Recent Advances in Intrusion Detection (RAID)*. Ed. by Angelos Stavrou, Herbert Bos, and Georgios Portokalidis. Vol. 8688. Lecture Notes in Computer Science (LNCS). Gothenburg, Sweden: Springer International Publishing, Sept. 2014, pp. 276–298. ISBN: 978-3-319-11379-1. DOI: `10.1007/978-3-319-11379-1_14`.

[5] Mathias Payer, Ling Huang, Neil Zhenqiang Gong, Kevin Borgolte, and Mario Frank. “What You Submit is Who You Are: A Multi-Modal Approach for Deanonymizing Scientific Publications”. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 10.1 (Jan. 2015), pp. 200–212. ISSN: 1556-6013. DOI: `10.1109/TIFS.2014.2368355`.

[6] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. "Meerkat: Detecting Website Defacements through Image-based Object Recognition". In: *Proceedings of the 24th USENIX Security Symposium (USENIX Security)*. Ed. by Jaeyeon Jung Jung and Thorsten Holz. Washington, D.C., USA: USENIX Association, Aug. 2015, pp. 595–610. ISBN: 978-1-931971-232. URL: `https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/borgolte` (visited on 08/20/2018).

[7] Shuang Hao, Kevin Borgolte, Nick Nikiforakis, Gianluca Stringhini, Manuel Egele, Michael Eubanks, Brian Krebs, and Giovanni Vigna. "Drops for Stuff: An Analysis of Reshipping Mule Scams". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Ninghui Li and Christopher Kruegel. Denver, CO, USA: Association for Computing Machinery (ACM), Oct. 2015, pp. 1081–1092. ISBN: 978-1-4503-3832-5. DOI: `10.1145/2810103.2813620`.

[8] Antonio Bianchi, Kevin Borgolte, Jacopo Corbetta, Francesco Disperati, Andrew Dutcher, John Grosen, Paul Grosen, Aravind Machiry, Christopher Salls, Yan Shoshitaishvili, Nick Stephens, Giovanni Vigna, and Ruoyu Wang. "Cyber Grand Shellphish". In: *Phrack* 15.70 (Jan. 2017). Authors listed alphabetically. URL: `http://phrack.org/papers/cyber_grand_shellphish.html` (visited on 08/20/2018).

[9] Tobias Fiebig, Kevin Borgolte, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. "Something From Nothing (There): Collecting Global IPv6 Datasets From DNS". In: *Proceedings of the 18th Passive and Active Measurement (PAM)*. Ed. by Mohamed Ali Kâafar, Steve Uhlig, and Johanna Amann. Vol. 10176. Lecture Notes in Computer Science (LNCS). Sydney, Australia: Springer International Publishing, Mar. 2017, pp. 30–43. ISBN: 978-3-319-54328-4. DOI: `10.1007/978-3-319-54328-4_3`.

[10] Kevin Borgolte, Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. "Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates". In: *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS)*. Ed. by Patrick Traynor and Alina Oprea. San Diego, CA, USA: Internet Society (ISOC), Feb. 2018. ISBN: 1891562-49-5. DOI: `10.14722/ndss.2018.23327`.

[11] Yan Shoshitaishvili, Antonio Bianchi, Kevin Borgolte, Amat Cama, Jacopo Corbetta, Francesco Disperati, Andrew Dutcher, John Grosen, Paul Grosen, Aravind Machiry, Christopher Salls, Nick Stephens, Ruoyu Wang, and Giovanni Vigna. "Mechanical Phish: Resilient Autonomous Hacking". In: *IEEE Security & Privacy* 16.2 (Mar.–Apr. 2018), pp. 12–22. ISSN: 1558-4046. DOI: `10.1109/MSP.2018.1870858`.

[12] Tobias Fiebig, Kevin Borgolte, Shuang Hao, Christopher Kruegel, Giovanni Vigna, and Anja Feldmann. "In rDNS We Trust: Revisiting a Common Data-Source's Reliability". In: *Proceedings of the 19th Passive and Active Measurement (PAM)*. Ed. by Robert Beverly and Georgios Smaragdakis. Vol. 10771. Lecture Notes in Computer Science (LNCS). Berlin, Germany: Springer International Publishing, Mar. 2018, pp. 131–145. ISBN: 978-3-319-54327-7. DOI: `10.1007/978-3-319-76481-8_10`.

[13] Kevin Borgolte, Shuang Hao, Tobias Fiebig, and Giovanni Vigna. "Enumerating Active IPv6 Hosts for Large-scale Security Scans via DNSSEC-signed Reverse Zones". In: *Proceedings of the 39th IEEE Symposium on Security & Privacy (S&P)*. Ed. by Bryan Parno and Christopher Kruegel. San Francisco, CA, USA: Institute of Electrical and Electronics Engineers (IEEE), May 2018, pp. 438–452. ISBN: 978-1-5386-4353-2. DOI: `10.1109/SP.2018.00027`.

[14] Wei Meng, Chenxiong Qian, Shuang Hao, Kevin Borgolte, Giovanni Vigna, Christopher Kruegel, and Wenke Lee. "Rampart: Protecting Web Applications from CPU-Exhaustion Denial-of-Service Attacks". In: *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*. Ed. by William Enck and Adrienne Porter Felt. Baltimore, MD, USA: USENIX Association, Aug. 2018. URL: `https://www.usenix.org/conference/usenixsecurity18/presentation/meng` (visited on 08/20/2018).

[15] Constanze Dietrich, Katharina Krombholz, Kevin Borgolte, and Tobias Fiebig. "Investigating Operators' Perspective on Security Misconfigurations". In: *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Michael Backes and XiaoFeng Wang. Toronto, ON, Canada: Association for Computing Machinery (ACM), Oct. 2018. ISBN: 978-1-4503-5693-0. DOI: `10.1145/3243734.3243794`.

[16] Giorgio Davanzo, Eric Medvet, and Alberto Bartoli. "A Comparative Study of Anomaly Detection Techniques in Web Site Defacement Detection". In: *Proceedings of the 23rd IFIP TC11 Information Security Conference & Privacy Conference*. Ed. by Sushil Jajodia, Pierangela Samarati, and Stelvio Cimato. Milan, Italy: International Federation for Information Processing (IFIP), Sept. 2008, pp. 711–716. ISBN: 978-0-387-09699-5. DOI: `10.1007/978-0-387-09699-5_50`.

[17] Kevin Borgolte. *A Brief Analysis of the ISIS/ISIL Defacement Campaign*. Nov. 6, 2014. URL: `https://kevin.borgolte.me/notes/team-system-dz-isis-isil-defacement-campaign/` (visited on 08/20/2018).

[18] Gaurav Raghuvanshi, Newley Purnell, and Jason Ng. *Malaysia Airlines Website Hacked by Group Calling Itself 'Cyber Caliphate'*. The Wall Street Journal. Jan. 26, 2015. URL: `https://www.wsj.com/articles/malaysia-airlines-website-hacked-by-group-calling-itself-cyber-caliphate-1422238358` (visited on 08/20/2018).

[19] British Broadcasting Company (BBC). *Keighley Cougars website hacked to read 'I love you Isis'*. Nov. 3, 2014. URL: `http://www.bbc.com/news/uk-england-leeds-29876394` (visited on 08/20/2018).

[20] Roberto Preatoni, Marcelo Almeida, Kevin Fernandez, and other unknown authors. *Zone-H.org - Unrestricted Information*. Jan. 1998. URL: `http://www.zone-h.org/` (visited on 08/20/2018).

[21] Eduard Kovacs. *Softpedia Interview: Alberto Redi, Head of Zone-H*. June 8, 2013. URL: `https://news.softpedia.com/news/Softpedia-Interview-Alberto-Redi-Head-of-Zone-H-359499.shtml` (visited on 08/20/2018).

[22] Radhesh Krishnan Konoth, Emanuele Vineti, Veelasha Moonsamy, Martina Lindorfer, Christopher Kruegel, Herbert Bos, and Giovanni Vigna. “MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense”. In: *Proceedings of the 25th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Michael Backes and XiaoFeng Wang. Toronto, ON, Canada: Association for Computing Machinery (ACM), Oct. 2018. ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243858.

[23] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. “The Ghost in the Browser: Analysis of Web-based Malware”. In: *Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (HotBots)*. Ed. by Niels Provos. Cambridge, MA, USA: USENIX Association, Apr. 10, 2007. URL: https://www.usenix.org/conference/hotbots-07/ghost-browser-analysis-web-based-malware (visited on 08/21/2018).

[24] Sophos Security Team. *Sophos Security Threat Report 2013*. Tech. rep. Sophos, Dec. 4, 2012. URL: https://www.sophos.com/en-us/medialibrary/PDFs/other/sophossecuritythreatreport2013.pdf (visited on 08/21/2018).

[25] Paul Baccas. *Malware injected into legitimate JavaScript code on legitimate websites*. Feb. 13, 2013. URL: https://nakedsecurity.sophos.com/2013/02/13/malware-javascript (visited on 08/21/2018).

[26] Dan Goodin. *Twitter detects and shuts down password data hack in progress*. Ars Technica. Feb. 1, 2013. URL: https://arstechnica.com/information-technology/2013/02/twitter-detects-and-shuts-down-password-data-hack-in-progress/ (visited on 08/21/2018).

[27] Facebook Security Team. *Protecting People On Facebook*. Feb. 15, 2013. URL: https://www.facebook.com/note.php?note_id=10151249208250766 (visited on 08/21/2018).

[28] Jim Finke and Joseph Menn. *Exclusive: Apple, Macs hit by hackers who targeted Facebook*. Reuters. Feb. 13, 2013. URL: https://www.reuters.com/article/us-apple-hackers/exclusive-apple-macs-hit-by-hackers-who-targeted-facebook-idUSBRE91I10920130219 (visited on 08/21/2018).

[29] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. “A Large-Scale Study of the Evolution of Web Pages”. In: *Proceedings of the 12th World Wide Web Conference (WWW)*. Ed. by Yih-Farn Robin Chen, László Kovács, and Steve Lawrence. Budapest, Hungary: International World Wide Web Conference Committee (IW3C2), May 2003, pp. 669–678. ISBN: 1-58113-680-3. DOI: 10.1145/775152.775246.

[30] Bernardo A. Huberman and Lada A. Adamic. “Evolutionary Dynamics of the World Wide Web”. Jan. 13, 1999. arXiv: cond-mat/9901071.

[31] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. "Rate of Change and other Metrics: a Live Study of the World Wide Web". In: *Proceedings of the 1st USENIX Symposium on Internet Technologies and Systems (USITS)*. Ed. by Carl Staelin. Monterey, CA, USA: USENIX Association, Dec. 1997, pp. 147–158. URL: `https://www.usenix.org/conference/usits-97/rate-change-and-other-metrics-live-study-world-wide-web` (visited on 08/21/2018).

[32] Ricardo Baeza-Yates, Carlos Castillo, and Felipe Saint-Jean. "Web Dynamics". In: ed. by Mark Levene and Alexandra Poulovassilis. Springer International Publishing, 2004. Chap. Web Dynamics, Structure, and Page Quality, pp. 93–109. ISBN: 978-3-662-10874-1. DOI: `10.1007/978-3-662-10874-1_5`.

[33] Federico Maggi, William Robertson, Christopher Kruegel, and Giovanni Vigna. "Protecting a Moving Target: Addressing Web Application Concept Drift". In: *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID)*. Ed. by Engin Kirda. Vol. 5758. Lecture Notes in Computer Science (LNCS). Saint-Malo, France: Springer International Publishing, Sept. 2009, pp. 21–40. ISBN: 978-3-642-04342-0. DOI: `10.1007/978-3-642-04342-0_2`.

[34] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. "Flip Feng Shui: Hammering a Needle in the Software Stack". In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security)*. Ed. by Thorsten Holz and Stefan Savage. Austin, TX, USA: USENIX Association, Aug. 2016, pp. 1–18. ISBN: 978-1-931971-32-4. URL: `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi` (visited on 08/22/2018).

[35] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation". In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security)*. Ed. by Thorsten Holz and Stefan Savage. Austin, TX, USA: USENIX Association, Aug. 2016, pp. 19–35. ISBN: 978-1-931971-32-4. URL: `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/xiao` (visited on 09/01/2018).

[36] *Case Study: T-Mobile US Goes IPv6-only Using 464XLAT*. Internet Society (ISOC). June 13, 2014. URL: `https://www.internetsociety.org/resources/deploy360/2014/case-study-t-mobile-us-goes-ipv6-only-using-464xlat/` (visited on 08/31/2018).

[37] Jakub Czyz, Matthew Luckie, Mark Allman, and Michael Bailey. "Don't Forget to Lock the Back Door! A Characterization of IPv6 Network Security Policy". In: *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS)*. Ed. by Srdjan Capkun. San Diego, CA, USA: Internet Society (ISOC), Feb. 2016. ISBN: 189156241X. DOI: `10.14722/ndss.2016.23047`.

[38] Malaysia Computer Emergency Response Team (MyCERT). *MyCERT Incident Statistics*. Jan. 2014. URL: `https://www.mycert.org.my/en/services/statistic/mycert/2013/main/detail/914/index.html` (visited on 08/20/2018).

[39] Malaysia Computer Emergency Response Team (MyCERT). *MyCERT 2nd Quarter 2013 Summary Report*. Tech. rep. Aug. 6, 2013. URL: `https://www.mycert.org.my/data/content_files/27/804.pdf` (visited on 08/20/2018).

[40] Steve Mansfield-Devine. "Hacktivism: assessing the damage". In: *Network Security* 2011.8 (Aug. 2011), pp. 5–13. ISSN: 1353-4858. DOI: `10.1016/S1353-4858(11)70084-8`.

[41] Mathieu Gorge. "Cyberterrorism: hype or reality?" In: *Computer Fraud & Security* 2007.2 (Feb. 2007), pp. 9–12. ISSN: 1361-3723. DOI: `10.1016/S1361-3723(07)70021-0`.

[42] Henner Kircher. "The Practice of War: Production, Reproduction and Communication of Armed Violence". In: ed. by Aparna Rao, Michael Bollig, and Monika Böck. Berghahn Books, Mar. 2011. Chap. 12. Martyrs, Victims, Friends and Foes: Internet Representations by Palestinian Islamists, pp. 285–304. ISBN: 978-0857451415.

[43] Gabriel Weimann. "Terror on the Internet: The New Arena, the New Challenges". In: United States Institute of Peace Press, Mar. 2006. Chap. 6. Fighting Back: Responses to Terrorism on the Internet, and Their Cost, pp. 197–202. ISBN: 978-1929223718.

[44] Newley Purnell. *Google Access Is Disrupted in Vietnam*. The Wall Street Journal. Feb. 23, 2015. URL: `https://www.wsj.com/articles/google-access-is-disrupted-in-vietnam-1424680458` (visited on 08/20/2018).

[45] Limbikani Makani. *100+ Zambian websites hacked & defaced: Spar, Postdotnet, SEC, Home Affairs, Ministry of Finance*. TechTrends Zambia. Apr. 15, 2014. URL: `http://www.techtrends.co.zm/zambian-websites-hacked/` (visited on 08/20/2018).

[46] British Broadcasting Company (BBC). *Angry Birds website hacked after NSA-GCHQ leaks*. Jan. 29, 2014. URL: `http://www.bbc.com/news/technology-25949341` (visited on 08/20/2018).

[47] Aarshit Mittal. *NIC Of Suriname, Antigua & Barbuda And Saint Lucia Hacked By Pakistani Hackers*. via Internet Archive Wayback Machine. Oct. 7, 2013. URL: `https://web.archive.org/web/20131013110912/http://cyber-n.com/2013/10/nic-of-suriname-antigua-barbuda-and-saint-lucia-hacked-by-pakistani-hackers.html` (visited on 08/20/2018).

[48] John Leyden. *Islamist hackers attack Danish sites*. The Register. Feb. 9, 2006. URL: `https://www.theregister.co.uk/2006/02/09/islamic_defacement_protests/` (visited on 08/20/2018).

[49] John Leyden. *Hacktivists attack UN.org*. The Register. Aug. 13, 2007. URL: `https://www.theregister.co.uk/2007/08/13/un_hack/` (visited on 08/20/2018).

[50] Giorgio Maone. *United Nations VS SQL Injections,* Aug. 12, 2007. URL: `https://hackademix.net/2007/08/12/united-nations-vs-sql-injections` (visited on 08/20/2018).

[51] Shaheem Reid. *Hip-Hop Sites Hacked By Apparent Hate Group; SOHH, AllHipHop Temporarily Suspend Access*. MTV News. June 27, 2008. URL: `http://www.mtv.com/news/1590117/hip-hop-sites-hacked-by-apparent-hate-group-sohh-allhiphop-temporarily-suspend-access/` (visited on 08/20/2018).

[52] Byron Acohido. *State Department webpages defaced*. USA Today. Oct. 23, 2013. URL: `https://www.usatoday.com/story/cybertruth/2013/10/23/department-of-state-webpages-defaced/3170277/` (visited on 08/20/2018).

[53] John Leyden. *Foxconn website defaced after iPhone assembly plant suicides*. The Register. May 26, 2010. URL: `http://www.channelregister.co.uk/2010/05/26/foxconn_defacement_suicide_protest/` (visited on 08/20/2018).

[54] John Leyden. *Anti-Israel hackers deface central bank site*. The Register. Apr. 30, 2008. URL: `https://www.theregister.co.uk/2008/04/30/bank_of_israel_hacking/` (visited on 08/20/2018).

[55] British Broadcasting Company (BBC). *Nottinghamshire Police website hacked by AnonGhost*. Nov. 7, 2014. URL: `http://www.bbc.com/news/uk-england-nottinghamshire-29951605` (visited on 08/20/2018).

[56] British Broadcasting Company (BBC). *Shropshire Fire Service website hacked by AnonGhost*. Nov. 8, 2014. URL: `http://www.bbc.com/news/uk-england-shropshire-29966897` (visited on 08/20/2018).

[57] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. "Increased DNS Forgery Resistance Through 0x20-Bit Encoding: SecURItY viA LeET QueRieS". In: *Proceedings of the 15th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Paul Syverson and Somesh Jha. Alexandria, VA, USA: Association for Computing Machinery (ACM), Oct. 2008, pp. 211–222. ISBN: 978-1-59593-810-7. DOI: `10.1145/1455770.1455798`.

[58] Giovanni Vigna and Christopher Kruegel. "Handbook of Information Security". In: ed. by Hossein Bidgoli. Vol. 3. John Wiley and Sons, Dec. 30, 2005. Chap. Host-based Intrusion Detection. ISBN: 978-0471648338.

[59] Alberto Bartoli, Giorgio Davanzo, and Eric Medvet. "The Reaction Time to Web Site Defacements". In: *IEEE Internet Computing* 13 (4 July 21, 2009), pp. 52–58. ISSN: 1941-0131. DOI: `10.1109/MIC.2009.91`.

[60] David S. Anderson, Chris Fleizach, Stefan Savage, and Geoffrey M. Voelker. "Spamscatter: Characterizing Internet Scam Hosting Infrastructure". In: *Proceedings of the 16th USENIX Security Symposium (USENIX Security)*. Ed. by Niels Provos. Boston, MA, USA: USENIX Association, Aug. 2007. URL: `https://www.usenix.org/conference/16th-usenix-security-symposium/spamscatter-characterizing-internet-scam-hosting` (visited on 08/20/2018).

[61] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*. Ed. by Aaron Courville, Rob Fergus, and Brian Kingsbury. Banff, AB, Canada: Computational & Biological Learning Society (CBLS), Apr. 2014. arXiv: `1312.6229 [cs.CV]`.

[62] Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Y. Ng. "Building High-level Features Using Large Scale Unsupervised Learning". In: *Proceedings of the 29th International Conference on Machine Learning (ICML)*. Ed. by JOhn Langford and Joelle Pineau. Edinburgh, Scotland: Omnipress, June 2012, pp. 81–88. ISBN: 978-1-4503-1285-1. arXiv: `1112.6209 [cs.LG]`.

[63] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th Conference on Neural Information Processing Systems (NIPS)*. Ed. by Léon Bottou and Chris J.C. Burges. Lake Tahoe, NV, USA: Curran Associates Inc., Dec. 2012, pp. 1097–1105. URL: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf` (visited on 08/21/2018).

[64] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Ed. by Aleix Martinez, Ronen Basri, Rene Vidal, and Cornelia Fermuller. Columbus, OH, USA: Institute of Electrical and Electronics Engineers (IEEE), June 2014, pp. 580–587. ISBN: 978-1-4799-5118-5. DOI: `10.1109/CVPR.2014.81`.

[65] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0018-9219. DOI: `10.1109/5.726791`.

[66] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. "Large-scale Deep Unsupervised Learning using Graphics Processors". In: *Proceedings of the 26th International Conference on Machine Learning (ICML)*. Ed. by Léon Bottou and Michael Littman. Montréal, QC, Canada: Association for Computing Machinery (ACM), June 2009, pp. 873–880. ISBN: 978-1-60558-516-1. DOI: `10.1145/1553374.1553486`.

[67] Quoc V. Le, Jiquan Ngiam, Zhenghao Chen, Daniel Chia, Pang Wei Koh, and Andrew Y. Ng. "Tiled convolutional neural networks". In: *Proceedings of the 23rd Conference on Neural Information Processing Systems (NIPS)*. Ed. by Richard Zemel and John Shawe-Taylor. Vancouver, BC, Canada: Curran Associates Inc., Dec. 2010, pp. 1279–1287. URL: `http://papers.nips.cc/paper/4136-tiled-convolutional-neural-networks.pdf` (visited on 08/21/2018).

[68] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. “Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations”. In: *Proceedings of the 26th International Conference on Machine Learning (ICML)*. Ed. by Léon Bottou and Michael Littman. Montréal, QC, Canada: Association for Computing Machinery (ACM), June 2009, pp. 609–616. ISBN: 978-1-60558-516-1. DOI: `10.1145/1553374.1553453`.

[69] Pierre Sermanet, Soumith Chintala, and Yann LeCun. “Convolutional Neural Networks Applied to House Numbers Digit Classification”. In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR)*. Ed. by Alberto Del Bimbo, Kim L. Boyer, and Katsushi Ikeuchi. Tsukuba, Japan: Institute of Electrical and Electronics Engineers (IEEE), Nov. 2012, pp. 3288–3291. ISBN: 978-4-9906441-0-9. arXiv: `1204.3968 [cs.CV]`.

[70] Aapo Hyvärinen, Jarmo Hurri, and Patrik O. Hoyer. *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision*. Vol. 39. Computational Imaging and Vision. Springer International Publishing, June 4, 2009. ISBN: 978-1848824904.

[71] Karo Gregor and Yann LeCun. “Emergence of complex-like cells in a temporal product network with local receptive fields”. June 2, 2010. arXiv: `1006.0448 [cs.NE]`.

[72] Kevin Jarrett, Koray Kavukcuoglu, Marc’Aurelio Ranzato, and Yann LeCun. “What is the best multi-stage architecture for object recognition?” In: *Proceedings of the 12th IEEE Conference on Computer Vision (ICCV)*. Ed. by Roberto Cipolla, Martial Hebert, Xiaoou Tang, and Naokazu Yokoya. Kyoto, Japan: Institute of Electrical and Electronics Engineers (IEEE), Sept. 2009, pp. 2146–2153. ISBN: 978-1-4244-4419-9. DOI: `10.1109/ICCV.2009.5459469`.

[73] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. “Improving neural networks by preventing co-adaptation of feature detectors”. July 3, 2012. arXiv: `1207.0580 [cs.NE]`.

[74] Yangqing Jia and Eric Shelhamer. *Caffe: An Open Source Convolutional Architecture for Fast Feature Embedding*. 2013. URL: `https://caffe.berkeleyvision.org/` (visited on 08/20/2018).

[75] Stefan Axelsson. “The Base-Rate Fallacy and the Difficulty of Intrusion Detection”. In: *ACM Transactions on Information and System Security (TISSEC)* 3 (3 Aug. 2000), pp. 186–205. ISSN: 1094-9224. DOI: `10.1145/357830.357849`.

[76] Giorgio Davanzo, Eric Medvet, and Alberto Bartoli. “Anomaly Detection Techniques for a Web Defacement Monitoring Service”. In: *Expert Systems with Applications (ESWA)* 38 (10 Sept. 15, 2011), pp. 12521–12530. ISSN: 0957-4174. DOI: `10.1016/j.eswa.2011.04.038`.

[77] Alberto Bartoli and Eric Medvet. “Automatic Integrity Checks for Remote Web Resources”. In: *IEEE Internet Computing* 10 (6 Nov. 13, 2006), pp. 56–62. ISSN: 1941-0131. DOI: `10.1109/MIC.2006.117`.

[78] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. "Adversarial Machine Learning". In: *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (AISEC)*. Ed. by Alvaro A. Cárdenas, Rachel Greenstadt, and Ben Rubinstein. Chicago, IL, USA: Association for Computing Machinery (ACM), Nov. 2011, pp. 43–58. ISBN: 978-1-4503-1003-1. DOI: 10.1145/2046684.2046692.

[79] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. "The Security of Machine Learning". In: *Machine Learning* 81 (2 Nov. 2010), pp. 121–148. ISSN: 1573-0565. DOI: 10.1007/s10994-010-5188-5.

[80] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. "Can Machine Learning Be Secure?" In: *Proceedings of the 1st ACM ASIA Symposium on Information, Computer and Communications Security (ASIACCS)*. Ed. by Shiuhpyng Shieh and Sushil Jajodia. Taipei, Taiwan: Association for Computing Machinery (ACM), Mar. 2006, pp. 16–25. ISBN: 1-59593-272-0. DOI: 10.1145/1128817.1128824.

[81] Nedim Šrndic and Pavel Laskov. "Practical Evasion of a Learning-Based Classifier: A Case Study". In: *Proceedings of the 35th IEEE Symposium on Security & Privacy (S&P)*. Ed. by Michael Backes, Adrian Perrig, and Helen Wang. San Jose, CA, USA: Institute of Electrical and Electronics Engineers (IEEE), May 2014, pp. 197–211. ISBN: 978-1-4799-4686-0. DOI: 10.1109/SP.2014.20.

[82] Daniel Lowd and Christopher Meek. "Adversarial Learning". In: *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*. Ed. by Roberto Bayardo and Kristin Bennett. Chicago, IL, USA: Association for Computing Machinery (ACM), Aug. 2005, pp. 641–647. ISBN: 1-59593-135-X. DOI: 10.1145/1081870.1081950.

[83] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. "Adversarial Classification". In: *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*. Ed. by Johannes Gehrke and William DuMochel. Seattle, WA, USA: Association for Computing Machinery (ACM), Aug. 2004, pp. 99–108. ISBN: 1-58113-888-1. DOI: 10.1145/1014052.1014066.

[84] Amir Globerson and Sam Roweis. "Nightmare at Test Time: Robust Learning by Feature Deletion". In: *Proceedings of the 23rd International Conference on Machine Learning (ICML)*. Ed. by William Cohen and Andrew Moore. Pittsburgh, PA, USA: Association for Computing Machinery (ACM), June 2006, pp. 353–360. ISBN: 1-59593-383-2. DOI: 10.1145/1143844.1143889.

[85] Han Xiao, Huang Xiao, and Claudia Eckert. "Adversarial Label Flips Attack on Support Vector Machines". In: *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*. Ed. by Luc De Raedt. Montepellier, France: IOS Press Amsterdam, Aug. 2012, pp. 870–875. ISBN: 978-1-61499-098-7. DOI: 10.3233/978-1-61499-098-7-870.

[86] David Wagner and Paolo Soto. “Mimicry Attacks on Host-based Intrusion Detection Systems”. In: *Proceedings of the 9th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Ravi Sandhu. Washington, D.C., USA: Association for Computing Machinery (ACM), Nov. 2002, pp. 255–264. ISBN: 1-58113-612-9. DOI: 10.1145/586110.586145.

[87] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. “Automating Mimicry Attacks Using Static Binary Analysis”. In: *Proceedings of the 14th USENIX Security Symposium (USENIX Security)*. Ed. by Patrick McDaniel. Baltimore, MD, USA: USENIX Association, July 2005. URL: https://www.usenix.org/conference/14th-usenix-security-symposium/automating-mimicry-attacks-using-static-binary-analysis (visited on 08/21/2018).

[88] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. “Revolver: An Automated Approach to the Detection of Evasive Web-based Malware”. In: *Proceedings of the 22nd USENIX Security Symposium (USENIX Security)*. Ed. by Sam King. Washington, D.C, USA: USENIX Association, Aug. 2013, pp. 637–652. ISBN: 978-1-931971-03-4. URL: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/kapravelos (visited on 08/21/2018).

[89] Clemens Kolbitsch, Engin Kirda, and Christopher Kruegel. “The Power of Procrastination: Detection and Mitigation of Execution-stalling Malicious Code”. In: *Proceedings of the 18th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by George Danezis and Vitaly Shmatikov. Chicago, IL, USA: Association for Computing Machinery (ACM), Oct. 2011, pp. 285–296. ISBN: 978-1-4503-0948-6. DOI: 10.1145/2046707.2046740.

[90] Marco Cova, Christopher Kruegel, and Giovanni Vigna. “Detection and analysis of drive-by-download attacks and malicious JavaScript code”. In: *Proceedings of the 19th World Wide Web Conference (WWW)*. Ed. by Juliana Freire and Soumen Chakrabarti. Raleigh, NC, USA: International World Wide Web Conference Committee (IW3C2), Apr. 2010, pp. 281–290. ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772720.

[91] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. “Prophiler: A Fast Filter for the Large-Scale Detection of Malicious Web Pages”. In: *Proceedings of the 20th World Wide Web Conference (WWW)*. Ed. by Krithi Ramamritham and Sowmyanarayanan Sadagopan. Hyderabad, India: International World Wide Web Conference Committee (IW3C2), Mar. 2011, pp. 197–206. ISBN: 978-1-4503-0632-4. DOI: 10.1145/1963405.1963436.

[92] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. “All your iFRAMEs point to Us”. In: *Proceedings of the 17th USENIX Security Symposium (USENIX Security)*. Ed. by Paul van Oorschot. San Jose, CA, USA: USENIX Association, Aug. 2008, pp. 1–15. URL: https://www.usenix.org/conference/17th-usenix-security-symposium/all-your-iframes-point-us (visited on 08/21/2018).

[93] Pierre-Marc Bureau. *Linux/Cdorked.A: New Apache backdoor being used in the wild to serve Blackhole*. ESET Security. Apr. 26, 2013. URL: `https://www.welivesecurity.com/2013/04/26/linuxcdorked-new-apache-backdoor-in-the-wild-serves-blackhole/` (visited on 08/21/2018).

[94] Daniel Cid. *Apache Binary Backdoors on Cpanel-based servers*. Sucuri Security. Apr. 26, 2013. URL: `https://blog.sucuri.net/2013/04/apache-binary-backdoors-on-cpanel-based-servers.html` (visited on 08/21/2018).

[95] *Internet Archive*. Internet Archive. URL: `https://archive.org` (visited on 09/01/2018).

[96] Sudarshan S. Chawathe and Hector Garcia-Molina. "Meaningful Change Detection in Structured Data". In: *Proceedings of the 23rd ACM SIGMOD International Conference on Management of Data (MOD)*. Ed. by Patrick Valduriez and Henry F. Korth. Tuscon, AZ, USA: Association for Computing Machinery (ACM), May 1997, pp. 26–37. ISBN: 0-89791-911-4. DOI: `10.1145/253260.253266`.

[97] Yuan Wang, David J. DeWitt, and Jin-Yi Cai. "X-Diff: An Effective Change Detection Algorithm for XML Documents". In: *Proceedings of the 19th International Conference on Data Engineering (ICDE)*. Ed. by Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman. Bangalore, India: Institute of Electrical and Electronics Engineers (IEEE), Mar. 2003, pp. 519–530. ISBN: 0-7803-7665-X. DOI: `10.1109/ICDE.2003.1260818`.

[98] Harold W. Kuhn. "The Hungarian Method for the Assignment Problem". In: *Naval Research Logistics Quarterly (NRL)* 2 (1/2 Mar. 1955), pp. 83–97. DOI: `10.1002/nav.3800020109`.

[99] Jesse Kornblum. "Identifying Almost Identical Files Using Context Triggered Piecewise Hashing". In: *Proceedings of the 6th Digital Forensic Research Workshop (DFRWS)*. Lafayette, IN, USA: Elsevier, Aug. 2006, pp. 91–97. DOI: `10.1016/j.diin.2006.06.015`.

[100] Matthew A. Jaro. "Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida". In: *Journal of the American Statistical Association (JASA)* 84.406 (1989), pp. 414–420. DOI: `10.1080/01621459.1989.10478785`.

[101] Andrei Nikolaevich Kolmogorov. "Three Approaches to the Quantitative Definition of Information". In: *International Journal of Computer Mathematics (IJCM)* 2 (1-4 1968), pp. 157–168. DOI: `10.1080/00207166808803030`.

[102] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. "You Are What You Include: Large-scale Evaluation of Remote JavaScript Inclusions". In: *Proceedings of the 19th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by George Danezis and Virgil Gligor. Raleigh, NC, USA: Association for Computing Machinery (ACM), Oct. 2012, pp. 736–747. ISBN: 978-1-4503-1651-4. DOI: `10.1145/2382196.2382274`.

[103] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. “Knowledge Discovery and Data Mining: Towards a Unifying Framework”. In: *Proceedings of the 2nd International Conference on Knowledge Discovery in Data Mining (KDD)*. Ed. by Evangelos Simoudis and Jiawei Han. Portland, OR, USA: Association for the Advancement of Artificial Intelligence (AAAI), Aug. 1996, pp. 82–88. ISBN: 978-1-57735-004-0. URL: `https://www.aaai.org/Papers/KDD/1996/KDD96-014.pdf` (visited on 08/21/2018).

[104] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, eds. *Advances in Knowledge Discovery and Data Mining*. Association for the Advancement of Artificial Intelligence (AAAI), Feb. 1, 1996. ISBN: 978-0262560979.

[105] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. “OPTICS: Ordering Points To Identify the Clustering Structure”. In: *Proceedings of the 25th ACM SIGMOD International Conference on Management of Data (MOD)*. Ed. by Christos Faloutsos and Shahram Ghandeharizadeh. Philadelphia, PA, USA: Association for Computing Machinery (ACM), May 1999, pp. 49–60. ISBN: 1-58113-084-8. DOI: `10.1145/304182.304187`.

[106] Markus Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. “OPTICS-OF: Identifying Local Outliers”. In: *Proceedings of the 3rd European Conference on Principles of Knowledge Discovery and Data Mining (PKDD)*. Ed. by Jan M. Żytkow and Jan Rauch. Vol. 1704. Lecture Notes in Computer Science (LNCS). Prague, Czech Republic: Springer International Publishing, Sept. 1999, pp. 262–270. ISBN: 978-3-540-48247-5. DOI: `10.1007/978-3-540-48247-5_28`.

[107] Luca Invernizzi, Paolo Milani Comparetti, Stefano Benvenuti, Marco Cova, Christopher Kruegel, and Giovanni Vigna. “EvilSeed: A Guided Approach to Finding Malicious Web Pages”. In: *Proceedings of the 33rd IEEE Symposium on Security & Privacy (S&P)*. Ed. by Somesh Jha and Wenke Lee. San Francisco, CA, USA: Institute of Electrical and Electronics Engineers (IEEE), May 2012, pp. 428–442. ISBN: 978-0-7695-4681-0. DOI: `10.1109/SP.2012.33`.

[108] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin Zorn. “NOZZLE: A Defense Against Heap-spraying Code Injection Attacks”. In: *Proceedings of the 18th USENIX Security Symposium (USENIX Security)*. Ed. by Fabian Monrose. Montréal, QC, Canada: USENIX Association, Aug. 2009, pp. 169–186. URL: `https://www.usenix.org/conference/usenixsecurity09/technical-sessions/presentation/nozzle-defense-against-heap-spraying` (visited on 08/21/2018).

[109] Charlie Curtsinger, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. “ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection”. In: *Proceedings of the 20th USENIX Security Symposium (USENIX Security)*. Ed. by David Wagner. San Francisco, CA, USA: USENIX Association, Aug. 2011. URL: `https://www.usenix.org/conference/usenix-security-11/zozzle-fast-and-precise-browser-javascript-malware-detection` (visited on 08/21/2018).

[110] Christian Seifert, Ian Welch, and Peter Komisarczuk. “Identification of Malicious Web Pages with Static Heuristics”. In: *Proceedings of the 2008 Australasian Telecommunication Networks and Applications Conference (ATNAC)*. Ed. by Arek Dadej and Richard Harris. Adelaide, SA, Australia: Institute of Electrical and Electronics Engineers (IEEE), Dec. 2008, pp. 91–96. ISBN: 978-1-4244-2602-7. DOI: `10.1109/ATNAC.2008.4783302`.

[111] *Discuz!* Wikipedia. URL: `https://en.wikipedia.org/wiki/Discuz!` (visited on 08/21/2018).

[112] Xu Yang. *Report on the success of the Discuz! software*. Chinese. Chinese National Radio. Aug. 24, 2010. URL: `http://china.cnr.cn/gdgg/201008/t20100824_506943622.html` (visited on 08/21/2018).

[113] Moheeb Abu Rajab, Lucas Ballard, Nav Jagpal, Panayiotis Mavrommatis, Daisuke Nojiri, Niels Provos, and Ludwig Schmidt. *Trends in Circumventing Web-Malware Detection*. Tech. rep. Google, Aug. 17, 2011. URL: `http://static.googleusercontent.com/media/research.google.com/en//archive/papers/rajab-2011a.pdf` (visited on 08/21/2018).

[114] Joshua Mason, Sam Small, Fabian Monrose, and Greg MacManus. “English Shellcode”. In: *Proceedings of the 16th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Somesh Jha and Angelos D. Keromytis. Chicago, IL, USA: Association for Computing Machinery (ACM), Nov. 2009, pp. 524–533. ISBN: 978-1-60558-894-0. DOI: `10.1145/1653662.1653725`.

[115] Michalis Polychronakis, Kostas Anagnostakis, and Evangelos Markatos. “Network-level polymorphic shellcode detection using emulation”. In: *Journal in Computer Virology (JCV)* 2 (4 Feb. 2007), pp. 257–274. ISSN: 1772-9904. DOI: `10.1007/s11416-006-0031-z`.

[116] Jaeun Choi, Gisung Kim, Tae Ghyoon Kim, and Sehun Kim. “An Efficient Filtering Method for Detecting Malicious Web Pages”. In: *Proceedings of the 10th Workshop on Information Security Applications (WISA)*. Ed. by Dong Hoon Lee and Moti Yung. Vol. 7690. Lecture Notes in Computer Science (LNCS). Jeju Island, Republic of Korea: Springer International Publishing, Aug. 2012, pp. 241–253. ISBN: 978-3-642-35416-8. DOI: `10.1007/978-3-642-35416-8_17`.

[117] Yung-Tsung Hou, Yimeng Chang, Tsuhan Chen, Chi-Sung Laih, and Chia-Mei Chen. “Malicious Web Content detection by machine learning”. In: *Expert Systems with Applications (ESWA)* 37 (1 Jan. 2010), pp. 55–60. ISSN: 0957-4174. DOI: `10.1016/j.eswa.2009.05.023`.

[118] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. “Beyond Blacklists: Learning to Detect Malicious Web sites from Suspicious URLs”. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*. Ed. by Peter Flach and Mohammed Zaki. Paris, France: Association for Computing Machinery (ACM), July 2009, pp. 1245–1254. ISBN: 978-1-60558-495-9. DOI: `10.1145/1557019.1557153`.

[119] Ingrid Lunden. *Amazon's AWS Is Now A $7.3B Business As It Passes 1M Active Enterprise Customers*. TechCrunch. Oct. 7, 2015. URL: https://techcrunch.com/2015/10/07/amazons-aws-is-now-a-7-3b-business-as-it-passes-1m-active-enterprise-customers/ (visited on 08/23/2018).

[120] Haje Jan Kamps. *Microsoft Celebrates Strong Azure Adoption at Build 2016*. TechCrunch. Mar. 31, 2016. URL: https://techcrunch.com/2016/03/31/azure-growth/ (visited on 08/23/2018).

[121] *Cisco Global Cloud Index: Forecast and Methodology*. Tech. rep. 1513879861264127. Cisco Public, Feb. 1, 2018. URL: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html (visited on 08/22/2018).

[122] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. "Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds". In: *Proceedings of the 16th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Somesh Jha and Angelos D. Keromytis. Chicago, IL, USA: Association for Computing Machinery (ACM), Nov. 2009, pp. 199–212. ISBN: 978-1-60558-894-0. DOI: 10.1145/1653662.1653687.

[123] Zineb Ait Bahajji and Gary Illyes. *HTTPS as a ranking signal*. Google Webmaster Central Blog. Google. Aug. 6, 2014. URL: https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html (visited on 08/22/2018).

[124] Kayce Basques. *Why HTTPS Matters*. Google Developers: Web Fundamentals. Google. URL: https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https (visited on 09/26/2017).

[125] Paul Venezia. *Code injection: A new low for ISPs*. InfoWorld. May 26, 2015. URL: http://www.infoworld.com/article/2925839/net-neutrality/code-injection-new-low-isps.html (visited on 08/22/2018).

[126] Eric Mill. *The Web Is Deprecating HTTP And It's Going To Be Okay*. Motherboard, VICE. May 14, 2015. URL: https://motherboard.vice.com/en_us/article/wnjyay/the-web-is-deprecating-http-and-its-going-to-be-okay (visited on 08/22/2018).

[127] Daniel Stenberg. *TLS in HTTP/2*. Mar. 6, 2015. URL: https://daniel.haxx.se/blog/2015/03/06/tls-in-http2/ (visited on 08/22/2018).

[128] Richard Barnes, Jacob Hoffman-Andrews, and James Kasten. *Automatic Certificate Management Environment (ACME)*. Internet-Draft draft-ietf-acme-acme-07. Internet Engineering Task Force (IETF), June 21, 2017. URL: https://tools.ietf.org/id/draft-ietf-acme-acme-07.txt (visited on 08/22/2018).

[129] Josh Aas. *Milestone: 100 Million Certificates Issued*. Let's Encrypt. June 28, 2017. URL: https://letsencrypt.org//2017/06/28/hundred-million-certs.html (visited on 08/23/2018).

[130] Dan Cvrcek. *Let's Encrypt in the spotlight*. June 29, 2017. URL: https://dan.enigmabridge.com/lets-encrypt-in-the-spotlight/ (visited on 08/23/2018).

[131] Deepak Kumar, Zane Ma, Zakir Durumeric, Ariana Mirian, Joshua Mason, J. Alex Halderman, and Michael Bailey. "Security Challenges in an Increasingly Tangled Web". In: *Proceedings of the 26th World Wide Web Conference (WWW)*. Ed. by Eugene Agichtein and Evgeniy Gabrilovich. Perth, Australia: International World Wide Web Conference Committee (IW3C2), Apr. 2017, pp. 677–684. ISBN: 978-1-4503-4913-0. DOI: 10.1145/3038912.3052686.

[132] Paul Mockapetris. *Domain Names - Implementation and Specification*. Tech. rep. 1035. RFC Editor, Nov. 1987. 55 pp. DOI: 10.17487/rfc1035.

[133] Samuel Weiler and Johan Ihren. *Minimally Covering NSEC Records and DNSSEC On-line Signing*. Tech. rep. 4470. RFC Editor, Oct. 2015. 8 pp. DOI: 10.17487/rfc4470.

[134] Roy Arenda, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. *DNS Security Introduction and Requirements*. Tech. rep. 4033. RFC Editor, Mar. 2005. 21 pp. DOI: 10.17487/rfc4033.

[135] Olaf M. Kolkman, W. (Matthijs) Mekking, and R. (Miek) Gieben. *DNSSEC Operational Practices, Version 2*. Tech. rep. 6781. RFC Editor, Dec. 2012. 71 pp. DOI: 10.17487/rfc6781.

[136] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. "A Longitudinal, End-to-End View of the DNSSEC Ecosystem". In: *Proceedings of the 26th USENIX Security Symposium (USENIX Security)*. Ed. by Engin Kirda and Thomas Ristenpart. Vancouver, BC, Canada: USENIX Association, Aug. 2017, pp. 1307–1322. ISBN: 978-1-931971-40-9. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/chung (visited on 08/22/2018).

[137] Peter Mell and Tim Grance. *The NIST Definition of Cloud Computing*. Tech. rep. SP 800-145. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg (NIST), Sept. 2011. DOI: 10.6028/NIST.SP.800-145.

[138] CAcert. *Welcome to CAcert*. URL: http://www.cacert.org/ (visited on 08/22/2018).

[139] J. R. Prins and Fox-IT Cybercrime. *Interim Report: DigiNotar Certificate Authority Breach "Operation Black Tulip"*. Tech. rep. Fox-IT BV, Sept. 5, 2011. URL: https://www.rijksoverheid.nl/binaries/rijksoverheid/documenten/rapporten/2011/09/05/diginotar-public-report-version-1/rapport-fox-it-operation-black-tulip-v1-0.pdf (visited on 08/22/2018).

[140] Bill Budington. *Symantec Issues Rogue EV Certificate for Google.com*. Electronic Frontier Foundation (EFF) Blog. Sept. 21, 2019. URL: https://www.eff.org/deeplinks/2015/09/symantec-issues-rogue-ev-certificate-googlecom (visited on 08/22/2018).

[141] Ben Laurie. "Certificate Transparency". In: *Queue* 12.8 (Aug. 2014), 10:10–10:19. ISSN: 1542-7730. DOI: 10.1145/2668152.2668154.

[142] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. Tech. rep. 6962. RFC Editor, June 2013. 27 pp. DOI: 10.17487/rfc6962.

[143] John Aas. *Why ninety-day lifetimes for certificates?* Let's Encrypt. Nov. 9, 2015. URL: https://letsencrypt.org/2015/11/09/why-90-days.html (visited on 08/22/2018).

[144] Maarten Aertsen, Maciej Korczyński, Giovane C. M. Moura, Samaneh Tajalizadehkhoob, and Jan van den Berg. "No domain left behind: is Let's Encrypt democratizing encryption?" In: *Proceedings of the 2017 Applied Networking Research Workshop (ANRW)*. Ed. by Jörg Ott and Renata Cruz Teixeira. Prague, Czech Republic: Association for Computing Machinery (ACM), July 15, 2017, pp. 48–54. ISBN: 978-1-4503-5108-9. DOI: 10.1145/3106328.3106338.

[145] Matthias Neugschwandtner, Martina Lindorfer, and Christian Platzer. "A View To A Kill: WebView Exploitation". In: *Proceedings of the 6st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*. Ed. by Vern Paxson. Washington, D.C., USA: USENIX Association, Aug. 2013. URL: https://www.usenix.org/conference/leet13/workshop-program/presentation/Neugschwandtner (visited on 08/22/2018).

[146] Tongbo Luo, Hao Hao, Wenliang Du, Yifei Wang, and Heng Yin. "Attacks on WebView in the Android System". In: *Proceedings of the 27th ACM ASIA Conference on Information, Computer and Communications Security (ASIACCS)*. Ed. by James McDermott and Michael Locasto. Orlando, FL, USA: Applied Computer Security Associates (ACSA), Dec. 2011, pp. 343–352. ISBN: 978-1-4503-0672-0. DOI: 10.1145/2076732.2076781.

[147] Matthew Bryant. *The .io Error – Taking Control of All .io Domains With a Targeted Registration*. July 10, 2017. URL: https://thehackerblog.com/the-io-error-taking-control-of-all-io-domains-with-a-targeted-registration/ (visited on 08/22/2018).

[148] Brian Krebs. *FBI: $2.3 Billion Lost to CEO Email Scams*. Apr. 7, 2016. URL: https://krebsonsecurity.com/2016/04/fbi-2-3-billion-lost-to-ceo-email-scams/ (visited on 08/22/2018).

[149] Arne Swinnen. *Authentication Bypass on Uber's Single Sign-On via Subdomain Takeover*. June 25, 2017. URL: https://www.arneswinnen.net/2017/06/authentication-bypass-on-ubers-sso-via-subdomain-takeover/ (visited on 08/22/2018).

[150] John Aas. *Wildcard Certificates Coming January 2018*. Let's Encrypt. July 6, 2017. URL: https://letsencrypt.org//2017/07/06/wildcard-certificates-coming-jan-2018.html (visited on 08/22/2018).

[151] Jeffrey Pang, Aditya Akella, Anees Shaikh, Balachander Krishnamurthy, and Srinivasan Seshan. "On the Responsiveness of DNS-based Network Control". In: *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC)*. Ed. by Jim Kurose. Taormina, Sicily, Italy: Association for Computing Machinery (ACM), Oct. 2004, pp. 21–26. ISBN: 1-58113-821-0. DOI: `10.1145/1028788.1028792`.

[152] Cameron Coles. *AWS vs Azure vs Google Cloud Market Share 2017*. URL: `https://www.skyhighnetworks.com/cloud-security-blog/microsoft-azure-closes-iaas-adoption-gap-with-amazon-aws/` (visited on 09/15/2017).

[153] Amazon Web Services, Inc. *Throttle API Requests for Better Throughput*. URL: `http://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-request-throttling.html` (visited on 08/08/2017).

[154] Farsight Inc. *Farsight - Security Information Exchange (SIE)*. URL: `https://www.farsightsecurity.com/solutions/security-information-exchange/` (visited on 08/22/2018).

[155] Mark Allman and Vern Paxson. "Issues and Etiquette Concerning Use of Shared Measurement Data". In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC)*. Ed. by Constantine Dovrolis and Matthew Roughan. San Diego, CA, USA: Association for Computing Machinery (ACM), Oct. 2007, pp. 135–140. ISBN: 978-1-59593-908-1. DOI: `10.1145/1298306.1298327`.

[156] Network Sorcery Inc. *Well known SCTP, TCP and UDP ports*. URL: `http://www.networksorcery.com/enp/protocol/ip/ports00000.htm` (visited on 08/22/2018).

[157] Artyom Gavrichenkov. "Breaking HTTPS with BGP Hijacking". In: *Proceedings of the 2015 Black Hat Conference*. Las Vegas, NV, USA: UBM Technology Group, Aug. 2015. URL: `http://www.blackhat.com/docs/us-15/materials/us-15-Gavrichenkov-Breaking-HTTPS-With-BGP-Hijacking-wp.pdf` (visited on 08/22/2018).

[158] Scott Helme. *Revocation is broken*. July 3, 2017. URL: `https://scotthelme.co.uk/revocation-is-broken/` (visited on 08/22/2018).

[159] Adam Langley. *No, don't enable revocation checking*. Apr. 19, 2014. URL: `https://www.imperialviolet.org/2014/04/19/revchecking.html` (visited on 08/22/2018).

[160] James Larisch, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. "CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers". In: *Proceedings of the 38th IEEE Symposium on Security & Privacy (S&P)*. Ed. by Úlfar Erlingsson and Bryan Parno. San Jose, CA, USA: Institute of Electrical and Electronics Engineers (IEEE), May 2017, pp. 539–556. ISBN: 978-1-5090-5533-3. DOI: `10.1109/SP.2017.17`.

[161] The Chromium Projects. *Chromium Security: CRLSets*. URL: `https://dev.chromium.org/Home/chromium-security/crlsets` (visited on 08/22/2018).

[162] Mark Goodwin. *Revoking Intermediate Certificates: Introducing OneCRL*. Mozilla Security Blog. Mozilla. Mar. 3, 2015. URL: https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/ (visited on 08/22/2018).

[163] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. "The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software". In: *Proceedings of the 19th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by George Danezis and Virgil Gligor. Raleigh, NC, USA: Association for Computing Machinery (ACM), Oct. 2012, pp. 38–49. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382204.

[164] Phillip Hallam-Baker and Rob Stradling. *DNS Certification Authority Authorization (CAA) Resource Record*. Tech. rep. 6844. RFC Editor, Jan. 2013. 18 pp. DOI: 10.17487/rfc6844.

[165] Paul Hoffman and Jakob Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. Tech. rep. 6698. RFC Editor, Aug. 2012. 37 pp. DOI: 10.17487/rfc6698.

[166] PowerDNS. *PowerDNS Online Signing*. URL: https://doc.powerdns.com/md/authoritative/dnssec/#online-signing (visited on 08/22/2018).

[167] Google Chrome Team. *Certificate Transparency in Chrome*. Jan. 31, 2018. URL: https://github.com/GoogleChrome/ct-policy/blob/master/ct_policy.md (visited on 08/22/2018).

[168] Kirk Hall. *[cabfpub] Results on Ballot 187 - Make CAA Checking Mandatory*. Mar. 8, 2017. URL: https://cabforum.org/pipermail/public/2017-March/009988.html (visited on 08/22/2018).

[169] Tobias Fiebig, Franziska Lichtblau, Florian Streibelt, Thorben Krueger, Pieter Lexis, Randy Bush, and Anja Feldmann. "SoK: An Analysis of Protocol Design: Avoiding Traps for Implementation and Deployment". Oct. 18, 2016. arXiv: 1610.05531 [cs.CR].

[170] Tatu Ylonen. "SSH - Secure Login Connections Over the Internet". In: *Proceedings of the 6th USENIX Security Symposium (USENIX Security)*. Ed. by Greg Rose. San Jose, CA, USA: USENIX Association, July 1996, pp. 37–42. URL: https://www.usenix.org/conference/6th-usenix-security-symposium/presentation/ssh-secure-login-connections-over-internet (visited on 08/22/2018).

[171] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, et al. "The Matter of Heartbleed". In: *Proceedings of the 2014 Internet Measurement Conference (IMC)*. Ed. by Aditya Akella and Nina Taft. Vancouver, BC, Canada: Association for Computing Machinery (ACM), Nov. 2014, pp. 475–488. ISBN: 978-1-4503-3213-2. DOI: 10.1145/2663716.2663755.

[172] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. “Analysis of the HTTPS Certificate Ecosystem”. In: *Proceedings of the 2013 Internet Measurement Conference (IMC)*. Ed. by Krishna Gummadi and Craig Partidge. Barcelona, Spain: Association for Computing Machinery (ACM), Oct. 2013, pp. 291–304. ISBN: 978-1-4503-1953-9. DOI: 10.1145/2504730.2504755.

[173] Frank Cangialosi, Taejoong Chung, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. “Measurement and Analysis of Private Key Sharing in the HTTPS Ecosystem”. In: *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Christopher Kruegel. Vienna, Austria: Association for Computing Machinery (ACM), Oct. 2016, pp. 628–640. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2976749.2978301.

[174] Tobias Fiebig, Anja Feldmann, and Matthias Petschick. “A One-Year Perspective on Exposed In-memory Key-Value Stores”. In: *Proceedings of the 2016 ACM Workshop on Automated Decision Making for Active Cyber Defense (SafeConfig)*. Ed. by Nicholas J. Multari, Anoop Singhal, and David O. Manz. Vienna, Austria: Association for Computing Machinery (ACM), Oct. 24, 2016, pp. 17–22. ISBN: 978-1-4503-4566-8. DOI: 10.1145/2994475.2994480.

[175] Christian Rossow. “Amplification Hell: Revisiting Network Protocols for DDoS Abuse”. In: *Proceedings of the 21st Network and Distributed System Security Symposium (NDSS)*. Ed. by Lujo Bauer. San Diego, CA, USA: Internet Society (ISOC), Feb. 2014. ISBN: 1-891562-35-5. DOI: 10.14722/ndss.2014.23233.

[176] Jakub Czyz, Michael Kallitsis, Manaf Gharaibeh, Christos Papadopoulos, Michael Bailey, and Manish Karir. “Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks”. In: *Proceedings of the 2014 Internet Measurement Conference (IMC)*. Ed. by Aditya Akella and Nina Taft. Vancouver, BC, Canada: Association for Computing Machinery (ACM), Nov. 2014, pp. 435–448. ISBN: 978-1-4503-3213-2. DOI: 10.1145/2663716.2663717.

[177] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. “ZMap: Fast Internet-wide Scanning and Its Security Applications”. In: *Proceedings of the 22nd USENIX Security Symposium (USENIX Security)*. Ed. by Sam King. Washington, D.C, USA: USENIX Association, Aug. 2013, pp. 605–619. ISBN: 978-1-931971-03-4. URL: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric (visited on 08/21/2018).

[178] John Curran. *ARIN IPv4 Free Pool Reaches Zero*. American Registry for Internet Numbers (ARIN). Sept. 24, 2015. URL: https://www.arin.net/vault/announcements/2015/20150924.html (visited on 08/23/2018).

[179] Stephen E. Deering and Robert M. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. Tech. rep. 2460. RFC Editor, Dec. 1998. 39 pp. DOI: 10.17487/rfc2460.

[180] Stephen E. Deering and Robert M. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. Tech. rep. 8200. RFC Editor, July 2017. 42 pp. DOI: 10.17487/rfc8200.

[181] Google Inc. *IPv6 - Google*. URL: `https://www.google.com/intl/en/ipv6/statistics.html` (visited on 08/23/2018).

[182] Internet Society (ISOC). *World IPv6 Launch*. URL: `http://www.worldipv6launch.org/` (visited on 08/23/2018).

[183] Kiran K. Chittimaneni, Merike Kaeo, and Eric Vyncke. *Operational Security Considerations for IPv6 Networks*. Internet-Draft draft-ietf-opsec-v6-09. Internet Engineering Task Force (IETF), July 7, 2016. URL: `https://tools.ietf.org/id/draft-ietf-opsec-v6-09.txt` (visited on 08/23/2018).

[184] Austin Murdock, Frank Li, Paul Bramsen, Zakir Durumeric, and Vern Paxson. "Target Generation for Internet-wide IPv6 Scanning". In: *Proceedings of the 2017 Internet Measurement Conference (IMC)*. Ed. by Steve Uhlig and Olaf Maennel. London, United Kingdom: Association for Computing Machinery (ACM), Nov. 2017, pp. 242–253. ISBN: 978-1-4503-5118-8. DOI: `10.1145/3131365.3131405`.

[185] Jakub Czyz, Mark Allman, Jing Zhang, Scott Iekel-Johnson, Eric Osterweil, and Michael Bailey. "Measuring IPv6 Adoption". In: *Proceedings of the 2014 ACM SIGCOMM Conference (SIGCOMM)*. Ed. by Fabián E. Bustamante, Y. Charlie Hu, Arvind Krishnamurthy, and Sylvia Ratnasamy. Chicago, IL, USA: Association for Computing Machinery (ACM), Aug. 2014, pp. 87–98. ISBN: 978-1-4503-2836-4. DOI: `10.1145/2619239.2626295`.

[186] David Plonka and Arthur Berger. "Temporal and Spatial Classification of Active IPv6 Addresses". In: *Proceedings of the 2015 Internet Measurement Conference (IMC)*. Ed. by Vivek Pai and Neil Spring. Tokyo, Japan: Association for Computing Machinery (ACM), Oct. 2015, pp. 509–522. ISBN: 978-1-4503-3848-6. DOI: `10.1145/2815675.2815678`.

[187] Marek Majkowski and Ólafur Guðmundsson. *Deprecating the DNS ANY meta-query type*. Cloudflare Inc. Mar. 6, 2015. URL: `https://blog.cloudflare.com/deprecating-dns-any-meta-query-type/` (visited on 08/23/2018).

[188] Tobias Fiebig, Kevin Borgolte, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. "You can -j REJECT but you can not hide: Global scanning of the IPv6 Internet". In: *Proceedings of the 33rd Chaos Computer Congress (CCC)*. Hamburg, Germany: Chaos Computer Club, Dec. 2016. URL: `https://media.ccc.de/v/33c3-8061-you_can_-j_reject_but_you_can_not_hide_global_scanning_of_the_ipv6_internet` (visited on 08/23/2018).

[189] Internet Society (ISOC). *The Future is Forever – World IPv6 Launch: Measurements*. URL: `http://www.worldipv6launch.org/measurements/` (visited on 08/23/2018).

[190] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. "Increased DNS Forgery Resistance Through 0x20-bit Encoding: SecURItY viA LeER QueRieS". In: *Proceedings of the 15th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Paul Syverson and Somesh Jha. Alexandria, VA, USA: Association for Com-

puting Machinery (ACM), Oct. 2008, pp. 211–222. ISBN: 978-1-59593-810-7. DOI: `10.1145/1455770.1455798`.

[191] David Dagon, Manos Antonakakis, Kevin Day, Xiapu Luo, Christopher P. Lee, and Wenke Lee. "Recursive DNS Architectures and Vulnerability Implications". In: *Proceedings of the 16th Network and Distributed System Security Symposium (NDSS)*. Ed. by Giovanni Vigna. San Diego, CA, USA: Internet Society (ISOC), Feb. 2009. URL: `http://www.ndss-symposium.org/ndss2009/recursive-dns-architectures-and-vulnerability-implications/` (visited on 08/23/2018).

[192] Bill Adler. *Who Stole My Web Browser?* Oct. 13, 2009. URL: `https://infiniteedge.blogspot.com/2009/10/who-stole-my-web-browser.html` (visited on 08/23/2018).

[193] Cade Metz. *Comcast trials DNS hijacker*. The Register. July 28, 2009. URL: `http://www.theregister.co.uk/2009/07/28/comcast_dns_hijacker/` (visited on 08/23/2018).

[194] David Barr. *Common DNS Operational and Configuration Errors*. Tech. rep. 1912. RFC Editor, Mar. 2013. 16 pp. DOI: `10.17487/rfc1912`.

[195] John C. Klensin. *Simple Mail Transfer Protocol*. Tech. rep. 2821. RFC Editor, Mar. 2013. 79 pp. DOI: `10.17487/rfc2821`.

[196] Yakov Rekhter, Bernie Volz, and Mark Stapp. *The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option*. Tech. rep. 4702. RFC Editor, Mar. 2013. 17 pp. DOI: `10.17487/rfc4702`.

[197] Matt Crawford and Brian Haberman. *IPv6 Node Information Queries*. Tech. rep. 4620. RFC Editor, Oct. 2015. 14 pp. DOI: `10.17487/rfc4620`.

[198] Terry Manderson. *IP6.ARPA DNSSEC Report*. Internet Corporation for Assigned Names and Numbers (ICANN). URL: `http://stats.research.icann.org/dns/ip6_report/` (visited on 02/23/2017).

[199] Internet Corporation for Assigned Names and Numbers (ICANN). *DNSSEC Deployment Report*. URL: `http://dnssec-deployment.icann.org/dctld/` (visited on 11/01/2017).

[200] Ramaswamy Chandramouli and Scott Rose. *Secure Domain Name System (DNS) Deployment Guide*. Tech. rep. SP 800-81-2. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg (NIST), Sept. 2013. DOI: `10.6028/NIST.SP.800-81-2`.

[201] Ben Laurie, Geoff Sisson, and Roy Arends. *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*. Tech. rep. 5155. RFC Editor, Mar. 2008. 52 pp. DOI: `10.17487/rfc5155`.

[202] Joe Abley and Terry Manderson. *Nameservers for IPv4 and IPv6 Reverse Zones*. Tech. rep. 5855. RFC Editor, Oct. 2015. 12 pp. DOI: `10.17487/rfc5855`.

[203] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. “Stretching NSEC3 to the Limit: Efficient Zone Enumeration Attacks on NSEC3 Variants”. Feb. 8, 2015. URL: `https://www.cs.bu.edu/~goldbe/papers/nsec3attacks.pdf` (visited on 08/23/2018). .

[204] Daniel J. Bernstein. *The nsec3walker tool*. Jan. 22, 2017. URL: `https://dnscurve.org/nsec3walker.html` (visited on 08/23/2018).

[205] Paul Mockapetris. *Domain names - concepts and facilities*. Tech. rep. 1034. RFC Editor, Nov. 1987. 55 pp. DOI: `10.17487/rfc1034`.

[206] Institute of Electrical and Electronics Engineers (IEEE). *IEEE Organizationally Unique Identifier*. URL: `http://standards-oui.ieee.org/oui.txt` (visited on 01/13/2018).

[207] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. “Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices”. In: *Proceedings of the 21st USENIX Security Symposium (USENIX Security)*. Ed. by Tadayoshi Kohno. Bellevue, WA, USA: USENIX Association, Aug. 2012, pp. 205–220. ISBN: 978-1-931971-95-9. URL: `https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger` (visited on 08/23/2018).

[208] Réseaux IP Européens (RIPE). *Routing Information Service (RIS)*. URL: `https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris` (visited on 08/23/2018).

[209] University of Oregon. *University of Oregon Route Views Archive Project*. URL: `http://routeviews.org` (visited on 08/23/2018).

[210] Beichuan Zhang, Raymond Liu, Daniel Massey, and Lixia Zhang. “Collecting the Internet AS-level Topology”. In: *ACM SIGCOMM Computer Communication Review* 35.1 (Jan. 2005), pp. 53–61. ISSN: 0146-4833. DOI: `10.1145/1052812.1052825`.

[211] American Registry for Internet Numbers (ARIN). *ARIN Number Resource Policy Manual*. July 13, 2016. URL: `https://www.arin.net/vault/policy/archive/nrpm_20160713.pdf` (visited on 08/23/2018).

[212] Geoff Huston and Dr. Thomas Narten. *IPv6 Address Assignment to End Sites*. Tech. rep. 6177. RFC Editor, Mar. 2011. 9 pp. DOI: `10.17487/rfc6177`.

[213] Level 3 Threat Research Labs. *A New DDoS Reflection Attack: Portmapper; An Early Warning to the Industry*. Aug. 17, 2015. URL: `http://blog.level3.com/security/a-new-ddos-reflection-attack-portmapper-an-early-warning-to-the-industry/` (visited on 08/11/2016).

[214] Kostya Kortchinsky and Joel Land. *CVE-2016-2342: Quagga bgpd with BGP peers enabled for VPnv4 contains a buffer overflow vulnerability*. Mar. 10, 2016. URL: `http://www.kb.cert.org/vuls/id/270232` (visited on 08/23/2018).

[215] *CVE-2016-4049: Denial of Service Vulnerability in Quagga BGP Routing Daemon (bgpd)*. Mar. 10, 2016. URL: `https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-4049` (visited on 08/23/2018).

[216] Olaf M. Kolkman and R. (Miek) Gieben. *DNSSEC Operational Practices*. Tech. rep. 4641. RFC Editor, Mar. 2013. 35 pp. DOI: `10.17487/rfc4641`.

[217] R. (Miek) Gieben and W. (Matthijs) Mekking. *DNS Security (DNSSEC) Authenticated Denial of Existence*. Internet-Draft draft-gieben-nsec4-01. Internet Engineering Task Force (IETF), July 4, 2012. URL: `https://tools.ietf.org/id/draft-gieben-nsec4-01.txt` (visited on 08/23/2018).

[218] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Sachin Vasant, and Asaf Ziv. "NSEC5: Provably Preventing DNSSEC Zone Enumeration". In: *Proceedings of the 22nd Network and Distributed System Security Symposium (NDSS)*. Ed. by Engin Kirda. San Diego, CA, USA: Internet Society (ISOC), Feb. 2015. ISBN: 189156238X. DOI: `10.14722/ndss.2015.23211`.

[219] Jan Vcelak, Sharon Goldberg, Dimitrios Papadopoulos, Shumon Huque, and David C. Lawrence. *NSEC5, DNSSEC Authenticated Denial of Existence*. Internet-Draft draft-vcelak-nsec5-05. Internet Engineering Task Force (IETF), July 3, 2017. URL: `https://tools.ietf.org/id/draft-vcelak-nsec5-05.txt` (visited on 08/23/2018).

[220] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. "NSEC5: Provably Preventing DNSSEC Zone Enumeration". July 25, 2014. IACR Cryptology ePrint: `2014/582`..

[221] Eric Medvet, Cyril Fillon, and Alberto Bartoli. "Detection of Web Defacements by Means of Genetic Programming". In: *Proceedings of the 3rd International Symposium on Information Assurance and Security (IAS)*. Ed. by Qi Shi and Johnson Thomas. Manchester, United Kingdom: Institute of Electrical and Electronics Engineers (IEEE), Aug. 2007, pp. 227–234. ISBN: 978-0-7695-2876-2. DOI: `10.1109/IAS.2007.13`.

[222] Gene H. Kim and Eugene H. Spafford. "The Design and Implementation of Tripwire: A File System Integrity Checker". In: *Proceedings of the 2nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Ravi Ganesan and Ravi Sandhu. Fairfax, VA, USA: Association for Computing Machinery (ACM), Nov. 1994, pp. 18–29. ISBN: 0-89791-732-4. DOI: `10.1145/191177.191183`.

[223] Adam G. Pennington, John D. Strunk, John Linwood Griffin, Craig A. N. Soules, Garth R. Goodson, and Gregory R. Ganger. "Storage-based Intrusion Detection: Watching Storage Activity for Suspicious Behavior". In: *Proceedings of the 12th USENIX Security Symposium (USENIX Security)*. Ed. by Vern Paxson. Washington, D.C., USA: USENIX Association, Aug. 2003. URL: `https://www.usenix.org/conference/12th-usenix-security-symposium/storage-based-intrusion-detection-watching-storage` (visited on 08/21/2018).

[224] Eric Medvet, Engin Kirda, and Christopher Kruegel. “Visual-Similarity-Based Phishing Detection”. In: *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm)*. Istanbul, Turkey: Association for Computing Machinery (ACM), Sept. 2008, 22:1–22:6. ISBN: 978-1-60558-241-2. DOI: 10.1145/1460877.1460905.

[225] Wenyin Liu, Xiaotie Deng, Guanglin Huang, and Anthony Y. Fu. “An Antiphishing Strategy Based on Visual Similarity Assessment”. In: *IEEE Internet Computing* 10 (2 Mar. 2006), pp. 58–65. ISSN: 1941-0131. DOI: 10.1109/MIC.2006.23.

[226] Antonio Nappa, M.Zubair Rafique, and Juan Caballero. “Driving in the Cloud: An Analysis of Drive-by Download Operations and Abuse Reporting”. In: *Proceedings of the 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. Ed. by Konrad Rieck. Vol. 7967. Lecture Notes in Computer Science (LNCS). Berlin, Germany: Springer International Publishing, July 2013, pp. 1–20. ISBN: 978-3-642-39235-1. DOI: 10.1007/978-3-642-39235-1_1.

[227] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. “Manufacturing Compromise: The Emergence of Exploit-as-a-Service”. In: *Proceedings of the 19th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by George Danezis and Virgil Gligor. Raleigh, NC, USA: Association for Computing Machinery (ACM), Oct. 2012, pp. 821–832. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382283.

[228] Birhanu Eshete, Adolfo Villafiorita, and Komminist Weldemariam. “Malicious Website Detection: Effectiveness and Efficiency Issues”. In: *Proceedings of the 1st SysSec Workshop*. Ed. by Evanglos Markatos and Stefano Zanero. Amsterdam, Netherlands: Institute of Electrical and Electronics Engineers (IEEE), July 6, 2011, pp. 123–126. ISBN: 978-1-4577-1528-0. DOI: 10.1109/SysSec.2011.9.

[229] Alexander Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. “A Crawler-based Study of Spyware in the Web”. In: *Proceedings of the 13th Network and Distributed System Security Symposium (NDSS)*. Ed. by William Arbaugh and Dan Simon. San Diego, CA, USA: Internet Society (ISOC), Feb. 2006. ISBN: 1-891562-22-3. URL: http://www.isoc.org/isoc/conferences/ndss/06/proceedings/papers/spycrawler.pdf (visited on 08/21/2018).

[230] Jon Anthony Bell and Edward Glen Britton. “Host Identity Takeover Using Virtual Internet Protocol (IP) Addressing”. US Patent 5917997 (United States of America). June 29, 1999. Granted.

[231] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A. L. Narasimha Reddy, and Supranamaya Ranjan. "Detecting Algorithmically Generated Domain-Flux Attacks with DNS Traffic Analysis". In: *IEEE/ACM Transactions on Networking (TNET)* 20 (5 Feb. 10, 2012), pp. 1663–1677. ISSN: 1558-2566. DOI: `10.1109/TNET.2012.2184552`.

[232] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. "Your Botnet is My Botnet: Analysis of a Botnet Takeover". In: *Proceedings of the 16th ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Somesh Jha and Angelos D. Keromytis. Chicago, IL, USA: Association for Computing Machinery (ACM), Nov. 2009, pp. 635–647. ISBN: 978-1-60558-894-0. DOI: `10.1145/1653662.1653738`.

[233] Daiping Liu, Shuai Hao, and Haining Wang. "All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records". In: *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. Ed. by Christopher Kruegel. Vienna, Austria: Association for Computing Machinery (ACM), Oct. 2016, pp. 1414–1425. ISBN: 978-1-4503-3832-5. DOI: `10.1145/2976749.2978387`.

[234] Nick Nikiforakis, Steven Van Acker, Wannes Meert, Lieven Desmet, Frank Piessens, and Wouter Joosen. "Bitsquatting: Exploiting Bit-flips for Fun, or Profit?" In: *Proceedings of the 22nd World Wide Web Conference (WWW)*. Ed. by Ricardo Baeza-Yates and Sue Moon. Rio de Janeiro, Brazil: International World Wide Web Conference Committee (IW3C2), May 2010, pp. 989–998. ISBN: 978-1-4503-2035-1. DOI: `10.1145/2488388.2488474`.

[235] Yi-Min Wang, Doug Beck, Jeffrey Wang, Chad Verbowski, and Brad Daniels. "Strider Typo-Patrol: Discovery and Analysis of Systematic Typo-Squatting". In: *Proceedings of the 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*. Ed. by Steven M. Bellovin. San Jose, CA, USA: USENIX Association, July 2006, pp. 31–36. URL: `https://www.usenix.org/conference/sruti-06/strider-typo-patrol-discovery-and-analysis-systematic-typo-squatting` (visited on 08/22/2018).

[236] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich. "The Long "Taile" of Typosquatting Domain Names". In: *Proceedings of the 23rd USENIX Security Symposium (USENIX Security)*. Ed. by Kevin Fu and Jaeyeon Jung. San Diego, CA, USA: USENIX Association, Aug. 2014, pp. 191–206. ISBN: 978-1-931971-15-7. URL: `https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/szurdi` (visited on 08/22/2018).

[237] Mohammad Taha Khan, Xiang Huo, Zhou Li, and Chris Kanich. "Every Second Counts: Quantifying the Negative Externalities of Cybercrime via Typosquatting". In: *Proceedings of the 36th IEEE Symposium on Security & Privacy (S&P)*. Ed. by Vitaly Shmatikov and Lujo Bauer. San Jose, CA, USA: Institute of Electrical and Electronics Engineers (IEEE), May 2015, pp. 135–150. ISBN: 978-1-4673-6949-7. DOI: `10.1109/SP.2015.16`.

[238] Bojan Zdrnja, Nevil Brownlee, and Duane Wessels. "Passive Monitoring of DNS Anomalies". In: *Proceedings of the 4th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. Ed. by Robin Sommer. Vol. 4579. Lecture Notes in Computer Science (LNCS). Extended Abstract. Lucerne, Switzerland: Springer International Publishing, July 2007, pp. 129–139. ISBN: 978-3-540-73613-4. DOI: `10.1007/978-3-540-73614-1_8`.

[239] He Yan, Ricardo Oliveira, Kevin Burnett, Dave Matthews, Lixia Zhang, and Dan Massey. "BGPmon: A real-time, scalable, extensible monitoring system". In: *Proceedings of the 2009 Conference on Cybersecurity Applications & Technology Conference for Homeland Security (CATCH)*. Washington, D.C., USA: Institute of Electrical and Electronics Engineers (IEEE), Mar. 2009, pp. 212–223. ISBN: 978-0-7695-3568-5. DOI: `10.1109/CATCH.2009.28`.

[240] Matthias Wählisch, Olaf Maennel, and Thomas C. Schmidt. "Towards Detecting BGP Route Hijacking Using the RPKI". In: *Proceedings of the 2012 ACM SIGCOMM Conference (SIGCOMM)*. Ed. by Venkat Padmanabhan and George Varghese. POSTER. Helsinki, Finland: Association for Computing Machinery (ACM), Aug. 2012, pp. 103–104. ISBN: 978-1-4503-1419-0. DOI: `10.1145/2377677.2377702`.

[241] Hitesh Ballani, Paul Francis, and Xinyang Zhang. "A Study of Prefix Hijacking and Interception in the Internet". In: *Proceedings of the 2007ACM SIGCOMM Conference (SIGCOMM)*. Ed. by Nina Taft and Anja Feldmann. Kyoto, Japan: Association for Computing Machinery (ACM), Aug. 2007, pp. 265–276. ISBN: 978-1-59593-713-1. DOI: `10.1145/1282380.1282411`.

[242] Zheng Zhang, Ying Zhang, Y. Charlie Hu, and Z. Morley Mao. "Practical Defenses Against BGP Prefix Hijacking". In: *Proceedings of the 2007 ACM CoNEXT Conference (CoNEXT)*. Ed. by Olivier Bonaventure and Roch Guerin. New York, NY, USA: Association for Computing Machinery (ACM), Dec. 2007, 3:1–3:12. ISBN: 978-1-59593-770-4. DOI: `10.1145/1364654.1364658`.

[243] Maarten Aertsen, Maciej Korczyński, Giovane C. M. Moura, Samaneh Tajalizadehkhoob, and Jan van den Berg. "No domain left behind: is Let's Encrypt democratizing encryption?" Dec. 9, 2016. arXiv: `1612.03005 [cs.CR]`.

[244] Antonis Manousis, Roy Ragsdale, Ben Draffin, Adwiteeya Agrawal, and Vyas Sekar. "Shedding Light on the Adoption of Let's Encrypt". Nov. 2, 2016. arXiv: `1611.00469 [cs.CR]`.

[245] Jeremy Clark and Paul C. van Oorschot. "SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements". In: *Proceedings of the 34th IEEE Symposium on Security & Privacy (S&P)*. Ed. by Wenke Lee, Michael Backes, and Adrian Perrig. San Francisco, CA, USA: Institute of Electrical and Electronics Engineers (IEEE), May 2013, pp. 511–525. ISBN: 978-0-7695-4977-4. DOI: `10.1109/SP.2013.41`.

[246] Yanpei Chen, Vern Paxson, and Randy H. Katz. *What's New About Cloud Computing Security*. Tech. rep. UCB/EECS-2010-5. Department of Electrical Engineering and Computer Science, University of California, Berkeley, Jan. 20, 2010. URL: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-5.html` (visited on 08/22/2018).

[247] Subashini Subashini and Veeraruna Kavitha. “A survey on security issues in service delivery models of cloud computing”. In: *Journal of Network and Computer Applications (JNCA)* 34 (1 Jan. 2011), pp. 1–11. ISSN: 1084-8045. DOI: `10.1016/j.jnca.2010.07.006`.

[248] Meiko Jensen, Jörg Schwenk, Nils Gruschka, and Luigi Lo Iacono. “On Technical Security Issues in Cloud Computing”. In: *Proceedings of the 2009 IEEE Conference on Cloud Computing (CLOUD)*. Ed. by Liang-Jie Zhang. Bangalore, India: Institute of Electrical and Electronics Engineers (IEEE), Sept. 2009, pp. 109–116. ISBN: 978-1-4244-5199-9. DOI: `10.1109/CLOUD.2009.60`.

[249] Hassan Takabi, James B. D. Joshi, and Gail-Joon Ahn. “Security and Privacy Challenges in Cloud Computing Environments”. In: *IEEE Security & Privacy* 8.6 (Nov.–Dec. 2010), pp. 24–31. ISSN: 1558-4046. DOI: `10.1109/MSP.2010.186`.

[250] Yue Zhang and James B. D. Joshi. “Information Assurance, Security and Privacy Services”. In: ed. by Shambhu Upadhyaya and H. Raghav Rao. Emerald Group Publishing, June 4, 2009. Chap. Access Control and Trust Management for Emerging Multidomain Environments. ISBN: 978-1848551947.

[251] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Kasper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. “DROWN: Breaking TLS using SSLv2”. In: *Proceedings of the 25th USENIX Security Symposium (USENIX Security)*. Ed. by Thorsten Holz and Stefan Savage. Austin, TX, USA: USENIX Association, Aug. 2016, pp. 689–706. ISBN: 978-1-931971-32-4. URL: `https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram` (visited on 08/23/2018).

[252] John Heidemann, Yuri Pradkin, Ramesh Govindan, Christos Papadopoulos, Genevieve Bartlett, and Joseph Bannister. “Census and Survey of the Visible Internet”. In: *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC)*. Ed. by Konstantina Papagiannaki and Zhi-Li Zhang. Vouliagmeni, Greece: Association for Computing Machinery (ACM), Oct. 2008, pp. 169–182. ISBN: 978-1-60558-334-1. DOI: `10.1145/1452520.1452542`.

[253] 6lab Cisco. *Monitoring IPv6 adoption*. Cisco. URL: `http://6lab.cisco.com/stats/` (visited on 08/23/2018).

[254] Amogh Dhamdhere, Matthew Luckie, Bradley Huffaker, Ahmed Elmokashfi, Emile Aben, and Kimberly C. Claffy. “Measuring the Deployment of IPv6: Topology, Routing, and Performance”. In: *Proceedings of the 2012 Internet Measurement Conference (IMC)*. Ed. by Ratul Mahajan and Alex Snoeren. Boston, MA, USA: Association for Computing Machinery (ACM), Nov. 2012, pp. 537–550. ISBN: 978-1-4503-1705-4. DOI: `10.1145/2398776.2398832`.

[255] Internet Assigned Numbers Authority (IANA). *IPv6 Global Unicast Address Assignments*. URL: `http://www.iana.org/assignments/ipv6-unicast-address-assignments/ipv6-unicast-address-assignments.xhtml` (visited on 08/23/2018).

[256] Lorenzo Colitti, Steinar H. Gunderson, Erik Kline, and Tiziana Refice. "Evaluating IPv6 Adoption in the Internet". In: *Proceedings of the 11th Passive and Active Measurement (PAM)*. Ed. by Arvind Krishnamurthy. Vol. 6032. Lecture Notes in Computer Science (LNCS). Zurich, Switzerland: Springer International Publishing, Apr. 2010, pp. 141–150. ISBN: 978-3-642-12334-4. DOI: `10.1007/978-3-642-12334-4_15`.

[257] Pawel Foremski, David Plonka, and Arthur Berger. "Entropy/IP: Uncovering Structure in IPv6 Addresses". In: *Proceedings of the 2016 Internet Measurement Conference (IMC)*. Ed. by John Heidemann and Phillipa Gill. Santa Monica, CA, USA: Association for Computing Machinery (ACM), Nov. 2016, pp. 167–181. ISBN: 978-1-4503-4526-2. DOI: `10.1145/2987443.2987445`.

[258] Rapid7 Labs. *Forward DNS (FDNS) – (ANY) 2014–2017*. Feb. 1, 2017. URL: `https://scans.io/study/sonar.fdns` (visited on 08/23/2018).

[259] Stephane Bortzmeyer and Shumon Huque. *NXDOMAIN: There Really Is Nothing Underneath*. Tech. rep. 8020. RFC Editor, Nov. 2016. 9 pp. DOI: `10.17487/rfc8020`.

[260] Matthaus Wander, Lorenz Schwittmann, Christopher Boelmann, and Torben Weis. "GPU-based NSEC3 Hash Breaking". In: *Proceedings of the 13th IEEE International Symposium on Network Computing and Applications (NCA)*. Ed. by Alfredo Goldman and Greg Malewicz. Cambridge, MA, USA: Institute of Electrical and Electronics Engineers (IEEE), Aug. 2014, pp. 137–144. ISBN: 978-1-4799-5393-6. DOI: `10.1109/NCA.2014.27`.

[261] Scott Rose and Anastase Nakassis. "Minimizing Information Leakage in the DNS". In: *IEEE Network* 22.2 (Mar.–Apr. 2008), pp. 22–25. ISSN: 0890-8044. DOI: `10.1109/MNET.2008.4476067`.

[262] Roy Arends and Peter Koch. "DNS for Fun and Profit". In: *Proceedings of the 12th DFN-Workshop "Sicherheit in vernetzten Systemen"*. Hamburg, Germany: DFN-CERT, Mar. 2005. URL: `https://www.dfn-cert.de/dokumente/workshop/2005/dfncert-ws2005-f7paper.pdf` (visited on 08/23/2018).

[263] Scott Rose, Ramaswamy Chandramouli, and Anastase Nakassis. "Information Leakage through the Domain Name System". In: *Proceedings of the 2009 Conference on Cybersecurity Applications & Technology Conference for Homeland Security (CATCH)*. Washington, D.C., USA: Institute of Electrical and Electronics Engineers (IEEE), Mar. 2009, pp. 16–21. ISBN: 978-0-7695-3568-5. DOI: `10.1109/CATCH.2009.10`.