Remedying Security Concerns at an Internet Scale

by

Frank Li

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

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Vern Paxson, Chair
Professor David Wagner
Professor Deirdre Mulligan

Fall 2019

Remedying Security Concerns at an Internet Scale

Abstract

Remedying Security Concerns at an Internet Scale

by

Frank Li

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Vern Paxson, Chair

The state of security across the Internet is poor, and it has been so since the advent of the modern Internet. While the research community has made tremendous progress over the years in learning how to design and build secure computer systems, network protocols, and algorithms, we are far from a world where we can truly trust the security of deployed Internet systems. In reality, we may never reach such a world. Security concerns continue to be identified at scale throughout the software ecosystem, with thousands of vulnerabilities discovered each year. Meanwhile, attacks have become ever more frequent and consequential.

As Internet systems *will* continue to be inevitably affected by newly found security concerns, the research community must develop more effective ways to remedy these issues. To that end, in this dissertation, we conduct extensive empirical measurements to understand how remediation occurs in practice for Internet systems, and explore methods for spurring improved remediation behavior. This dissertation provides a treatment of the complete remediation life cycle, investigating the creation, dissemination, and deployment of remedies. We start by focusing on security patches that address vulnerabilities, and analyze at scale their creation process, characteristics of the resulting fixes, and how these impact vulnerability remediation. We then investigate and systematize how administrators of Internet systems deploy software updates which patch vulnerabilities across the many machines they manage on behalf of organizations. Finally, we conduct the first systematic exploration of Internet-scale outreach efforts to disseminate information about security concerns and their remedies to system administrators, with an aim of driving their remediation decisions. Our results show that such outreach campaigns can effectively galvanize positive reactions.

Improving remediation, particularly at scale, is challenging, as the problem space exhibits many dimensions beyond traditional computer technical considerations, including human, social, organizational, economic, and policy facets. To make meaningful progress, this work uses a diversity of empirical methods, from software data mining to user studies to Internet-wide network measurements, to systematically collect and evaluate large-scale datasets. Ultimately, this dissertation establishes broad empirical grounding on security remediation in practice today, as well as new approaches for improved remediation at an Internet scale.

To my parents for their infinite support

Contents

Acknowledgments

Of all the pages I have written throughout my Ph.D., producing this particular page gives me the greatest joy. I have been incredibly fortunate to have worked with, learned from, and been supported by wonderful people, and I would not be the researcher nor the person I am today without them. While I will surely fail to explicitly mention everyone here, I thank all of my family, friends, mentors, collaborators, and students for each and every little thing they have done to help me over the years.

First and foremost, I must thank my Ph.D. advisor Vern Paxson, who has been my sherpa through the winding uncertain paths of my Ph.D. years. Thanks to his constant support, guidance, mentorship, and insights, I have grown into the researcher I am today. I am so very excited to take what I have learned from him as I venture into the next stage of my research career.

I am also particularly thankful to my collaborators at Princeton University, where I visited for a significant portion of my latter two years. Thank you Nick Feamster and Marshini Chetty for extensive mentorship as I explored new research domains. I also thank my comrades in arms there: Danny Yuxing Huang, Gunes Acar, and Arunesh Mathurs. I must also give a special thanks to Danny again, Shuang Shao, and my favorite dog Momo for welcoming me into their home over many visits. My trips would not have been nearly as fruitful or enjoyable without their generosity.

It has also been a great privilege to be part of CESR (the Center for Evidence-Based Security Research). My research perspectives were transformed and shaped by the likes of Vern, Stefan Savage, Geoff Voelker, and Damon McCoy, as well as the many other students, researchers, and faculty involved. Thanks to all for an incredible group experience.

I had the tremendous opportunity of interning not once, but twice, for Kurt Thomas and Elie Bursztein at Google. These internships were fresh summer experiences that did much to rejuvenate my research. I thank them as well as my other collaborators there, including Lucas Ballard, Yuan Niu, Eric Kuan, Luca Invernizzi, and Emily Stark.

Beyond Vern, I thank my UC Berkeley family (a special shout-out to my 719/721 Soda Hall officemates), including those at the International Computer Science Institute. These include but are not limited to: Paul Pearce, Richard Shin, Grant Ho, Austin Murdock, Nathan Malkin, Pratyush Mishra, Rishabh Poddar, Jethro Beekman, Sadia Afroz, Michael Carl Tschantz, Nicholas Weaver, Mobin Javed, Chris Thompson, David Fifield, Neil Zhenqiang Gong, Narseo Vallina-Rodriguez, and Angie Abbatecola.

I am grateful for the opportunity to have worked extensively beyond my immediate communities. I wanted to particularly thank Michael Bailey, Zakir Durumeric, Chris Grier, Wenting Zheng, Raluca Ada Popa, Rachit Agarwal, Roya Ensafi, Ben Stock, Giancarlo Pellegrino, Eric Zeng, and Lisa Rogers, for fruitful collaborations.

I found myself lucky enough to be at Berkeley for my Ph.D. through the guidance of many earlier on. I specifically wanted to acknowledge Raluca Ada Popa for her mentorship during my undergraduate studies at MIT, Prateek Mittal and Matthew Caesar for supporting my first end-to-end research experience during a summer internship at UIUC, and Devdatta Akhawe, Prateek Saxena, and Dawn Song for a wonderful summer at Berkeley as an undergraduate intern.

I would certainly be remiss if I did not also thank my dissertation committee for their time, effort, and feedback. Thank you Vern Paxson, David Wagner, and Deirdre Mulligan.

A final thanks goes to my brother Alex Li and my parents Yuan (Larry) Li and Jie (Judy) Lin, to whom I dedicate this dissertation, for their unwavering love, support, and guidance on life. It seems fitting that the first and last people I thank should be the people who have been with me throughout the entirety of my Ph.D. experience.

To all, thank you.

Chapter 1

Introduction

The Internet has revolutionized our world, from democratizing information to supporting a global economy to providing crucial everyday online services. However, this technological innovation has not come without drawbacks. In large, the Internet and the systems connected to it were not designed and developed with security at the forefront. As a result, billions of systems are connected to the Internet today riddled with security concerns, and the frequency and consequences of attacks exploiting these security holes continue to grow.

This drawback has not gone unnoticed or unaddressed. The security community has made tremendous strides over the years in learning how to more securely design and implement computer systems, network protocols, and algorithms. For example, as a community, we have hardened software programs through static and dynamic analysis, fuzzing, and formal methods. We have better protected networks through intrusion detection systems, firewalls, and cryptographic protocols. We have explored and improved the usability of security techniques to be more readily applied in practice.

Despite all the advances and innovations in computer security, we are still far from a world where the systems we use are fully secure. Each year, thousands of new vulnerabilities are discovered and publicly disclosed [191], affecting software throughout the computing ecosystem. Damaging attacks regularly headline news articles, demonstrating that the threat of exploitation is real. In reality, we may never actually reach an ideal security state. Our software ecosystem is ever evolving and growing at a remarkable pace, permeating into every aspect of our society beyond core technology sectors. In addition, security problems often arise due to interactions with human actors or between systems managed by different stakeholders. It will be challenging, perhaps infeasible, for us to ever develop security methods that can perfectly handle this scale, diversity, complexity, and constant evolution. Even with such developments, there remain significant hurdles including enforcing the universal adoption of such security approaches, as well as addressing legacy designs that are essentially baked into the existing Internet.

Thus, for the foreseeable future, Internet systems *will* continue to be inevitably affected by newly found security issues, and we must effectively remedy these emergent security concerns once uncovered. However, the prevalence today of security attacks that successfully exploit known security vulnerabilities makes it painfully clear that we do not understand yet how to effectively do so. This dissertation aims to fill this gap by conducting extensive empirical measurements to understand how remediation occurs in practice for Internet systems, and exploring methods for spurring improved remediation behavior. Advancing remediation at scale is challenging, though, as the problem space expands beyond traditional computer technical considerations, including human, social, organizational, economic, and policy facets. To make meaningful progress, this work uses a diversity of empirical methods to provide a treatment of the complete remediation life cycle, from the creation to the dissemination to the deployment of security remedies.

In Part I, we start by investigating remedies themselves, to gain insights into what factors contribute to continued vulnerability and what barriers prohibit effective remediation. In particular, we consider how software development and system administration aspects impact vulnerability remediation, essentially looking at both the creation and application of security remedies. In Part II, we turn our focus to the dissemination of remediation information, and explore improving remediation behavior at scale through a techno-social approach. We systematically investigate Internet-scale outreach campaigns that distribute information on security concerns and their remedies, aiming to drive the remediation decisions of system administrators across the Internet. Ultimately, our work develops an empirically-grounded understanding of security remediation in practice today, as well as new directions for better remediation at an Internet scale.

1.1 Part I: Understanding How Security Concerns are Remedied

In the first part of this dissertation, we consider the remedies themselves that address security concerns, focusing on security patches. Applying security patches is one of the most significant facets of managing security, in principle immunizing systems from the thousands of vulnerabilities discovered each year. Prior work has correlated prompt patching with reductions in security incidents, providing rare empirical support for the security contributions of a particular behavior [39]. However, in practice, attackers still regularly exploit at scale vulnerabilities that already have patches, highlighting that security patches do not yet effectively fulfill their immunization role.

This first part strives to understand the barriers that limit the effectiveness of security patches. So far, our grasp of these barriers has been largely anecdotal. This limitation stems from the challenges in obtaining large-scale and representative data on security updating behavior across Internet systems and their administrators. This work develops sound means for collecting such data, using a variety of empirical methods to provide a more systematic and holistic comprehension of security patching considerations.

Chapter 3 - A Large-Scale Empirical Analysis of Security Patches: To understand how vulnerabilities are remedied by security patches in practice, we first consider how such fixes are developed. Prior work, particularly from the software engineering community, has studied bug fixes in general. However, there has been little investigation into how fixes vary across different classes of bugs. Given the vital role of timely security patches in combating security concerns, in this chapter, we conduct a large-scale empirical analysis specifically focused on the security patch development process and the characteristics of the resulting fixes. To perform the analysis, we collect and combine data across multiple disparate sources, including vulnerability databases, various online references (e.g., bug reports, security advisories, email archives), security patches themselves, and the associated software repositories. The resulting dataset, containing over 4K security fixes across a diverse set of 682 open-source software projects, allows us to conduct a deep analysis of the patch development life cycle, including investigations into the timeliness of patch development and the reliability of the security fixes. We also characterize the complexity of security fixes and their impact on code bases, in comparison to other non-security bug patches. We identify how aspects of the patch development process can influence the effectiveness of security fixes, and observe that security patches exhibit complexity and locality characteristics more amenable to program analysis and automatic repair than other bug fixes. The work in this chapter appeared at the ACM Conference on Computer and Communications Security (CCS) [114].

Chapter 4 - Examining How System Administrators Manage Software Updates: We next turn our attention to the application of released security patches, particularly by system administrators remediating their hosts. For many Internet hosts, a system administrator operating on behalf of an organization manages the host's security, and hence its software updates. While recent user studies have examined the software updating behavior of end users, administrators have not been rigorously and systematically studied, in part due to challenges in obtaining data on their actions. However, they are an important and distinct population, whose technical sophistication and unique responsibilities in maintaining their organizations' security distinguish them from end users. In this chapter, we conduct a user study that comprehensively encapsulates system administrator updating practices. We administer surveys and conducted semi-structured interviews with over 100 administrators in total. Our analysis identifies the major components of their update processes, and reveals technical, social, and organizational hurdles that impede their effectiveness. Our findings provide empirical evidence of unique pain points encountered when managing updates for multiple machines in an organization, illuminating avenues for improvements catered towards administrators. The study in this chapter appeared at the USENIX Symposium on Usable Privacy and Security (SOUPS) [115], receiving the Distinguished Paper Award.

1.2 Part II: Promoting Remediation through Internet-Scale Outreach

Part I of this dissertation investigates the remedies themselves and how they are developed and applied, specifically focused on security patches. A step remains between those two actions: those affected must learn about the security issues and their remedies, and decide to take action. Part II investigates this stage of the remediation life cycle, where information on remedies is disseminated.

Recent advances in Internet scanning have allowed us to analyze the security posture of Internet systems at an unprecedented scale. However, when a widespread problem is uncovered, such as a security misconfiguration, we resort to ad hoc and inefficient methods for disclosing the information to the affected parties. For example, we often propagate information through papers and reports, website postings, social media, mailing lists, and occasionally news outlets, in hopes of reaching those managing afflicted hosts. This process is slow and untargeted, lacks comprehensive coverage, and often fails when administrators do not recognize that their hosts are affected.

We need better methods for distributing security information and promoting remedies. In this part, we investigate leveraging our Internet scanning capabilities to identify vulnerable hosts and directly reach out to their administrators, notifying them about the security problem and their vulnerable systems to encourage remediation. While conceptually simple, an effective administrator notification method would provide us with an approach for proactively combating emergent security concerns at scale, in an efficient, comprehensive, and targeted fashion. However, how best to notify in practice is far from clear. Moreover, a common assumption held in the security community, largely fueled by historical anecdotes and beliefs that users neither care about nor can manage security, is that such outreach efforts are futile. Challenging this assumption, the studies in this dissertation have been among the first to systematically explore Internet-scale outreach efforts to spur positive remediation behavior, finding that security notifications can be surprisingly effective. As a result, other security researchers in academia, industry (e.g., Google), and government organizations (e.g., national CERTs) have begun conducting similar notifications for newly identified security problems.

Studying how to conduct effective administrator notification campaigns is difficult, requiring the careful design of controlled Internet-scale experiments that respect ethical considerations. Another fundamental complication is that notification effects may differ depending on the security issue at hand. As illustrated by the studies included in this dissertation, our exploration has moved us towards a general understanding of administrator security notifications by experimenting across a variety of different security issues, finding consistent results.

Chapter 7 - The Matter of Heartbleed: In this chapter, we perform an extensive analysis of the Internet's response to the notorious Heartbleed bug [128]. As part of this investigation, we conduct a large-scale A/B experiment to investigate if we can promote patching by sending email vulnerability notifications to network administrators. We find that we increased the patching rate by nearly 50% for those notified compared to those in the control.[1] Through surveying the contacted administrators, we learn that they were often unaware of their vulnerable hosts until our messages, and hence would not have patched otherwise. Throughout our outreach campaign, administrators reacted positively to our notifications, with a majority voicing support for similar efforts in the future. The success of this first foray into vulnerability notifications spurred our further investigation into this space. This chapter's work was published in the ACM Internet Measurement Conference (IMC) [65], and was awarded Best Paper.

Chapter 8 - Remedying Web Hijacking: In this chapter, we build upon our initial foray into Internet-wide outreach and evaluate the effectiveness of Google's notifications sent to webmasters of compromised websites. Hijacked websites are a unique problem as recovering from the compromise is a particularly challenging task, and webmasters may fail to successfully clean sites up even if notified. Despite the obstacles, we find that webmasters were twice as likely to clean up their sites when notified, and they recovered in half the time. In total, nearly 80% of notified

[1]For ethical reasons, we did notify the control group after the experiment.

websites remediated, indicating that notifications *can* drive a significant majority of recipients to rectify even complex security situations. This work, published at the World Wide Web Conference (WWW) [113], has stimulated the expansion of Google's notification efforts to other domains, such as HTTPS website misconfigurations.

Chapter 9 - Exploring Effective Vulnerability Notifications: The prior studies established that administrator security notifications can significantly increase remediation levels. But how does one most effectively communicate information about security problems? To answer this question, in this chapter we conduct controlled multivariate notification experiments across multiple populations, evaluating variables such as whom to contact, amount of message detail, inclusion of external references, and translation of messages into local languages. For experiment populations, we conduct Internet-scale network scans to identify hosts exhibiting one of three widespread network service misconfigurations (overly open IPv6 firewalls, DDoS amplifiers, and exposed industrial control systems).

We again find that our outreach promoted remediation at a significant fraction of notified contacts, and that our notifications were well received. Different notification regimens *did* result in varying outcomes. The best observed process, consistent across the misconfigurations, was sending detailed notifications to WHOIS abuse contacts with a link to an information website, *without* translations. We were initially surprised that message translation resulted in less effective notifications, but through surveying message recipients, we identified that this was due to an issue of trust. They expected English from a US institution and were more suspicious of a translated message. These results provide evidence-based recommendations for conducting administrator security notifications today. The work in this chapter appeared at the USENIX Security Symposium [112].

Part I

Understanding How Security Concerns are Remedied

Chapter 2

Remediation Introduction and Related Work

2.1 Introduction

Thousands of software vulnerabilities continue to be discovered year after year [191], plaguing the security of Internet systems. When these security concerns arise, system administrators should take action to remedy the issues before attackers discover and exploit the security hole. In practice, we recognize that administrators must not be doing so effectively, as devastating attacks leveraging known vulnerabilities are successfully conducted frequently. These attacks have resulted in headline-grabbing data breaches [146], as well as compromise of Internet systems at scale.

To improve how we remedy emergent security problems, we must understand what factors contribute to continued vulnerability and what barriers limit effective remediation. In Part I of this dissertation, we look to expand our understanding of these facets by investigating remedies themselves, specifically focusing on security patches that seek to eliminate security vulnerabilities. (We briefly note that there are other types of security issues, such as security misconfigurations and compromise incidents, each with different forms of remedies.)

In Chapter 3, we start by performing a large-scale empirical study of security patches, exploring how their development processes and characteristics can impact remediation. We develop a method for collecting thousands of security patches affecting a large set of open-source software projects, providing a more diverse and expansive dataset than considered in prior work. This dataset affords us the opportunity to conduct a deep analysis of the patch development process, including investigation into the life span of vulnerabilities in code bases, the timeliness of security fixes, and the reliability and safety of the security patches produced. We also characterize the security fixes themselves in comparison to non-security bug patches, considering characteristics such as the complexity of the patches as well as their impact on code bases. The findings of this work provide empirical grounding on the real-world characteristics of security patches and their development process, as well as the impact on patching behavior.

In Chapter 4, we turn to the application of security patches through software updates by system administrators. System administrators are typically the primary caretakers of organizations' Internet systems, managing their operation and security. To improve how we remedy security concerns, it is vital that we understand better how administrators manage updates in practice, and identify avenues for improvements in existing processes. To date, we have little empirical grounding on administrator updating behavior. Existing studies have focused on the behavior of end users, who are a drastically different user population; system administrators are often technically sophisticated and operate with expanded responsibilities managing machines on behalf of an organization. In this chapter, we conduct an extensive user study of system administrator software updating through surveying and interviewing over 100 administrators. Our findings demonstrate that while the high-level stages of software updating are similar between end users and administrators, the considerations and actions at each stage are significantly different, highlighting the value in a study focused on administrators. Our study also identified challenges that administrators encountered and limitations of existing procedures at each stage of the updating process, providing insights on paths forward for improving administrator updating.

Together, the chapters of Part I explore both the development and application of vulnerability remedies for Internet systems. By establishing an empirically-grounded understanding of how remediation operates in practice, we identified problems with existing processes and ways to potentially improve them moving forward, providing guidance for the security community in advancing remediation methods in the future. In Part II of this dissertation, we then explore how information on remedies is distributed, and investigate a class of approaches for driving better remediation behavior at an Internet scale through outreach campaigns.

2.2 Related Work

Here we outline prior work related to the investigations explored in Part I. We first consider related work analyzing security vulnerabilities and how they are remedied through security patches, relevant to our study of security patch development in Chapter 3. Then we survey existing studies on how these remedies are deployed through software updates, related to our exploration of system administrator software updating in Chapter 4.

2.2.1 Developing Security Patches

A body of work has investigated aspects of the vulnerability and patching life cycle. Frei et al. [74] and Shahzad et al. [175] conducted similar analyses based on public documentation from vulnerability databases and security advisories. For example, they compared a vulnerability's public disclosure date with announcement dates for fixed releases available for distribution, finding them concurrent in 60% of cases. Ozment et al. [156] investigated the evolution of vulnerabilities in the OpenBSD OS over time, observing that it took on average 2.6 years for a release version to remedy half of the known vulnerabilities. Huang et. al. [97] manually analyzed 131 cherry-picked security patches from five open-source projects, demonstrating that there exist cases where patch development was lengthy and error-prone. Nappa et al. [141] shed light on the patch deployment process from an end-user perspective, analyzing when security updates were available to clients and how quickly clients patched. In Chapter 3, we extensively explore new aspects of patch development dynamics that require merging information collected from vulnerability databases with that gleamed from software source code, such as vulnerability life spans in the code base and the timeliness of patching the code base relative to public disclosure. In addition, we aim to generate generalizable insights by studying a diverse set of over 650 open-source projects.

Explorations of bug remedies in general (beyond just security bugs) have been performed in the software engineering community. Zhong and Su [212] conducted an empirical study of over 9,000 bug fixes across six Java projects. They framed their investigation around patch properties that would make them suitable for generation by automatic program repair, finding that the majority are too complex or too delocalized to likely be automatically created. Similarly, Park et al. [158] studied supplementary bug fixes, additional fixes produced when the initial fix was incomplete. Their analysis covered three open-source projects and showed that over a quarter of remedies required multiple patches. Sliwerski et. al. [176] investigated two projects and correlated updates that required fixes with the update sizes, finding larger updates were more likely to require subsequent fixes. Soto et. al. [178] applied common bug fix patterns to Java patches, finding that less than 15% could be matched.

While these works are similar to our investigation in Chapter 3 in their focus on patch characteristics, they mostly were conducted at a smaller scale, and do not differentiate between different kinds of bugs. Security patches are of special interest, given their importance in protecting users and the time sensitivity of their development. We seek to tease apart the differences between security and non-security bug fixes, a distinction that has not been previously scrutinized extensively. Most relevant is a case study performed by Zama et al. [208] on security and performance fixes in Mozilla Firefox. They noted differing remediation rates and complexities between security and performance patches. Perl et. al. [160] also analyzed Git commits that fixed vulnerabilities to produce a code analysis tool that assists in finding dangerous code commits. They found that indicative features of problematic commits include code which handles errors or manages memory, or is contributed by a new project developer. Most recently, Xu et. al. [205] developed a method for identifying security patches at the binary level based on execution traces, providing a method for obtaining and studying security patches on binaries and closed-source software. These early findings highlight the importance of considering different types of software bugs; a deep understanding of security patches and their development process can inform the security community in matters related to vulnerability management.

2.2.2 Deploying Security Patches through Software Updates

Once a security patch is developed, software users much apply the patch to remediate, typically through a software update. Understanding this process and current limitations is vital for improving this remediation step. In Chapter 4, we study how system administrators apply software updates to a multitude of Internet systems on behalf of their organizations. However, prior work has investigated software updating, particularly from end user perspectives.

End Users and Software Updates. Numerous works [69, 71, 101, 123–125, 141, 150, 193, 194, 200, 201] have examined end user perceptions, attitudes, and behavior towards applying software updates. Ion *et al.* [101] and Wash *et al.* [200] found that non-expert computer users failed to recognize the security benefits of updates and frequently avoided installing them. Other studies measured the time users took to apply updates and discovered that reaching half of all vulnerable desktop [141] and mobile applications [150] took nearly 45 days and 1 week, respectively. One set of studies has examined why users avoid or fail to install software updates, discovering a variety of factors related to costs, necessity, and risks [125]. Example factors include that updates cause unexpected changes to user interfaces [71, 124, 193, 194], that updates take a long time to install [124, 194], that updates raise privacy concerns [69], and that updates require unnecessary restarts of applications [71, 124, 193, 194].

Given that automatic updates are more effective than manual updates in keeping end user systems updated [62, 76, 141], another set of studies has examined user attitudes towards and experiences with automatic updates [123, 201]. Rader and Wash [201] identified that automatic updates with only partial user involvement (e.g., during restarts) often led to poor mental models of updating, and consequently resulted in less secure systems. More recently, Mathur and Chetty [123] found that negative experiences with automatic updating resulted in users disabling auto-updates on Android devices.

While these studies have shed light on how end users deal with software updates, their findings do not necessarily generalize to system administrators, who are more technically sophisticated and operate with expanded responsibilities managing an organization's Internet systems. In Chapter 4, we investigate this vital subpopulation.

Administrators and Software Updates. Several studies [51, 106, 108, 109, 197, 198] have examined the workflows and needs of administrators to enable better security practices, but did not focus on software updating processes specifically. Kraemer and Carayon [108] conducted interviews with 16 network administrators and security workers, identifying that organizational structures and policies played an important role in how they handled security. Kandogan *et al.* [106] discussed various stories from IT administrators about their experiences. Krombholz *et al.* [109] investigated usability problems encountered by website administrators trying to securely deploy HTTPS. Chiasson *et al.* [51] devised usability and interface design principles to help system administrators better diagnose security issues. Velasquez and Weisband [197] conducted interviews with administrators and designed a model to understand their beliefs and attitudes. In this model, the authors identified that both informational factors (e.g., quality) and system factors (e.g., ease of use) informed these beliefs and attitudes. In a follow-up study [198], the same authors found that administrators largely acquired their knowledge through practice rather than education and certification. They recommended that software developers should design tools with administrator technical sophistication in mind.

Closely related to our own work in Chapter 4 is the preliminary study conducted over a decade ago by Crameri *et al.* [55]. Although not the primary focus of their work, these researchers conducted brief surveys of 50 system administrators to learn about their updating practices. They found that nearly 70% of administrators refrained from installing software updates and that administrators tested updates on a smaller set of machines before patching their production systems. The study investigated certain aspects of administrator behavior to inform the design of their update testing system, but did not perform a comprehensive and rigorous exploration of update management. More recently, Dietrich *et al.* [58] looked at how system administrator operations could result in security misconfigurations, finding that missing and delaying software updates are among the most commonly reported security misconfigurations.

Unlike these previous studies, our work in Chapter 4 provides an in-depth investigation of system administrator practices for updating the machines they manage. Using a combination of surveys and interviews, we examine a larger sample of administrators than Crameri *et al.* [55] and provide more recent and in-depth insights into their complete update management process.

Chapter 3

A Large-Scale Empirical Study of Security Patches

3.1 Introduction

Miscreants seeking to exploit computer systems incessantly discover and weaponize new security vulnerabilities. As malicious attacks become increasingly advanced, system administrators continue to rely on many of the same processes as practiced for decades to update their software against the latest threats. Given the central role that the "patching treadmill" plays in countering emergent security concerns, it behooves the security community to understand the patch development process and the characteristics of the resulting fixes. Illuminating the nature of security patch development can inform us of shortcomings in existing remediation processes and provide insights for improving current practices.

Seeking such understanding has motivated several studies exploring various aspects of vulnerability and patching life cycles. Some have analyzed public documentation about vulnerabilities, such as security advisories, to shed light on the vulnerability disclosure process [74, 175]. These studies, however, did not include analyses of the corresponding code bases and the patch development process itself. Others have tracked the development of specific projects to better understand patching dynamics [97, 156, 208]. While providing insights on the responsiveness of particular projects to security issues, these investigations have been limited to a smaller scale across a few (often one) projects.

Beyond the patch development life cycle, the characteristics of security fixes themselves are of particular interest, given their importance in securing software and the time sensitivity of their development. The software engineering community has studied bug fixes in general [158, 176, 178, 212]. However, there has been little investigation into how fixes vary across different classes of issues. For example, one might expect that patches for performance issues qualitatively differ from those remediating vulnerabilities. Indeed, Zama et al.'s case study on Mozilla Firefox bugs revealed that developers address different classes of bugs differently [208].

In this chapter, we conduct a large-scale empirical study of security patches, investigating

4,000+ bug fixes for 3,000+ vulnerabilities that affected a diverse set of 682 open-source software projects. We build our analysis on a dataset that merges vulnerability entries from the National Vulnerability Database [191], information scraped from relevant external references, affected software repositories, and their associated security fixes. Tying together these disparate data sources allows us to perform a deep analysis of the patch development life cycle, including investigation of the code base life span of vulnerabilities, the timeliness of security fixes, and the degree to which developers can produce safe and reliable security patches. We also extensively characterize the security fixes themselves in comparison to other non-security bug patches, exploring the complexity of different types of patches and their impact on code bases.

Among our findings we identify that: security patches have less impact on code bases and result in more localized changes than non-security bug patches; security issues reside in code bases for years, with a third introduced more than 3 years prior to remediation; security fixes are poorly timed with public disclosures, allowing attackers who monitor open-source repositories to get a jump of weeks to months on targeting not-yet-patched systems prior to any public disclosure and patch distribution; nearly 5% of security fixes negatively impacted the associated software; and 7% failed to completely remedy the security hole they targeted. The findings of our analysis provide us with insights that suggest paths forward for the security community to improve vulnerability management.

Our work provides several contributions. First, we specifically focus on security vulnerabilities and their patches. While aspects of our work have similarities to prior efforts from the software engineering community that examined general bug fixes [158,176,178,212], we tease apart the differences between security fixes vs. other bug fixes. Second, we develop a large-scale reproducible data collection methodology and associated analysis that ties extensive meta-data on vulnerabilities and their patches with the software source codes and change histories. As best we know, such a diverse set of data has not been previously collected and used to explore security patch development at scale. Conducting such an analysis at scale provides a third contribution: some prior works have considered analyses somewhat similar, but restricted to a small handful of software projects (often only one). We develop robust metrics that one can compute across a diverse group of projects, supporting a range of generalizable results.

The work in this chapter appeared at the ACM Conference on Computer and Communications Security (CCS) [114].

3.2 Data Collection Methodology

To explore vulnerabilities and their fixes, we must collect security patches and information pertaining to them and the remedied security issues. Given this goal, we restricted our investigation to open-source software for which we could access source code repositories and associated meta-data. Our data collection centered around the National Vulnerability Database (NVD) [191], a database provided by the U.S. National Institute of Standards and Technology (NIST) with information pertaining to publicly disclosed software vulnerabilities. These vulnerabilities are identified by CVE (Common Vulnerabilities and Exposures) IDs [133]. We mined the NVD and crawled external

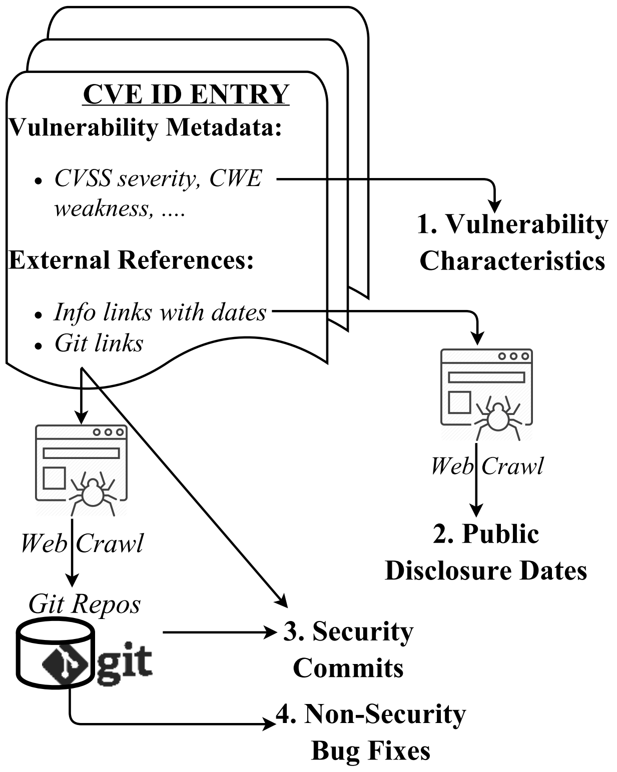


Figure 3.1: An overview of our data collection methodology. 1. We extracted vulnerability characteristics from CVE entries in the NVD with external references to Git commit links. 2. We crawled other references and extracted page publication dates to estimate public disclosure dates. 3. We crawled the Git commit links to identify and clone the corresponding Git source code repositories, and collected security fixes using the commit hashes in the links. 4. We also used the Git repositories to select non-security bug fixes.

references to extract relevant information, including the affected software repositories, associated security patches, public disclosure dates, and vulnerability classifications. Figure 3.1 depicts an overview of this process. In the remainder of this section, we describe these various data sources and our collection methodology.

Note that throughout our methodology, we frequently manually inspected random samples of populations to confirm that the population distributions accorded with our assumptions or expectations. We chose sample sizes (typically of 100) such that they proved manageable for manual analysis while large enough to reflect fine-grained aspects of population distributions.

3.2.1 Finding Public Vulnerabilities

We relied on the NVD to find publicly disclosed vulnerabilities. The NVD contains entries for each publicly released vulnerability assigned a CVE identifier. When security researchers or vendors identify a vulnerability, they can request a CVE Numbering Authority (such as the MITRE Corporation) to assign a CVE ID to it. At this point, information about the vulnerability may not yet be disclosed. Upon public release of the vulnerability information, the CVE ID along with its associated vulnerability information gets added to the CVE list, which feeds the NVD. NVD analysts investigate the vulnerability further, populating an entry for the CVE ID with additional information. In particular, they summarize the vulnerability, link to relevant external references (such as security advisories and reports), enumerate the affected software, identify the class of security weakness under the Common Weakness Enumeration (CWE) classifications [134], and evaluate the vulnerability severity using the Common Vulnerability Scoring System (CVSS) [72, 189].

While there exist other vulnerability databases (e.g., *securityfocus.com*, IBM's X-Force Threat Intelligence, and *securitytracker.com*), we focused on the NVD as it is: (1) public, free, and easily accessible in XML format, allowing for reproducibility and follow-on studies, (2) expansive, as the NVD aims to catalog all publicly disclosed vulnerabilities across numerous software packages, (3) manually vetted and curated, which in theory provides more accurate data, and (4) detailed, containing extensive documentation of vulnerabilities (notably external references).

We utilized the NVD XML dataset [192] as snapshotted on December 25th, 2016. Its 80,741 CVE vulnerabilities served as our starting point for further data collection.

3.2.2 Identifying Software Repositories and Security Patches

Many open-source version-controlled software repositories provide web interfaces to navigate project development (such as *git.kernel.org*). We frequently observed URLs to these web interfaces among the external references for CVE entries, linking to particular repository commits that fixed the security vulnerability. These links afforded us the ability to collect security patches and access the source code repositories.

As Git is arguably the most popular version control system for open-source software [169], we focused on references to Git web interfaces. This popularity was consistent with the CVE external references as well, where links to Git web interfaces were by far the most common. We observed more than 5,700 unique URLs with "git" as a substring, excluding those with another common substring "digit". To determine if these URLs were indeed related to Git, we randomly sampled 100 URLs. The vast majority of these were associated with Git web interfaces; only two out of the 100 URLs were non-Git URLs. In comparison, 1,144 external references contained "svn" (for SVN), 613 contained "cvs" (for CVS), and 347 contained "hg" or "mercurial" (for Mercurial), significantly fewer for these other popular version control systems compared to Git.

To find Git repositories and their security patches, we first reverse-engineered the URL paths and parameters used by popular Git web interfaces. These included cgit [8], GitWeb [14], github.com, and GitLab [13], and accounted for 95% of references with "git" as a substring. (Thus, to consider more Git web interfaces would have required additional URL reverse-engineering while producing diminished returns.) We also identified only an additional 128 URLs without "git" that were consistent with a common Git web interface, suggesting that we identified the majority of Git URLs. For the 80% of these Git URLs that linked to a particular commit (specified in Git by a commit hash), we crawled the web interfaces' summary/home pages and extracted the Git clone URLs, if listed.

In total, we retrieved 4,080 commits across 682 unique Git repositories, tied to 3,094 CVEs. Note that these repositories are distinct, as we de-duplicated mirrored versions. It is possible that some commits are not security fixes, as they may instead reference the change that introduced the vulnerability, or may contain a proof-of-concept exploit instead. However, we found that this is rarely the case. By manually investigating 100 randomly sampled commits, we found that all commits reflect fixes for the corresponding vulnerabilities.

3.2.3 Identifying Non-Security Bug Fixes

We can gain insight into any particularly distinct characteristics of security patches by comparing them to *non-security* bug fixes. However, to do so at scale we must automatically identify non-security bug fixes. We tackled this problem using a logistic regression that models the character n-grams in Git commit messages to identify likely bug fix commits.[1]

To train our commit classifier, we manually labeled 400 randomly selected commits drawn from all Git repositories as bug fixes or non-bug fix commits (136 were bug fixes). We then featurized a commit message into a binary vector indicating the presence of common character n-grams in the commit message. To determine the size of n-grams, the threshold on the number of n-grams to include, and model parameters, we ran a grid search using 10-fold cross-validation on the training data. Our feature vector search space considered n-grams of lengths 2 to 10 and feature vectors that included the top 10,000 to the top 250,000 most frequently occurring n-grams for each class. Our model parameter search space considered both L_1 and L_2 regularization, with regularization strengths ranging from 0.1 to 10, and the inclusion of a bias term.

Our final classifier utilized n-grams of lengths 3 to 9, with feature vectors corresponding to the top 50,000 most common n-grams for each class. The model used L_2 regularization with a regularization strength of 1, and included a bias term. During 10-fold cross-validation, the classifier had an average recall of 82% and precision of 91%. While the classifier is not extremely accurate, it results in only a small fraction of false positives and negatives, which should have limited effect on the overall distributions of patch characteristics. In Section 3.4.2, we compare characteristics of security patches versus generic bug fixes. We manually validated that for these characteristics, the distribution of values is similar between our manually labeled bug fixes and our classifier-collected bug patches, indicating that our results for classifier-labeled bug fixes should be representative of randomly selected true bug fixes.

With our classifier, we collected a dataset of bug fixes by randomly selecting per repository up to 10 commits classified as bug fixes. (Fewer for repositories with less than 10 total commits.) We chose to select 10 commits per repository as that provided us with a large set of over 6,000 bug fixes (similar to our number of security fixes) balanced across repositories. Note that in our classifier training, security fixes were labeled as bug fixes. However, only 6% of bug fixes in our

[1]We also explored other commit features for classification, such as the number of files and lines affected by a commit, the type of commit changes (addition, deletion, modification), the day of week the commit was made, and the time since the previous commit. However, these did not provide adequate discriminating power.

training data (a random sample) were security-related, thus our dataset consists almost entirely of non-security bug fixes.[2]

3.2.4 Processing Commits

For each commit we collected (both security and non-security patches), we extracted the historical versions of affected files both before and after the commit. The diff between these file versions is the patch itself. In addition, it is often useful to consider only changes to functional source code, rather than documentation files or source code comments. We processed the commit data using a best-effort approach (as follows) to filter non-source code files and remove comments, providing an alternative "cleaned" commit to analyze.

To do so, we mapped the top 30 most frequently occurring file extensions to the programming or templating languages associated with them, if any (e.g., an extension of *.java* corresponds to Java, whereas we assume *.txt* reflects non-source code). These included C/C++, PHP, Ruby, Python, SQL, HTML, Javascript, Java, and Perl. We stripped comments and trailing whitespaces under the assumed programming language's syntax for source code files, and filtered out all other files. This provided a cleaned snapshot of files involved in a commit, from which we computed a cleaned diff.

This method is ultimately best-effort,[3] as we handled only the top 30 extensions and relied on extensions as file type indicators. However, we note that these top 30 extensions accounted for 95% of commit files, and incorporating additional extensions would have resulted in diminishing returns given that each extension potentially required a new cleaning process. Also, in a random sample of 100 files with a top 30 extension, all extensions corresponded correctly to the expected file type. This is unsurprising given these projects are open-source and often involve a number of developers, which likely discourages a practice of using non-intuitive and non-standard file extensions.

3.2.5 Estimating Vulnerability Public Disclosure Dates

Determining the public disclosure date of a vulnerability is vital to understanding the timeline of a vulnerability's life cycle. NVD entries contain a CVE publication date that corresponds to when the vulnerability was published in the database, not necessarily when it was actually publicly disclosed [190]. To obtain a more accurate estimate of the public disclosure date, we analyzed the external references associated with CVEs. These web pages frequently contain publication dates

[2]We also investigated developing a commit message classifier to automatically distinguish between security and non-security fixes, using as ground truth the manually-labeled commits as well as randomly selected CVE-related security fixes. Given the base rate challenge arising due to the relative rarity of security fixes, we found that the classifiers we tried did not provide nearly enough accuracy. We did not consider using patch characteristics (such as those explored in Section 3.4.2) as features as we aimed to understand how security and non-security bug fixes differed along these properties, thus using such features would provide skewed populations.

[3]We also evaluated using the Linux "`file`" utility, but found it suffered from frequent errors.

	Domain	# References
1.	openwall.com	2,413
2.	ubuntu.com	2,055
3.	lists.opensuse.org	1,784
4.	securityfocus.com	1,505
5.	rhn.redhat.com	1,328
6.	bugzilla.redhat.com	1,158
7.	debian.org	830
8.	lists.fedoraproject.org	673
9.	oracle.com*	573
10.	mandriva.com*	540
11.	vupen.com*	482
12.	xforce.iss.net*	422
13.	marc.info	305
14.	support.apple.com	259
15.	securitytracker.com	235
16.	lists.apple.com	235
17.	seclists.org	204
18.	bugs.wireshark.org	143
19.	bugs.php.net	127
20.	security.gentoo.org	102

Table 3.1: List of the 20 most common externally referenced sites for CVEs corresponding to our collected security Git commits. We crawled references to these sites for publication dates to better estimate vulnerability public disclosure dates, although not all web pages were still active. Note that 4 sites (marked with asterisks) were no longer active, did not provide publication dates, or employed anti-crawling measures.

for information pertaining to vulnerabilities, which can serve as closer estimates of the public disclosure dates.

For the CVEs corresponding to our collected security commits, we identified the top 20 most commonly referenced sites that may contain publication dates, listed in Table 3.1. Of these, two sites were no longer active (*mandriva.com* and *vupen.com*), one did not provide fine-grained dates (*oracle.com*), and IBM's Threat Intelligence X-Force site employed aggressive anti-crawling measures. For the remaining 16 sites, we constructed per-site parsers that extracted the date of the relevant publication for a given page. These pages include security advisories (such as from Debian and Redhat), mailing list archives (e.g., *marc.info*, *openwall.com/lists*), other vulnerability database entries (e.g., *securityfocus.com*, *securitytracker.com*), and bug reports (such as Bugzilla bug tracking sites). We restricted our crawling to the top 20 sites, as each site required developing a new site parser, and we observed diminishing returns as we added more sites.

We crawled about 13,600 active external references in total, extracting a publication date from

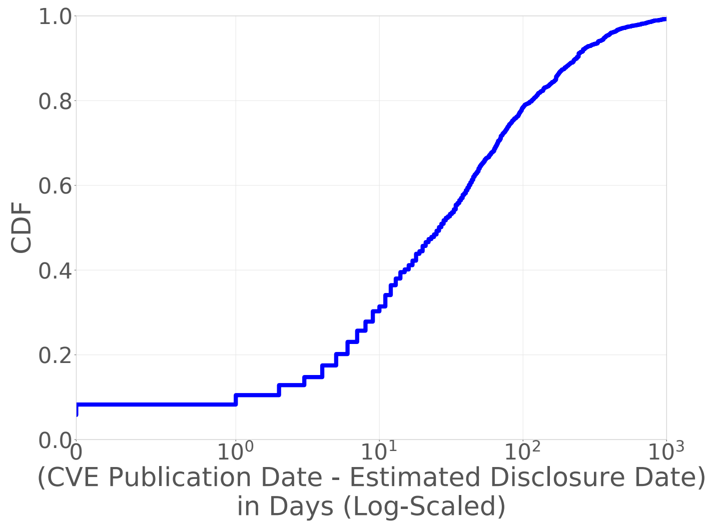


Figure 3.2: CDF of the number of days the estimated disclosure date precedes the CVE publication date.

94% of pages. This provided at least one date extracted from an external reference for 93% of CVEs, with multiple dates extracted for 73% of CVEs. To confirm the soundness of this strategy, we randomly sampled 100 crawled pages, finding all relevant dates were correctly extracted.

We estimate the earliest disclosure date as the earliest amongst the extracted reference dates and the CVE publication date. While this is a best-effort approach, we observe that it yields *significantly* improved disclosure estimation. Figure 3.2 plots the CDF of the number of days the estimated disclosure date precedes the CVE publication date. For approximately 8% of CVEs, we did not extract an earlier external reference date, resulting in no improvement for disclosure estimation. However, the median difference is nearly a month (27 days). At the extreme, we witness differences on the order of years. These correspond to vulnerabilities that are assigned CVE IDs and publicly disclosed, but are not published to the NVD until much later. For example, CVE-2013-4119 is a vulnerability in FreeRDP that was first discussed on an OpenWall mailing list in July, 2013 and assigned a CVE ID. However, its NVD entry was not published until October, 2016, resulting in a large discrepancy between the CVE publication date and the true disclosure date. Thus, our method provides us with significantly improved disclosure date estimates.

3.2.6 Limitations

Vulnerability databases (VDBs) can provide rich sources of data for analysis of security issues and fixes. However, we must bear in mind a number of considerations when using them:

1. **Vulnerability Granularity:** By relying on the NVD, we can only assess vulnerabilities at CVE ID granularity. While CVE IDs are widely used, alternative metrics exist for determining what qualifies as a distinct vulnerability [52].

2. **Completeness:** No VDB is complete, as they all draw from a limited set of sources. However, by using a VDB as expansive as the NVD, we aim for our analysis to provide meaningful and generalizable insights into vulnerabilities and security fixes.
3. **Quality:** The NVD data is manually curated and verified when a vulnerability is assigned a CVE ID, which ideally improves the data quality. However, given the sheer number of vulnerabilities reported, the NVD may contain errors. Throughout our analysis, we aim to identify and investigate anomalous data as part of our methodology for reducing the impact of faulty information.
4. **Source Bias:** A VDB may be biased towards certain vulnerabilities or types of software, depending on their vulnerability data sources. Given the extensive range of software considered by the NVD, we anticipate that our findings will remain largely applicable to open-source software.
5. **Reporting Bias:** Security researchers may exhibit bias in what security issues they investigate and report, potentially affecting a VDB's set of vulnerabilities. For example, researchers may focus more on publishing high-severity issues, rather than low impact, hard-to-exploit vulnerabilities. Additionally, researchers may favor investigating certain vulnerability types, such as SQL injections or buffer overflows. As a result, we can find raw vulnerability counts ineffective for comparing trends in the security status of software, and we avoid drawing conclusions from such analysis.

In addition to the above considerations, our data collection methodology introduces bias towards open-source software projects, particularly those using Git for versioning. Thus, our findings might not directly apply to other software systems, such as closed-source ones. However, our dataset does provide a diverse sample of 682 software projects.

Finally, our methodology and analyses do rely on some approximations. With a diverse dataset of different types of vulnerabilities across numerous projects, we argue that approximations will often prove necessary, as more accurate metrics would require perhaps intractable levels of manual effort. For example, evaluating a vulnerability's life span requires understanding the context about the vulnerability type and the code logic. An automated approach, if feasible, likely still requires developing a different method for each vulnerability class, and perhaps each type of project. Prior case studies [103, 107, 156] that considered vulnerability life spans relied on manual identification of vulnerability introduction, limiting their scope of investigation. When we do use approximations, we use conservative methods that provide upper/lower bounds in order to still obtain meaningful insights. However, we acknowledge that these bounds may not fully reflect observed effects or properties.

3.3 Data Characterization

In this section, we explore the characteristics of the selected CVEs and the collected Git software repositories.

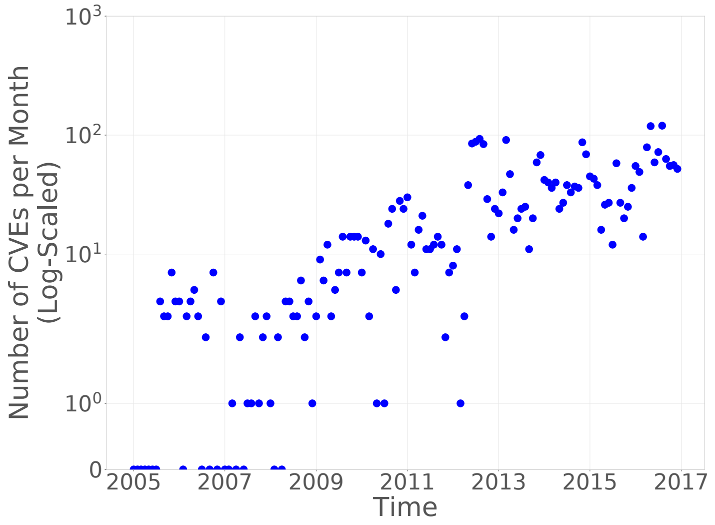


Figure 3.3: Timeline of CVEs with collected security fixes, grouped by publication month.

3.3.1 Vulnerability Publication Timeline

In total, we collected 4,080 security fixes for 3,094 CVEs (implying multiple security fixes for some CVEs, an aspect we explore further in Section 3.4.1.3). The earliest CVE with a collected security patch was published on August 4, 2005, and the most recent on December 20, 2016. In Figure 3.3, we plot the timeline of these CVEs, bucketed by the publication month. We observe that our CVE dataset spans this 11 year period, although it exhibits skew towards more recent vulnerabilities. Note that, as discussed in Section 3.2.6, these raw counts do not imply that our studied software projects have become more vulnerable over time. Rather the increase may reflect other factors such as additional reporting by security researchers.

3.3.2 Affected Software Products

The NVD also enumerates software products affected by a particular vulnerability for all CVEs in our dataset. We observe a long tail of 856 distinct products, with the top 10 listed in Table 3.2. The number of products affected exceeds the number of software projects we collected because a CVE vulnerability in one project can affect multiple products that depend on it. Similarly we note that many of the top affected products are Linux distributions, as a vulnerability that affects one distribution frequently occurs in others. This bias in our CVE dataset towards Linux-related vulnerabilities informs us of the importance of per-repository analysis, in addition to aggregate analysis over all CVEs. Such analysis equally weighs the influence of each software project on any computed metrics.

	Top Products	**# CVEs**
1.	Linux Kernel	917
2.	Ubuntu	211
3.	FFmpeg	187
4.	Debian	170
5.	Wireshark	146
6.	openSUSE	134
7.	PHP	125
8.	Android	121
9.	Fedora	105
10.	QEMU	77

Table 3.2: The top 10 software products by the number of associated CVE IDs.

3.3.3 Vulnerability Severity

The NVD quantifies the severity of vulnerabilities using a standardized method called CVSS (version 2) [72,189]. While the CVSS standard is imperfect [138], it provides one of the few principled ways to characterize vulnerability risk and potential impact. We use this score as is, however acknowledging the difficulties in objectively assessing vulnerability severity.

All CVEs in our dataset are assigned CVSS severity scores, ranging from 0 to 10. In Figure 3.4, we depict the distribution of CVSS severity scores for these vulnerabilities, rounded to the nearest integer. These scores reflect the severity of the vulnerability, with 0 to 3.9 deemed low severity, 4.0 to 7.9 labeled medium, and 8.0 to 10.0 regarded as highly severe. We observe that the NVD data consists of vulnerabilities ranging across all severity scores. However, there is a substantial skew towards medium and high scores, which may be the visible effect of security researchers favoring reports of higher-value vulnerabilities (related to the limitations outlined in Section 3.2.6).

3.3.4 Vulnerability Categories

The Common Weakness Enumeration (CWE) is a standard for identifying the class of software weaknesses that resulted in a particular security issue [134]. The final NVD annotation we consider is the vulnerability's CWE identifiers, indicating the vulnerability categories. A CWE ID is assigned for 87% of CVEs in our dataset. In total, there are 45 unique CWE IDs associated with our vulnerabilities. Table 3.3 enumerates the most common software weaknesses, including frequent security problems such as buffer overflows and cross-site scripting errors. However, again we observe that our vulnerabilities span a wide variety of security issues.

3.3.5 Vulnerability Distribution over Repositories

Our selected CVE vulnerabilities were unevenly distributed over 682 Git projects, as visible in Figure 3.5. Our dataset contains one vulnerability for the majority of projects, and a heavy skew

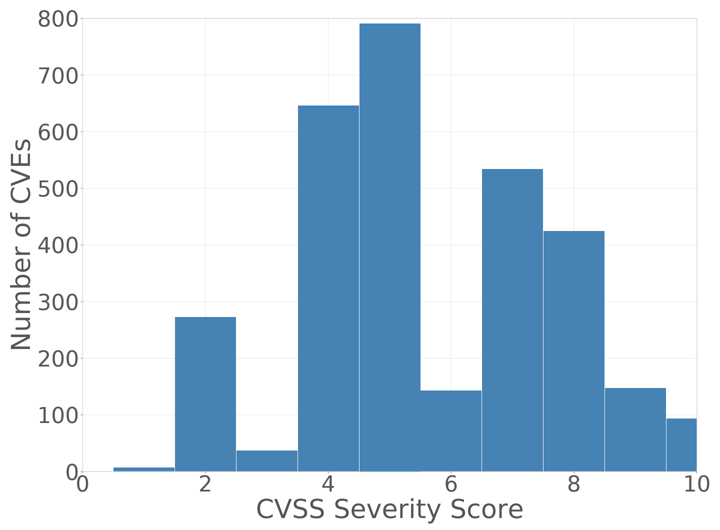


Figure 3.4: Distribution of CVSS severity scores, which are on a scale of 0 to 10, rounded to the nearest integer.

	CWE ID	Weakness Summary	Num. CVEs
1.	119	Buffer Overflow	539
2.	20	Improper Input Validation	390
3.	264	Access Control Error	318
4.	79	Cross-Site Scripting	273
5.	200	Information Disclosure	228
6.	189	Numeric Error	221
7.	399	Resource Management Error	219
8.	362	Race Condition	72
9.	89	SQL Injection	61
10.	310	Cryptographic Issues	42

Table 3.3: Top 10 CWE software weaknesses by the number of CVEs.

towards a smaller set of projects (e.g., the Linux kernel has over 900 CVE-related commits). Due to this skew, our analysis must consider per-repository averages, in addition to aggregates.

Figure 3.5 also illustrates the total number of commits in repository logs. We see that our repositories have varying levels of development, ranging from 3 commits for the "Authoring HTML" Drupal module to over 100,000 commits for projects such as the Linux kernel, LibreOffice, MySQL server, and the PHP interpreter.

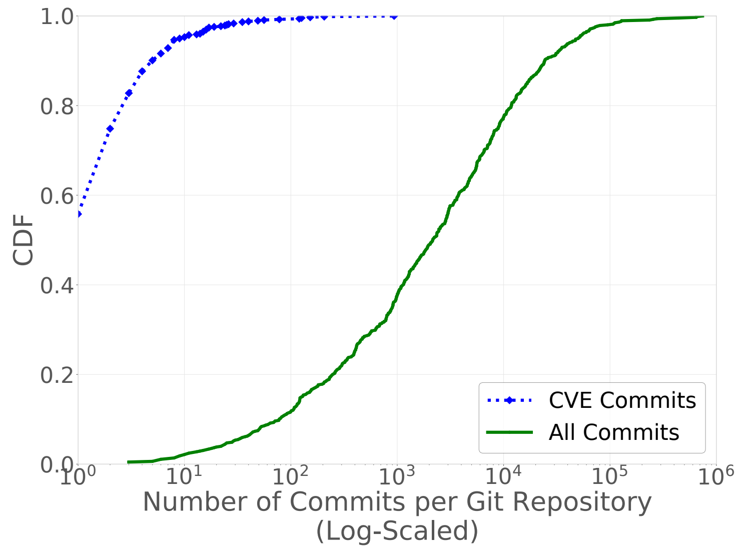


Figure 3.5: CDFs of the number of CVE commits and all commits for our collected Git repositories.

3.3.6 Repository Size

We can characterize a repository's size by the number of files it has, or the number of lines in those files. In Figure 3.6, we plot the CDFs of these metrics, for both the original repositories and their cleaned versions (as described in Section 3.2.4). Our selected projects vary widely in their sizes along both metrics. We find small projects affected by vulnerabilities, such as the SQL injection bug (CVE-2013-3524) in phpVMS's "PopUpNews" module, consisting of 4 PHP files with 103 lines of code. On the other extreme, the Linux kernel contains 20 million lines of code across 44,000 files.

3.4 Analysis Results

Our collected dataset consists of a diverse set of security vulnerabilities across numerous software projects, for which we have downloaded the source code repositories and amassed a set of both security and non-security bug fixes. The combination of the meta-data about patched vulnerabilities and the direct visibility into the corresponding source codes (as well as their history of changes) affords us with a unique perspective on the development life cycle of security fixes, as well as on the characteristics of the security patches themselves (in comparison to non-security bug fixes). In this section, we discuss our corresponding analysis and findings.

When exploring differences between two groups, we determine the statistical significance of our observations using permutation tests with 1,000 rounds. For each group we use a summary statistic of the area under the CDF curve for the investigated metric. In each round of a permutation test, we randomly reassign group labels to all data points (such that group sizes remain constant),

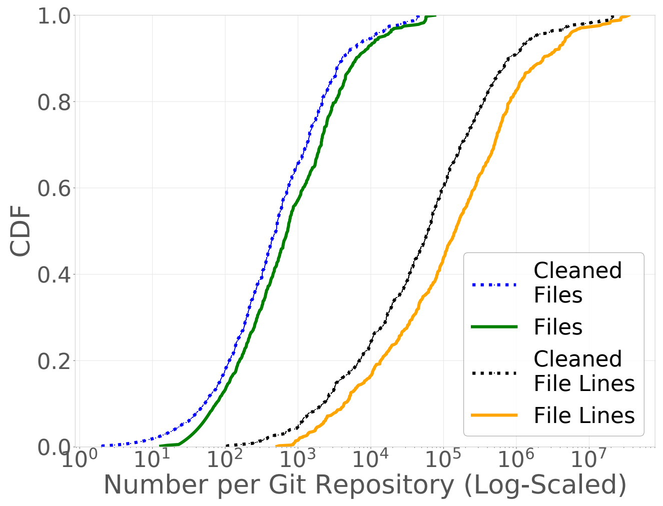


Figure 3.6: CDFs of the number of files and file lines in our collected repositories and their cleaned versions.

recompute the summary statistic for each group, and determine if the summary statistic difference between the newly formed groups exceeds that of the original groups. If the null hypothesis holds true and no significant difference exists between the groups, then the random permutation will only reflect stochastic fluctuations in the summary statistic difference. We assess the empirical probability distribution of this measure after the permutation rounds, allowing us to determine the probability (and significance) of our observed differences. We compute all of the reported p-values via this approach, and using a significance threshold of $\alpha = 0.05$.

3.4.1 Patch Development Life Cycle

From a software project's perspective, a vulnerability advances through several events throughout its life, such as its introduction into the code base, its discovery and the subsequent patch development, the public disclosure of the security issue, and the distribution of the fix. Prior studies have analyzed the vulnerability life cycle from a public perspective [74, 141, 175], observing when a vulnerability became disclosed to the public and when the corresponding patch was publicly distributed. However, these works have not delved into the project developer side of the remediation process and the life cycle of the patch development itself. Such an exploration can help illuminate the responsiveness of developers to patching vulnerabilities, how long fixes are available before they are actually distributed publicly, and how successfully developers resolve security issues. Here, we investigate the patch development process by connecting the vulnerability information available in the NVD with the historical logs available in Git repositories.

3.4.1.1 Vulnerability Life Spans in Code Bases

Upon a vulnerability's first discovery, we might naturally ask how long it plagued a code base before a developer rectified the issue. We call this duration the vulnerability's *code base life span*—a notion distinct from the vulnerability's *window of exposure* as investigated in prior work [74, 175], which measures the time from the first release of a vulnerable software version to the public distribution of its patch. As the development and distribution of a patch often occur at different times (a factor we explore in Section 3.4.1.2), the code base life span reflects the window of opportunity for attackers who silently discover a vulnerability to leverage it offensively, before any defensive measures are taken.

Reliably determining when a vulnerability was born in an automated fashion is difficult, as it requires semantic understanding of the source code and the nature of the vulnerability. However, we can approximate a *lower bound* on age by determining when the source code affected by a security fix was previously last modified. We note that this heuristic does assume that security fixes modify the same lines that contained insecure code, which may not always be the case. However, we assessed whether this is a robust approximation by randomly sampling 25 security patches. We observed that only 1 did not touch an originally insecure code region, enabling us to conclude that the vast majority of security fixes do modify the culprit code regions.

We analyzed the cleaned versions of security commit data to focus on source code changes. For all lines of code deleted or modified by a security commit, we used Git's "blame" functionality to retrieve the last time each line was previously updated (the blame date).[4] We conservatively designate the most recent blame date across all lines as the estimated date of vulnerability introduction. Then, the duration between this date and the commit date provides a lower bound on the vulnerability's code base life span.

How long do vulnerabilities live in code bases? Figure 3.7 illustrates the distribution of the lower bound estimates for vulnerability life spans. We plot the distribution for the aggregate of all CVEs, conservatively using the shortest life span for CVEs with multiple commits. To consider potential bias introduced by the uneven distribution of CVEs across repositories (discussed in Section 3.3.5), we also group commits by their repositories and plot the distributions of the minimum, median, and maximum life span per repository. The aggregate CVE distribution largely follows that of the per-repository median, although it exhibits skew towards longer life spans.

We observe that vulnerabilities exist in code bases for extensive durations. Looking at per-repository medians, we see that 50% had life spans exceeding 438 days (14.4 months). Furthermore, a quarter of repository medians and a third of all CVEs had life spans beyond three years. The longest surviving vulnerability was CVE-2015-8629 in the Kerberos 5 project, patched in January, 2016. The information disclosure vulnerability was first introduced over *21 years ago*.

We observe that 6.5% of our CVEs had a life span lower bound of less than 10 days. Manual inspection identified these as cases where our lower bound was overly conservative, as the vulner-

[4]Note that we cannot similarly process newly added lines, as they did not exist prior to the commit. We ignore the 22.8% of commits with only additions.

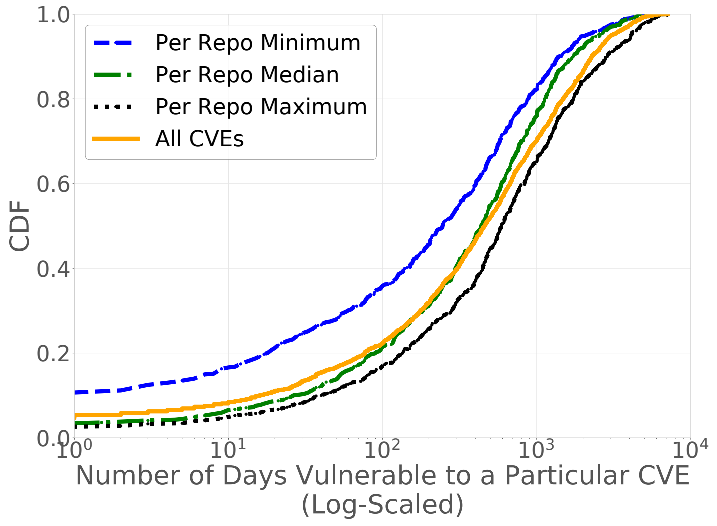


Figure 3.7: CDFs of CVE vulnerability life spans, for all CVEs and when grouped by software repositories.

ability was introduced at an earlier date. Recent commits happened to touch the same area of code involved in the security fix, resulting in our under-approximation.

Our results concur with prior findings that vulnerabilities live for years, generalized across numerous types of software. Manual evaluation of Ubuntu kernel vulnerabilities [103, 107] found that the average vulnerability's code base life span was approximately 5 years. Similarly, Ozment and Schechter [156] manually analyzed vulnerabilities in OpenBSD, finding the median vulnerability lifetime exceeded 2.6 years, although they noted that OpenBSD emphasizes secure coding practices. We observe that our typical life span estimates are lower than these previous ones, which may be due to our consideration of software projects beyond Linux variants, or our conservative approximation method.

Do more severe vulnerabilities have shorter lives? One might hypothesize that more severe vulnerabilities reside in code bases for shorter periods, as their more visible impact may correlate with more likely discovery and quicker remediation. To explore this aspect, we correlate CVSS severity scores with life spans, computing a Spearman's correlation coefficient of $\rho = -0.062$. This indicates that there is *no* substantial (monotonic) correlation between a vulnerability's severity and its life span. Even if developers are more motivated to remedy severe vulnerabilities, their expediency pales in comparison to the time scale of the initial vulnerability discovery, which our analysis shows is uncorrelated with severity. We note this generalizes an observation that Ubuntu vulnerability life spans likewise did not correlate with severity [107].

Do different types of vulnerabilities have varying life spans? Different classes of vulnerabilities may exhibit varying life spans, as some vulnerabilities might prove more challenging to

	Weakness Summary	Median Life Span
1.	SQL Injection	230.0
2.	Cross-Site Scripting	290.0
3.	Improper Input Validation	350.0
4.	Access Control Error	373.0
5.	Cryptographic Issues	456.0
6.	Resource Management Error	480.0
7.	Information Disclosure	516.5
8.	Race Condition	573.0
9.	Numeric Error	659.5
10.	Buffer Overflow	781.0

Table 3.4: Median vulnerability life span in days for the top 10 software weakness categories, as classified by CWE.

uncover. In Table 3.4, we summarize the vulnerability life spans for CVEs exhibiting the top 10 software weaknesses as classified by CWE (as discussed in Section 3.3.4). We observe that vulnerability life spans vary widely based on the software weakness class. Web-oriented vulnerabilities like SQL injection and cross-site scripting have significantly shorter life spans compared to errors in software logic and memory management. In comparison, race conditions, numeric errors, and buffer overflows remain undiscovered for two to three times as long. (Balancing across software repositories did not change the findings.) We conjecture that the life span variation across different vulnerability types results from both the type of software affected and the nature of the vulnerability. For example, web-oriented issues may appear on websites visited by thousands of users, increasing the likelihood that some problematic scenario arises that uncovers the vulnerability. Also, certain vulnerabilities such as cross-site scripting and SQL injection may be isolated to a small portion of code where reasoning about and identifying issues is more straightforward (compared to other problems such as race conditions).

3.4.1.2 Security Fix Timeliness

The timeliness of a security fix relative to the vulnerability's public disclosure affects the remediation process and the potential impact of the security issue. On the one hand, developers who learn of insecurities in their code base through unanticipated public announcements have to quickly react before the attackers leverage the information for exploitation. On the other hand, developers who learn of a security bug through private channels can address the issue before public disclosure, but the available patch may not be released for some time due to a project's release cycle, expanding the vulnerability's window of exposure.

We explore this facet of remediation by comparing the patch commit date for CVEs in our dataset with public disclosure dates (estimated as described in Section 3.2.5). We note that disclosures are not necessarily intertwined with patch releases, although this is the case for the majority of disclosures [74]. In Figure 3.8, we depict the CDFs of the number of days between disclosure

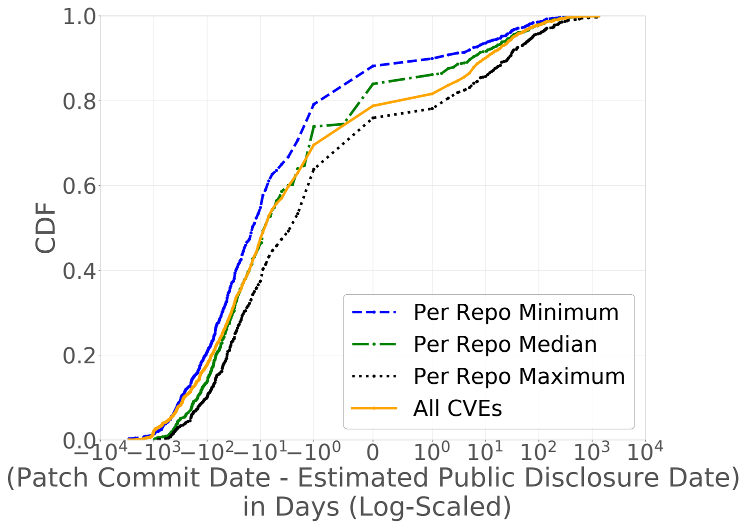


Figure 3.8: CDFs of the number of days between a vulnerability's public disclosure and its fix, plotted for all CVEs and grouped by software repositories.

and patching. We plot this for all CVEs, using the earliest patch commit date if a CVE has multiple commits associated with it. We additionally group CVEs by their software repositories, and plot the distribution across repositories. Here, we observe that the aggregate distribution over all CVEs largely matches the distribution over per-repository medians, although the per-repository medians exhibit a slight skew towards smaller absolute values.

How frequently are vulnerabilities unpatched when disclosed? In Figure 3.8, vulnerabilities publicly disclosed but not yet fixed manifest as positive time difference values. This occurred for 21.2% of all CVEs. We cannot determine whether these vulnerabilities were privately reported to project developers but with no prior action taken, or disclosed without any prior notice. However, over a quarter (26.4%) of these unpatched security issues remained unaddressed 30 days after disclosure, leaving a window wide open for attacker exploitation. This generalizes the observation made by Frei [73], who found that approximately 30% of Windows vulnerabilities were unpatched at disclosure and some remained so for over 180 days.

How frequently are vulnerabilities fixed by disclosure time? The predominant behavior in Figure 3.8, occurring for 78.8% of all CVEs, is that the security fixes were committed by public disclosure time, manifesting as negative or zero time differences. This suggests that the majority of vulnerabilities were either internally discovered or disclosed to project developers using private channels, the expected best practice.

Are vulnerability patches publicly visible long before disclosure? From Figure 3.8, we see that nearly 70% of patches were committed before disclosure (having negative time difference val-

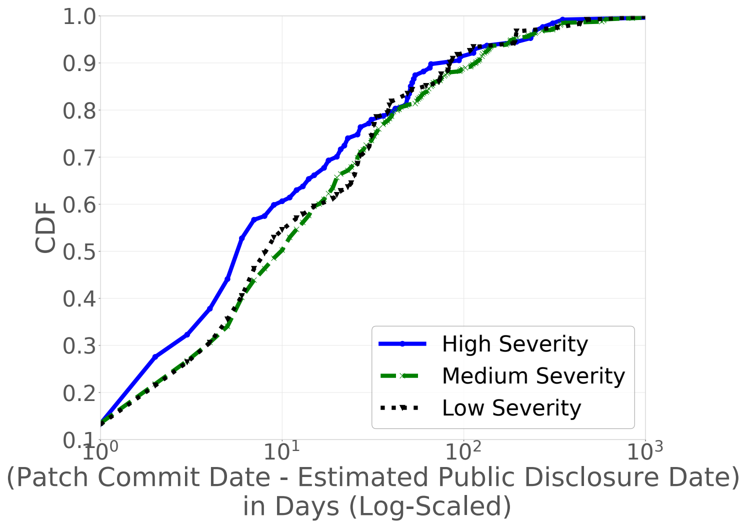


Figure 3.9: CDFs of the number of days after public disclosure until a CVE has a patch committed, grouped by the CVE severity class.

ues). The degree to which security commits precede disclosures varies widely, which upon manual inspection appears to arise due to the different release cycles followed by various projects (and variations within each project's development timeline). This behavior highlights the security impact of an interesting aspect of the open-source ecosystem. Open-source projects are not frequently in a position to actively distribute security updates. Rather, we observe that projects roll security fixes into periodic version releases that users must directly download and install, or updates are pulled downstream for incorporation by software distribution platforms (such as package repositories maintained by Linux OS variants). Announcements about the releases or updates, and the security fixes they contain, follow shortly after.

Unfortunately, this development and deployment process also provides a window of opportunity for exploitation. Given the public nature of open-source projects and their development, an attacker targeting a specific software project can feasibly track security patches and the vulnerabilities they address. While the vulnerability is addressed in the project repository, it is unlikely to be widely fixed in the wild before public disclosures and upgrade distribution. From Figure 3.8, we note that over 50% of CVEs were patched more than a week before public disclosure, giving attackers ample time to develop and deploy exploits.

Are higher severity vulnerabilities patched quicker? All vulnerabilities are not equal, as they vary in exploitation complexity and requirements, as well as security impact. One might expect these factors to affect the patch development process, as developers may prioritize fixing certain vulnerabilities over others. To explore whether a vulnerability's severity (scored using CVSS) affects patch timeliness behavior, we cluster CVEs by their severity categories (low, medium, and high). We find that severity significantly affects whether a fix is developed before or after public disclosure. 88.1% of high severity CVEs were patched prior to public announcements, compared to 78.2% of medium severity bugs and 58.8% of low severity issues. These differences indicate that project developers prioritize higher impact vulnerabilities when determining if and when to address them.

While one might also expect earlier disclosures for more severe vulnerabilities, we observe no significant differences ($p > 0.12$) across severity categories when investigating the time by which a patch precedes disclosure (for vulnerabilities fixed by disclosure time). This fits with the common model used by many open-source projects of rolling security patches (of all severity levels) into recurrent releases and announcements. When exploring the time after disclosure until patching (for vulnerabilities unpatched at disclosure), we find that highly severe vulnerabilities get patched more quickly, as shown in Figure 3.9. This difference is significant ($p < 0.013$), indicating project developers respond quicker to more serious disclosed-yet-unpatched vulnerabilities.

3.4.1.3 Patch Reliability

The patch a developer creates to address a vulnerability may unfortunately disrupt existing code functionality or introduce new errors. Beyond the direct problems that arise from such patches, end user trust in generally applying patches (or in the software itself) can erode, as we will examine from discussions with system administrators in Chapter 4. To assess how successful developers are at producing reliable and safe security fixes, we attempted to identify instances of multiple commits for the same CVE, and classify the causes.

How frequently are security patches broken (e.g., incomplete or regressive)? In total, 11.5% of CVEs had multiple associated commits for a single repository in the NVD data. However, if an initial patch introduced an error or was incomplete, the NVD entry might not have been updated with the follow-on fix. After the NVD entry is published, NVD analysts are unlikely to continue tracking a CVE unless new updates are reported to them. Thus, we attempted to identify further commits that may be associated with a CVE using repository Git logs.

For each security patch commit and its commit hash H, we searched the repository's Git log for any other commits that had a commit message including the CVE ID or the 7-character prefix of the commit hash. We considered this prefix as it is used as the Git short version of the commit hash, and matches any longer hash prefixes. This method finds related commits which were not distinct patches, such as merge, rebase, and cherry-pick commits. To filter these, we ignored commits with diffs identical to an earlier one, and commits with multiple parents (e.g., merges). Note that we could only identify multiple patches when commit messages contained this explicit linkage, so our analysis provides a lower bound.

Using this approach, we identified a total of 682 CVEs with multiple commits, 22.0% of all CVEs. Not all multi-commit fixes are necessarily problematic though, as project developers may split a fix into multiple commits that they push to the repository in close succession. We observed that 242 CVEs had all fixes committed within 24 hours. Given the limited time window for potential newly introduced problems, we designate these as *piecewise* fixes and non-problematic.

CVE Commits Label	#. CVEs	Median Num. Follow-On Commits	Median Fix Inter-Arrival Time
Incomplete	26 (52%)	1.0	181.5 Days
Regressive	17 (34%)	1.0	33.0 Days
Benign	14 (28%)	1.5	118.5 Days

Table 3.5: Summary of our manual investigation into 50 randomly sampled CVEs with multiple commits. Note that a CVE may have commits in multiple categories. Follow-on commits include all commits associated with the original patch.

We randomly sampled 50 of the remaining 440 CVEs and manually investigated if the fixes were problematic. Table 3.5 summarizes our results. We identified 26 (52%) as *incomplete*, where the initial fix did not fully patch the vulnerability, requiring a later patch to complete the job. We labeled 17 (34%) as *regressive*, as they introduced new errors that required a later commit to address. The overlap included 4 CVEs (8%) with both incomplete and regressive patches. Other follow-on fixes were *benign*, such as commits for added documentation, testing, or code cleanup/refactoring. 11 CVEs (22%) had only these benign additional commits (although 3 other CVEs had both benign and problematic commits). Note that our random sample was not biased towards any particular project, as it spanned 42 repositories.

Extrapolating from the random sample to the remaining 440 CVEs with non-piecewise multiple commits (accounting for 14.2% of all CVEs), we estimate that about 7% of *all* security fixes may be incomplete, and about 5% regressive. These findings indicate that broken patches occur unfortunately frequently, and applying security patches comes with non-negligible risks. In addition, these numbers have a skew towards underestimation: we may not have identified all existing problematic patches, and recent patches in our dataset might not have had enough time yet to manifest as ultimately requiring multiple commits.

We note that our observed frequency of failed security fixes is similar to or lower than that observed by prior studies on general bug fixes. Gu et al. [92] observed that 9% of bug fixes were bad across three Java projects while Yin et al. [207] found that between 15%–25% of general bug patches for several Linux variants were problematic. As our detection of problematic security fixes skews towards underestimation, it is undetermined whether security fixes are more or less risky than other bug fixes. However, it is clear that security patches do suffer failures similarly to non-security bug fixes.

How long do problematic patches remain unresolved? As shown in Table 3.5, for both incomplete and regressive patches in our sample, we find the median number of additional patches required to rectify the original broken patches to be only one commit. The typical incomplete fix takes *half a year* to remedy, and patches problematic enough to require reverting typically take a month to repair. Thus, problematic security patches can remain unresolved for extensive durations of time.

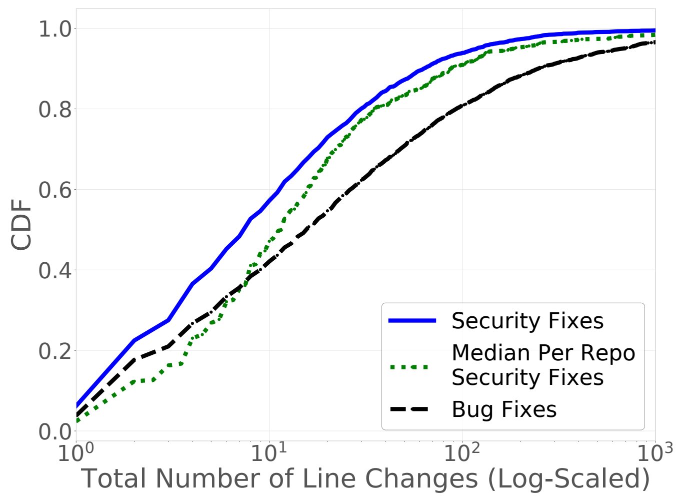


Figure 3.10: CDFs of the total number of line changes, for all security and non-security bug fixes, and the median of security commits grouped by repository.

3.4.2 Patch Characteristics

While numerous works have investigated general software patches [158, 176, 178, 212], few have considered what distinguishes security patches from other non-security bug fixes. Intuitively, the software conditions resulting, for example, in buffer overflow and SQL injection vulnerabilities can differ greatly from those that produce performance and concurrency bugs. Thus, the characteristics of their fixes may likewise prove different. Indeed, Zama et al. [208] conducted a case study on security and performance fixes for Mozilla Firefox, observing differences in the remediation for the two bug types. These characteristics are important to understand as they may reflect our ability to expeditiously generate patches, verify their safety, or assess their impact on applications. Here, we compare our collection of security and non-security bug fixes to help illuminate their differences, considering facets such as the complexity of fixes and the locality of changes.

3.4.2.1 Non-Source Code Changes

Do security and non-security bug fixes always modify source code? Given the nature of bug fixes, one might expect them to universally involve source code changes. We explore this hypothesis by contrasting our commit data with their cleaned versions (source code comments and non-source code files removed). We find that the hypothesis does not hold: a non-trivial fraction of commits involved no code changes. For non-security bug fixes, 6.1% involved erroneous configurations, build scripts with broken dependencies or settings, incorrect documentation, and other non-source code changes.

More surprising, we find that 1.3% of security fixes also did not touch source code. In some cases, the commit added a patch file to the repository without applying the patch to the code base. However, numerous CVE vulnerabilities *do not* reside in the source code. For example, CVE-2016-7420 was assigned to the Crypto++ Library for not documenting the security-related compile-time requirements, such that default production builds may suffer information disclosure vulnerabilities. Similarly, the fix for CVE-2016-3693 involved changing a project library dependency to a new version, as the inclusion of the older versions allowed attackers to access sensitive system information.

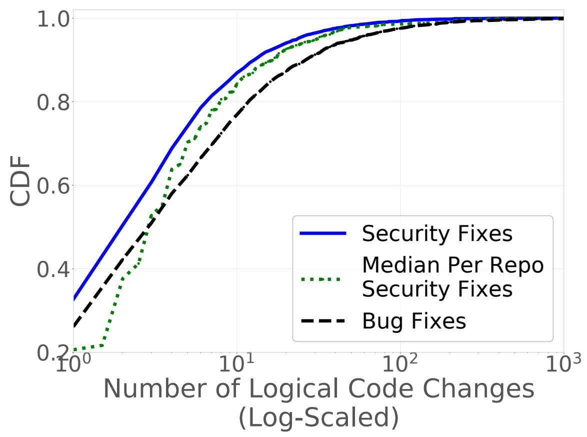
Thus, bug fixes are not exclusively associated with source code modifications, although this is significantly more likely with non-security bug fixes than with security patches. For further analysis on commit characteristics, we focus on the cleaned versions, excluding the commits that did not modify code.

3.4.2.2 Patch Complexity

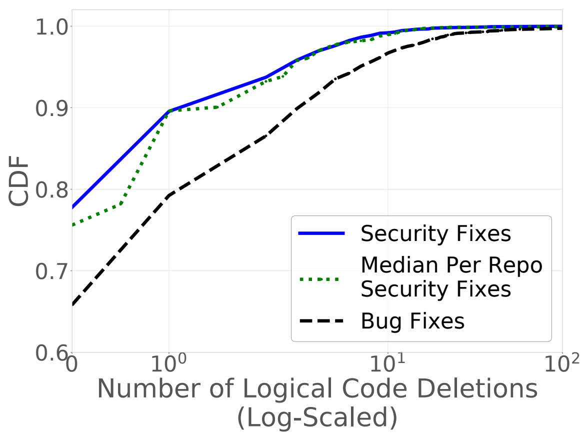
How complex are security patches compared to other non-security bug fixes? We can assess software complexity using various metrics, although some, such as cyclomatic complexity [127], require deep analysis of a code base. Given the number and diversity of software projects we consider, we chose lines of code (LOC) as a simple-albeit-rudimentary metric, as done in prior studies [97, 140, 176, 208].

Are security patches smaller than non-security bug fixes? In Figure 3.10, we plot the CDFs of the total LOC changed in cleaned commit diffs, for all security and non-security patches, as well as the median security fix per repository. This conservative metric sums the LOC deleted or modified in the pre-commit code with those modified or added post commit, providing an upper bound on the degree of change. We see that compared to per-repository medians, the aggregate of security commits skews towards fewer total LOC changed. Under this metric, security commits overall are statistically significantly less complex and smaller than non-security bug patches ($p \approx 0$). The median security commit diff involved 7 LOC compared to 16 LOC for non-security bug fixes. Approximately 20% of non-security patches had diffs with over 100 lines changed, while this occurred in only 6% of security commits. When considering per-repository medians, our conclusions differ only slightly, in that non-security bug fixes have a slightly larger portion of very small commits with diffs less than 9 LOC, but are typically larger.

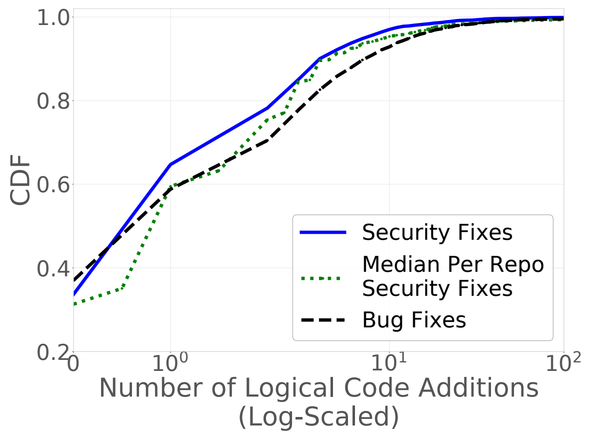
Do security patches make fewer "logical" changes than non-security bug fixes? As an alternative to our raw LOC metric, we can group consecutive lines changed by a commit as a single "logical" change. Under this definition, several lines updated are considered a single logical update, and a chunk of deleted code counts as a single logical delete. We depict the CDFs of the number of logical actions per commit in Figure 3.11, although we omit a plot for logical updates as it closely resembles that of all logical changes. In all cases, we observe that per-repository medians skew less towards very small numbers of logical actions compared to security commits in aggregate. Across all logical actions, we observe that security commits involve significantly fewer changes (all $p < 0.01$). Nearly 78% of security commits did not delete any code, compared to 66%



(a) All Logical Changes



(b) Logical Deletes



(c) Logical Additions

Figure 3.11: CDFs of the number of logical changes introduced by a commit, for all security and non-security bug fixes, and for the median amongst security commits grouped by repository. We omit a plot for logical updates, which looks very similar to that for all logical changes because logical updates predominate. Note the varying minimum y-axis values.

of non-security bug-fix commits. Between 30% to 40% of all commits also did not add any code, indicating the majority of logical changes were updates.

Do security patches change code base sizes less than non-security bug fixes? Another metric for a patch's complexity is its impact on the code base size. The net number of lines changed by a commit reflects the growth or decline in the associated code base's size. In Figure 3.12, we plot the CDFs of these size changes. We observe that significantly more non-security bug patches result in a net reduction in project LOC, compared to security fixes: 18% of non-security bug fixes reduced code base sizes compared to 9% of security patches. For all commits, ap-

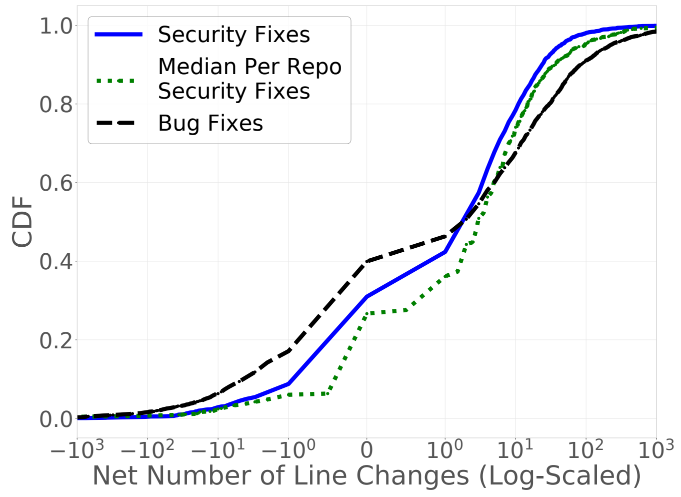


Figure 3.12: CDFs of the net number of line changes, for all security and non-security patches, and the median of security commits grouped by repository.

proximately a quarter resulted in no net change in project LOC, which commonly occurs when lines are only updated. Overall, projects are more likely to grow in size with commits, as the majority of all commits added to the code base. However, security commits tend to contribute less growth compared to non-security bug fixes, an observation that accords with our earlier results.

These findings support the notion that security fixes are generally less complex than other bug fixes. We note that this generalizes the same conclusion drawn for Mozilla Firefox by Zama et al. [208].

3.4.2.3 Commit Locality

Finally, we can quantify the impact of a patch by its *locality*. We consider two metrics: the number of files affected and the number of functions affected.

Do security patches affect fewer source code files than non-security bug fixes? Figure 3.13 illustrates the CDFs of the number of files touched by fixes. From this, we see that security patches modify fewer files compared to non-security bug fixes, a statistically significant observation ($p \approx 0$). In aggregate, 70% of security patches affected one file, while 55% of non-security bug patches were equivalently localized. Fixes typically updated, rather than created or deleted, files. Only 4% of security fixes created new files (vs. 13% of non-security bug fixes), and only 0.5% of security patches deleted a file (vs. 4% of non-security bug fixes).

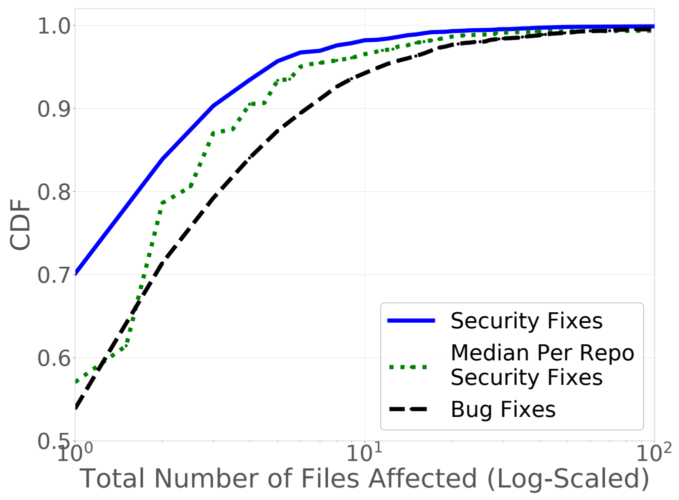


Figure 3.13: CDFs of the number of files affected, for all security and non-security bug fixes, and the median of security fixes grouped by repository.

Do security patches affect fewer functions than non-security bug fixes? To pinpoint the functions altered by patches, we used the `ctags` utility [11] to identify the start of functions in our source code. We determined the end of each function under the scoping rules of the corresponding programming language, and mapped line changes in our commit diffs to the functions they transformed. Figure 3.14 shows the CDFs of the number of functions affected by patches. We find that 5% of non-security bug fixes affected only global code outside of function boundaries, compared to 1% of security patches. Overall, we observe a similar trend as with the number of affected files. Security patches are significantly ($p \approx 0$) more localized across functions: 59% of security changes resided in a single function, compared to 42% of other bug fixes.

In summary, our metrics indicate that security fixes are more localized in their changes than other bug fixes.

3.5 Discussion

In this chapter, we have conducted a large-scale empirical analysis of security patches across over 650 projects. Here we discuss the main takeaways, highlighting the primary results developed and their implications for the security community moving forward.

Need for more extensive or effective code testing and auditing processes for open-source projects: Our results show that vulnerabilities live for years and their patches are sometimes problematic. Using a lower bound estimation method, our exploration of vulnerability life spans

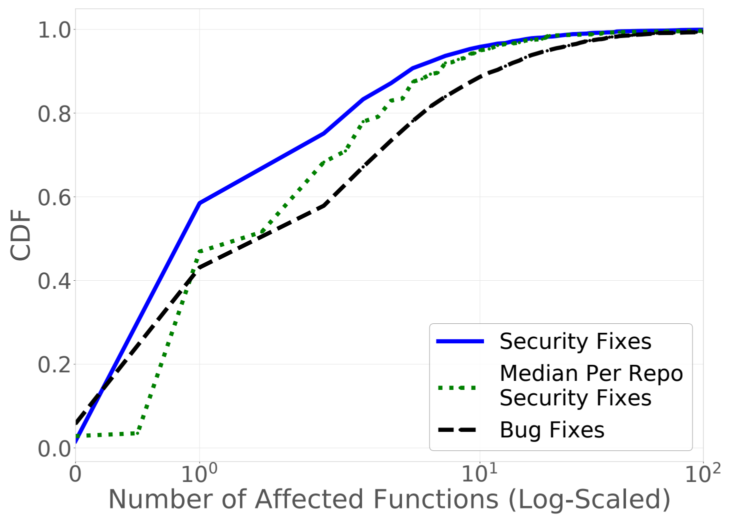


Figure 3.14: CDFs of the number of functions modified, for all security and non-security bug patches, and the median of security fixes grouped by repository.

revealed that over a third of all security issues were first introduced more than 3 years prior to remediation. The issues do not cease once a patch is created; almost 5% of security patches negatively impacted the software, and over 7% did not completely remedy the security hole.

These findings indicate that the software development and testing process, at least for open-source projects, is not adequate at quickly detecting and properly addressing security issues. There are several important implications due to these shortcomings. An attacker who discovers a zero-day vulnerability can retain its viability with reasonable confidence for on the order of years. While large-scale exploitation of a zero-day may result in its detection and subsequent remediation, targeted attacks may persist unnoticed. Similarly, a subtle backdoor inserted into a code base will also likely survive for a prolonged period, with only commit code reviews (if performed) as the final barrier. The not infrequent occurrences of broken security patches also have negative implications on user patching behavior. Applying a patch has often been viewed as risky, and negative experiences with problematic updates (particularly regressive ones) can drive users away from remedying security bugs in a timely fashion (as we will see regarding system administrator updating in Chapter 4).

A natural avenue for future work is to develop more effective testing processes, particularly considering usability, as developers are unlikely to leverage methods that prove difficult to deploy or challenging to interpret. One example of such research is VCCFinder [160], a code analysis tool that assists with finding vulnerability-introducing commits in open-source projects. In addition, software developers can already make strides in improving their testing processes by using existing tools more extensively. For example, sanitizers such as ASan [78], TSan [78] and UBSan [27] help detect various errors that may result in security bugs. Fuzzers (such as AFL [2]) also assist in identifying inputs that trigger potentially exploitable issues.

The transparency of open-source projects makes them ripe for such testing not only by the developers, but by external researchers and auditors as well. Community-driven initiatives, such as those supported by the Core Infrastructure Initiative [10], have already demonstrated that they can significantly improve the security of open-source software. For example, the Open Crypto Audit Project [20] audited the popular encryption software TrueCrypt, while Google's OSS-Fuzz program [85] offers continuous fuzzing of critical open-source infrastructure for free, already discovering and reporting hundreds of bugs. Further support of such efforts, and more engagement between various project contributors and external researchers, can help better secure the open-source ecosystem.

Need for refined bug reporting and public disclosure processes for open-source projects: Our analysis of the timeliness of security fixes revealed that they are poorly timed with vulnerability public disclosures. Over 20% of CVEs were unpatched when they were first announced, perhaps sometimes to the surprise of project developers. While we observed that these were more likely to be low-severity vulnerabilities, many were still medium- and high-severity bugs, unfixed for days to weeks post-disclosure. This gap provides attackers with the knowledge and time to strike.

In the opposite direction, we discovered that when security issues are reported (or discovered) privately and fixed, the remedy is not immediately distributed and divulged, likely due to software release cycles. Over a third of fixed vulnerabilities were not publicly disclosed for more than a month. While operating in silence may help limit to a small degree the dissemination of information about the vulnerability, it also forestalls informing affected parties and spurring them to remediate. Given the transparency of open-source projects, attackers may be able to leverage this behavior by tracking the security commits of target software projects (perhaps by training a classifier or keying in on common security-related terms in commit messages). From the public visibility into these commits, attackers can identify and weaponize the underlying vulnerabilities.

However, the open-source nature of projects need not be a liability when patching vulnerabilities. Transparent bug reporting instructions, containing the proper point of contact, the required diagnostic information, the expected remediation timeline, and potential incentives (such as bug bounties or "hall of fame" status), can expedite the vulnerability reporting process. Fixes for vulnerabilities can also be disclosed in better coordination with public disclosures. For example, the Internet Systems Consortium (ISC), maintainer of the open-source DNS software BIND and DHCP implementations, has established explicit disclosure policies that embargo publicly revealing security patches until near public disclosure time [18]. Instead, ISC customers, OEMs, operating system maintainers, and other vendors who re-package ISC open-source software are privately notified about vulnerabilities and their patches prior to public disclosure. A controlled disclosure process informs some of the most heavily affected parties before public disclosure, providing adequate time to prepare properly, while reducing the leakage of vulnerability information pre-disclosure. Additionally, Internet-scale outreach efforts to notify end-systems affected by a vulnerability could spur expedient remediation. We explore this topic in Part II of this dissertation.

Opportunities for leveraging characteristics of security patches: Our comparison of security patches with non-security bug fixes revealed that security fixes have a smaller impact on code bases, along various metrics. They involve fewer lines of code, fewer logical changes, and are more localized in their changes. This has implications along various patch analysis dimensions.

Tying back to broken patches, the lower complexity of security patches can perhaps be leveraged for safety analysis customized for evaluating just security fixes. Also, as these remedies involve fewer changes, automatic patching systems may operate more successfully if targeting security bugs. Zhong and Su [212] observed that general patches are frequently too complex or too delocalized to be amenable to automatic generation. However, security patches may be small and localized enough. From a usability angle, we may additionally be able to better inform end users of the potential impact of a security update, given its smaller and more localized changes. The need for more exploration into the verification and automated generation of security patches is quite salient as our ability to respond to security concerns has remained relatively unchanged, while the attack landscape has grown ever more dangerous.

3.6 Conclusion

In this chapter, we conducted a large-scale empirical study of security patches, evaluating over 4,000 security fixes across a diverse set of 682 software projects. The investigation centered around a dataset we collected that merges vulnerability entries from the NVD, information scraped from relevant external references, affected source code repositories, and their associated security fixes. Using these disparate data sources, we analyzed facets of the patch development life cycle. In addition, we extensively characterized the security patches themselves, contrasting them with non-security bug fixes.

Our findings have revealed shortcomings in our ability to quickly identify vulnerabilities and reliably address them. Additionally, we have observed that the timing of public disclosure does not closely align with the date a patch is applied to the code base, providing windows of opportunity for attacker exploitation. Our characterization of security fixes shows they are less complex and more localized than other non-security bug fixes, perhaps making them more amenable to software analysis and automatic repair techniques. By leveraging these insights, we hope the security community can progress in improving the remediation process for security vulnerabilities.

Chapter 4

Examining How System Administrators Manage Software Updates

4.1 Introduction

System administrators serve as "keepers of the machines," entrusted by organizations to oversee their computers, many of which are vital to an organization's operations. Their duties include regularly applying software updates in a timely manner to ensure organizational safety against crippling attacks. Failure to patch known vulnerabilities can lead to devastating consequences [53] such as the colossal 2017 Equifax data breach which exposed sensitive personal data on over *140 million* individuals [146].

While prior studies have investigated how end users deal with software updates [69, 71, 101, 123–125, 141, 150, 193, 194, 200, 201], there has been less attention on system administrators, whose technical sophistication and unique responsibilities distinguish them from end users. Industry reports and guides on administrator patching exist (e.g., Sysadmin 101 [166]), but these lack peer-review and transparent, rigorous methods. Prior academic work on system administrators is often dated and focuses on aspects of administrator operations other than updating (e.g., on general tools used [38]) or specific technical (rather than user) updating aspects. Given the critical role that system administrators play in protecting an organization's machines, it behooves us to better understand how they manage updates and identify avenues for improved update processes. Therefore, in this chapter, we set out to answer two primary research questions: (1) what processes do system administrators follow for managing updates, and (2) how do administrator actions impact how effectively they perform system updates. To answer these questions, we surveyed 102 administrators and conducted semi-structured interviews with 17 of them. Note that here we study administrator software updating in general, beyond just focusing on security-related updates. We do so because many software updates bundle security patches with other changes, and administrators may not explicitly distinguish between different update types.

Our study determined that system administrators proceed through software updates through five main stages: (1) learning about updates from information sources, (2) deciding to update based on update characteristics, (3) preparing for update installation, (4) deploying an update, and finally (5) handling post-deployment update issues that may arise. By analyzing the factors that system administrators consider and the actions that they perform, we identified challenges that they encounter and limitations of existing procedures at each stage of the updating process. We observed problems with comprehensively obtaining relevant information about available updates, effectively testing and deploying updates in a timely fashion, and recovering from update-induced errors. We also witnessed how organizational and management influences, such as through policies and decisions, can impact the administrator's ability to handle updates effectively at multiple stages, sometimes for better, sometimes for worse. In addition, we note that while high-level aspects of software update workflows for system administrators mirror those of end users [124, 194], we found that the particular factors considered and the actions taken by system administrators are significantly different across all stages of the update process. This difference highlights the value of specifically studying the administrator population.

Our evidence-based study extends the research literature on updating practices to system administrators, a unique population. In particular, our work makes two primary contributions: first, we provide empirical grounding on how administrators update multiple machines for their organizations, examining the consequences of their actions at depths beyond prior explorations [55]. This evidence includes insights into how their actions impact how effectively they perform software updates to better secure their systems. Second, we make data-driven recommendations for improving administrator update processes through better systems for managing updates, better designed updates, and a shift in organizational policies.

The study in this chapter appeared at the USENIX Symposium on Usable Privacy and Security (SOUPS) [115], receiving the Distinguished Paper Award.

4.2 Method

To investigate how system administrators manage updates at scale, we conducted a qualitative study of current administrators responsible for managing updates in their organizations. We studied administrator software updating in general, beyond only considering security-related updates, as many software updates bundle security patches with other changes, and administrators might not differentiate between different update types. Our study proceeded in two phases. In phase one, we administered a large-scale survey of administrator updating practices, whose design was informed by pilot interviews. In phase two, we conducted semi-structured interviews with administrators. We specifically sought participants who had been working at an organization with five employees or more for a period of at least one year, to ensure they had job familiarity. We restricted participation to those over 18 years old residing in the United States (US). Both study phases received Institutional Review Board (IRB) approval. Our survey and interview questions are listed in Appendix A.

4.2.1 Preliminary Phase: Pilot Interviews

In Fall 2015-Spring 2016, to inform the design of our large-scale study, we recruited seven system administrators to participate in semi-structured pilot interviews about software updates. The interview questions were developed based on prior studies on software updating [124,194] and previous knowledge about the software update development and management process (see Appendix A.1 for details). We recruited participants via institutional mailing lists and social media, filtering for those who explicitly dealt with software updates. All interviews were conducted over the phone via Skype and recorded. The interviews lasted between 30-50 minutes. Participants were also asked to fill out a background survey that contained general questions about demographics, the type of software or programming languages used, the types of updates they handled, and any positive and negative aspects of their job responsibilities. Participants were compensated with $20 gift cards and a chance to win a hard drive.

Demographics: All seven participants were male and lived in the US. They were predominantly 20–40 years of age and only one participant did not have a bachelor's degree. The majority of participants had 1–10 years of work experience as a system administrator.

Analysis: We transcribed all pilot interviews and three coders used inductive thematic analysis [174] to derive the following over-arching themes in administrator update management: finding information about available software updates, testing and preparing for updates, deploying updates, and monitoring for update-triggered issues post-deployment. We used these themes to design questions for our study's two phases.

4.2.2 Phase One: Survey

Based on the pilot interviews, we constructed a survey asking about a participant's organization and responsibilities (e.g., size of organization, number of machines managed), how they manage the security of their systems, how they handle each stage of the update management process, and what works well and poorly for them (see Appendix A.2 for details). The survey consisted of 41 questions and took approximately 15 minutes to complete. We recruited system administrators in September and October of 2017 using social media, blogs associated with our research labs, and Reddit [22]. In addition, we recruited administrators attending the 2017 Large Installation System Administration Conference (LISA) by distributing fliers about our survey and providing a computer at the venue where administrators could complete the survey. As an incentive, we entered administrators who participated into a drawing for a Samsung S8 phone. In total, 102 system administrators completed the entire survey. We note that we recruited 22/102 survey participants at the LISA conference and the rest from online.

Data Analysis Method: The survey consisted of multiple-choice and open-ended questions. We focused our analysis on questions pertaining to software updating, as our survey also contained several less relevant questions on other security practices. We analyzed open-ended questions using open coding, identifying themes in the question responses [202]. Two researchers independently developed a set of codes across all questions and met to converge on a final codebook. Then, each researcher independently coded all question responses using that codebook. We had 199 codes with 611 coded segments in total, discussing themes of interest such as “Testing”, “Update Issues”, “Addressing Update Issues”, “What Works Well”, and “What is Challenging”. We use Kupper-Hafner inter-rater agreement scores [111] to quantify the consistency of the coding, finding an average agreement of 0.83, indicative of largely consistent coding. The survey coders met and converged upon the final codes for all open-ended question responses.

4.2.3 Phase Two: Semi-Structured Interviews

Using the themes identified by our pilot interviews, we developed a guide for conducting semi-structured interviews with system administrators. The guide contained questions about a participant’s demographics and job, and their update management process (see Appendix A.3 for details). Throughout Fall 2017, we recruited 17 interview subjects through the same channels as with the survey. All but one of our subjects participated in the survey as well. Interviews ranged from 1 to 3 hours long, were conducted in person or over Skype, and were recorded. We compensated participants with a $20 Amazon gift card.

Data Analysis Method: Using transcriptions of the recorded interviews, we developed a codebook for the responses through regular peer review meetings, based on the themes of interest for the interviews such as “Job Responsibilities”, “Update Importance”, and the various update stages, including “Seeking Update Information”, “Deployment”, “Testing”, and “Update Issues”. The codes were initially created by one team member and refined by group discussions and consensus [202]. Two coders independently coded the interview responses using the resulting codebook using inductive thematic analysis [174]. We had 347 codes with 1447 coded segments in total. Calculating inter-rater reliability for such qualitative coding of non-survey data has been shown to be difficult because of the nature of assigning multiple codes to data and inherent biases of coders [36]. For completeness, however, we randomly sampled 6/17 transcripts and computed an average agreement percentage between the two independent coders of 0.77, indicating high consistency. We discussed points of disagreement and ensured that the resulting themes discussed in this chapter were in line with both team members’ interpretations of the data.

4.2.4 Participant Demographics

Here we present the demographics of the 102 survey respondents and 17 interview subjects.

4.2.4.1 Respondent Characteristics

The population was male-dominated; only 6/102 survey and 2/17 interview subjects were female. The most common age bracket was 26-35 years old, containing 43/102 survey participants and 8/17 interview subjects. Other common age brackets were 36-45 years old (24 survey and 4 interview participants), and 46-55 years old (14 survey and 2 interview subjects). Most administrators had some higher education; 57/102 survey and 10/17 interview participants had a bachelor’s degree while 37 survey and 5 interview participants had some college education but no degree. Salaries varied widely, evenly distributed primarily between $35,000 to $150,000 (accounting for 93/102 survey and 14/17 interview participants). Survey respondents had a median of 11 years of experience, ranging from 1 to 35 years. In contrast, interview subjects had a lower median experience of 6 years, although the range was similar (1-34 years).

4.2.4.2 Organization Characteristics

About half of our study participants (56/102 in the surveys and 8/17 from the interviews) worked at larger organizations with over 500 employees. In comparison, only 13/102 survey and 2/17 interview participants worked for small organizations with fewer than 50 employees. In total, 22 survey respondents did not indicate the number of hosts they managed (all interview subjects did provide a response). However, the remaining typically oversaw many machines: only 12/102 survey and 3/17 interview participants maintained fewer than 100 hosts, while 36 survey and 8 interview subjects indicated they administered between 100-499 machines and 22 survey and 5 interview participants said they handled over 1000 machines. Servers were the most common type of machine managed, handled by 96/102 survey respondents. Over half of the administrators also dealt with desktops (63), routers (60), and laptops (57). Our participants maintained primarily Linux (73) and Windows (71) machines, and less so Macs (44).

4.2.5 Limitations

Studying system administrators is challenging as they are a specialized population that is difficult to recruit compared to end users. Thus, our study's approach may have limitations.

1. As administrators are often paid well, our study's participation compensation may not have influenced their decisions to contribute. Instead, those more ideologically motivated may have donated their time.
2. Due to our recruitment method, our study participants may not be representative of system administrators in general. For example, we only studied individuals from the US, so our findings may not apply globally. Similarly, we only recruited administrators fully employed by an organization, which does not capture those working part-time or as contractors.
3. Our results reflect our study's sample, which skewed towards certain demographics (e.g., males). Similarly, we recruited many of our participants via Reddit and the LISA conference. These subpopulations may exhibit certain skewed characteristics. For example, those attending the LISA conference may operate with a larger budget (covering conference expenses).
4. Our surveys and interviews contained open-ended questions. During our analysis, we provide the number of study subjects who gave a particular response to these open-ended questions (and indicate when results are obtained from such questions). However, we caution that such counts are not necessarily reliable indicators of real-world prevalence. In particular, we cannot assume a respondent does not act a certain way just because they do not mention such behavior, as they may have simply focused on alternative discussion topics.
5. Our study is an exploratory one that focuses on the processes system administrators use to manage software updates. However, we did not investigate all updating aspects in depth. For

example, we did not explicitly solicit recommendations from our study participants on how to improve updating tools and methods, nor did we tease apart the differences in updating between different types of updates, organizations, or machines. Moving forward, our study can help inform the design of broader quantitative explorations of these updating dimensions at scale.

4.3 Overview of Findings

From the responses to our system administrator surveys and interviews on general software updating, we determined that administrator update workflows consisted of five primary stages. These five stages, as illustrated in Figure 4.1, are: (1) learning about updates from information sources, (2) deciding to update based on update characteristics, (3) preparing for update installation, (4) deploying the update, and finally (5) handling post-deployment update issues that may arise. For each stage, our analysis determined the factors that system administrators considered and the actions that they conducted (also listed in Figure 4.1). This data affords us insights into the challenges that administrators encountered when updating and limitations of existing procedures. In Section 4.10, we discuss recommendations for improving administrator update processes grounded in our findings. We also compare how update workflows differ for system administrators versus end users in Section 4.10.1, identifying significant differences.

In the following sections on update stages, we explore how system administrators proceed through each stage and the security implications of their behaviors. Throughout the results, we designate quotes from survey respondents with *S* and interview participants with *P*.

4.4 Stage 1: Learning About Updates

In both our surveys and interviews, participants reported that—before deploying software updates—they first had to learn about available updates and then make decisions about which updates to handle. We note that while automatically initiated updates circumvent the need to find and digest information, many of our study participants did not find them universally suitable. Thus, for our participants, it was still important to process update information efficiently. Note that in Part II of this dissertation, we will explore ways to improve security information dissemination at an Internet scale.

4.4.1 Update Processes

We asked our study participants about how they discovered the updates they applied. In our survey, we asked a closed-ended question with 11 possible options and a free-form response (as shown in Table 4.1), while our interview question was open-ended. In total, 99/102 survey participants and all 17 interview subjects responded. The types of information sources discussed by interview subjects overlapped with our survey question options, but we note that the distributions among

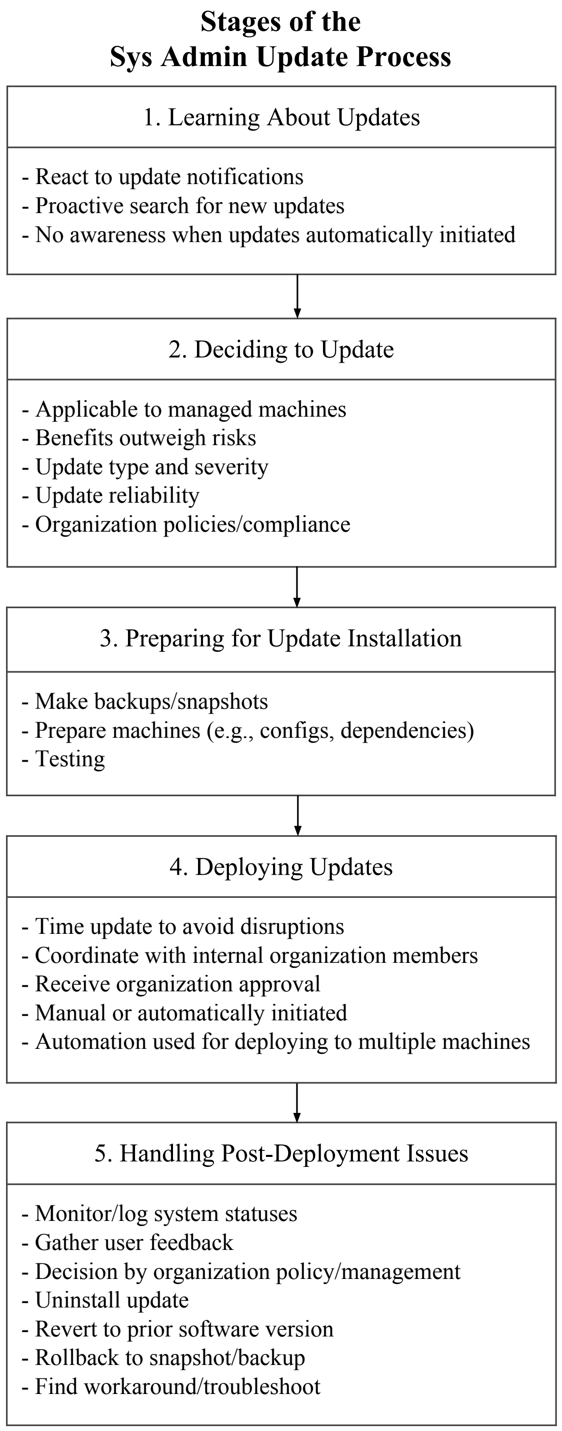

Stages of the Sys Admin Update Process
1. Learning About Updates
- React to update notifications
- Proactive search for new updates
- No awareness when updates automatically initiated
2. Deciding to Update
- Applicable to managed machines
- Benefits outweigh risks
- Update type and severity
- Update reliability
- Organization policies/compliance
3. Preparing for Update Installation
- Make backups/snapshots
- Prepare machines (e.g., configs, dependencies)
- Testing
4. Deploying Updates
- Time update to avoid disruptions
- Coordinate with internal organization members
- Receive organization approval
- Manual or automatically initiated
- Automation used for deploying to multiple machines
5. Handling Post-Deployment Issues
- Monitor/log system statuses
- Gather user feedback
- Decision by organization policy/management
- Uninstall update
- Revert to prior software version
- Rollback to snapshot/backup
- Find workaround/troubleshoot

Figure 4.1: Our study identified five primary stages of the update process for system administrators. We list the salient considerations for each stage.

Table 4.1: Sources used for discovering available updates.

	Source for Update Availability	**# Survey Responses**	**# Interview Responses**
1.	Security advisories	80 (78%)	4 (24%)
2.	Direct vendor notifications	72 (71%)	11 (65%)
3.	Professional mailing lists	54 (53%)	7 (41%)
4.	Online forums	53 (52%)	7 (41%)
5.	Alerts from software	41 (40%)	10 (59%)
6.	News	40 (39%)	5 (29%)
7.	Blogs	39 (38%)	5 (29%)
8.	Third-party services	28 (28%)	0 (0%)
9.	RSS feeds	22 (22%)	3 (18%)
10.	Project mailing lists	21 (21%)	0 (0%)
11.	Social media	18 (18%)	1 (6%)
12.	Other	9 (9%)	3 (18%)
13.	No Answer	3 (3%)	0 (0%)

survey and interview participants differed, likely due to the open-ended nature of the interview question.

As shown in Table 4.1, our participants relied on various types of information sources. Most survey respondents reported a median of 5 different types of sources, and a quarter reported using seven or more types. (We do not report the same counts for interview data given that open-ended responses are not necessarily comprehensive nor indicative of prevalence, as discussed in Section 4.2.5.) This large quantity of source types suggests that update information is highly dispersed, requiring administrators to diligently peruse a variety of outlets to stay informed on available updates. Some interview participants described sourcing information in this manner as non-ideal, as typified by P5's discussion on discovering updates that patched newly identified vulnerabilities: *"There's not always a canonical place to go for a web advisory. When these vulnerabilities get found on the Internet, they might affect you, it could be announced on the Apache web server mailing list, it could be on the Ubuntu server list, it could be a topic on Server Fault. There's a lot of places."* Also, not all sources were ideal. For example, P13 stated that *"sometimes if there's a really critical vulnerability, email's not the most real-time method of getting things going."*

4.4.2 Impact on Updating Effectiveness

Our study participants revealed that they each relied on a diverse set of methods for retrieving update information from multiple sources. Due to the lack of a centralized source of information, we note that it is possible that some system administrators may lack the full coverage of relevant information if they miss an important source. We also observed that administrators used some sources that require active retrieval and digestion, such as news articles, blog posts, forums, and social media. These sources may require more time and effort, compared to sources that push information directly to the administrators, such as direct vendor notifications or mailing lists. Our study ultimately does not concretely reveal how comprehensive or effective administrators are at update information retrieval, but suggests that this is a nontrivial task for many.

4.5 Stage 2: Deciding to Update

For the second stage of their updating process, administrators in our study filtered update information to decide if they should deploy an update. This was a nontrivial task because of the profusion of update information from a variety of sources.

4.5.1 Update Processes

In our survey, we asked respondents about which types of updates they most frequently apply. In our interviews, we asked our participants how they determined which updates to deploy, and which types of updates they considered important. From the responses, we observed five primary factors that our participants discussed for assessing the cost-benefit trade-off of applying available updates. Our interview question was open-ended, so this set of factors may not be comprehensive or indicative of prevalence, as discussed in Section 4.2.5.

1. Update Type: In a closed-ended question, we asked our survey participants which updates they regularly installed: security or non-security related updates. In total, 97/102 administrators regularly installed security updates, whereas only 63/102 administrators did likewise for non-security related updates. (3 respondents did not answer.) We similarly asked our interview subjects an open-ended question about their views on which updates were important or not. Most interview participants (15/17) said that they considered security updates to be vital, but they disagreed on the importance of other updates; 7 administrators considered them important, whereas 5 administrators did not, often feeling they could be disruptive. For example, in a quote that is typical of what we heard, P16 explained: *"Least important, anything that's like feature updates or considered upgrades. I don't really want new features, because new features mean new problems, so I just want to get the security stuff tucked away."* Thus, our study participants typically found security updates important to apply.

2. Update Severity: In an open-ended interview question on how administrators decided to apply an update, the severity of the issues addressed by an update was a factor discussed by 9/17 interview participants. In a canonical example, P13 prioritized updates to *"Only critical security ones...It mostly depends on the severity and what the risk is."*

3. Update Relevance: When discussing their process for deciding to apply an update, five interview participants (29%) explicitly described update information overload, where much of the information they acquired did not apply to their machines. As a result, they said that they had to tediously filter out unnecessary information (or possibly avoid overly verbose feeds altogether). For example, P6 thought that *"Sometimes there's an overabundance of information...there are some products, things like that, that we don't use here. So I have to actively filter that out myself."* Others described receiving multiple emails about specific upgrades (e.g., Linux patches simultaneously released in batches) and how these emails were easily lost or hard to process in an overflowing inbox.

4. Update Reliability: Three interview subjects brought up known update issues as another factor in determining whether to update. For example, P11 cared about the update quality, saying *"a reliability score of an update would be my number one [update characteristic]."*

5. Organizational Factors: In many cases, organizational or management policies and decisions influenced or even dictated the update decision. We discuss in more detail in Section 4.9.

4.5.2 Impact on Updating Effectiveness

We found that system administrators prioritized updates that fixed security (or other severe) bugs. However, many software updates bundle bug fixes with feature or performance changes, including popular software such as the Mozilla Firefox Browser [12] and the Apache HTTP web server [5]. This entanglement suggests that it is challenging for administrators to specifically address the most urgent software problems without contending with other potential changes. Additionally, certain update characteristics (e.g., update reliability) were important to our study participants in deciding whether to apply an update. However, updates may not contain information to assess such characteristics (e.g., Firefox [12], Apache HTTP daemon [5]), or provide too much irrelevant information (described by study subjects as information overload), making it challenging for administrators to make informed updating decisions.

4.6 Stage 3: Preparing for Update Installation

After identifying appropriate updates (potentially containing security patches), our study participants reported that they had to make preparations for installation, which fell into three overarching categories. First, administrators frequently *made backups/snapshots* in case problems arose through the updating process. Second, they *prepared machines* when necessary, such as by changing configurations or dependencies. These actions were often necessary due to the manual nature of many updates. Finally, they often extensively *tested* updates for unintended side-effects or bugs. Here, we focus on the testing considerations of administrators as we cover the other two considerations in the remaining sections.

Threat of Bad Updates: We asked our survey and interview participants to describe their experiences with problems caused by updates on the machines they managed. In a closed-ended survey question, we asked how frequently an administrator encountered a problematic update. Of the 98/102 survey respondents that answered, all but 2 said that they had encountered bad updates; 54 indicated this happened infrequently, 36 found problems every few update cycles, and 6 said most update cycles produced complications. When asked an open-ended question on whether they tested updates and why, our interview subjects expressed the same sentiments on update risk; 8/17 recounted running into a recent faulty update. While the participants' recollections may not have been entirely accurate, it reflected a general sentiment among them that updating comes with non-trivial risks that they should manage. Furthermore, we found in Chapter 3 that the development of security patches is often unreliable in reality, producing faulty fixes. Sliwerski et. al. [176] found similar results for bug fixes in general. Thus, administrator concerns are justified. In the worst case, the negative experiences drove administrators towards fewer updates: *"I stopped applying updates because it was becoming more of a problem to apply them than not to. Production machines, they don't get updates"* (P12). Such behavior can leave hosts riddled with security vulnerabilities and ripe for compromise. To combat the risks of bad updates, many of our study participants engaged in the time-consuming process of update testing.

4.6.1 Update Processes

Both our surveys and interviews contained an open-ended question asking respondents about what testing they do for updates, if any, and why. The majority of our participants (83/102 survey respondents and all 17 interview subjects) indicated they tested updates. (Seven survey participants did not respond.) Among those who tested, 22 survey participants and 3 interview subjects discussed only ad hoc testing methods (e.g., testing basic software functionality) without discussing any strategies in detail. For the remaining administrators, we found that testing strategies varied but fell into two general classes: *staggered deployments* and *dedicated testing setups*. Regardless of the chosen strategy, testing was often a pain point for administrators: in open-ended survey questions on what works well and poorly in an administrator's updating process, only 14/102 survey respondents recounted positive testing experiences, and 12 reported that developing a reliable testing workflow was the most difficult aspect of updating. Thus, many of our study participants found it challenging to develop a dependable testing process.

1. Staggered deployments. When staggering update deployment (as illustrated in Figure 4.2), administrators in our study described separating their machines into multiple groups, deploying updates to a group at a time, and waiting some time between each stage of deployment to observe if update issues arise. In an example that summarizes this approach, S72 said that they first *"install on non-important machines and let them bake for 1+ months."* This strategy, which merges update testing and deployment, was the most commonly used among our study participants, leveraged by 43/102 of the survey respondents and 11/17 of the interview subjects.

We identified three different ways that participants used to grouped machines in each stage. First, 22 survey respondents and 4 interview subjects categorized machines into priority levels, testing updates first on lower priority machines. A second approach (10 survey respondents and 2 interview subjects) was to test first on the machines of end users who opted into assisting with update testing. For example, P10 talked about deploying updates to volunteers for a week prior to company-wide rollout, a strategy many spoke of using because: *"They're very good at reporting things that have gone wrong."* A final less-frequently used strategy was to pick pilot groups at random, only discussed by one survey participant and two interview subjects. While P11 selected machines completely at random, independent of the user, P5 chose randomly with more nuance: *"Usually, it's randomly picking something that I know is active but not the most active machine out there. If I pick something that nobody's using for anything, then, that's not a good place to test it. But, it's also not one of our highest risk servers."*

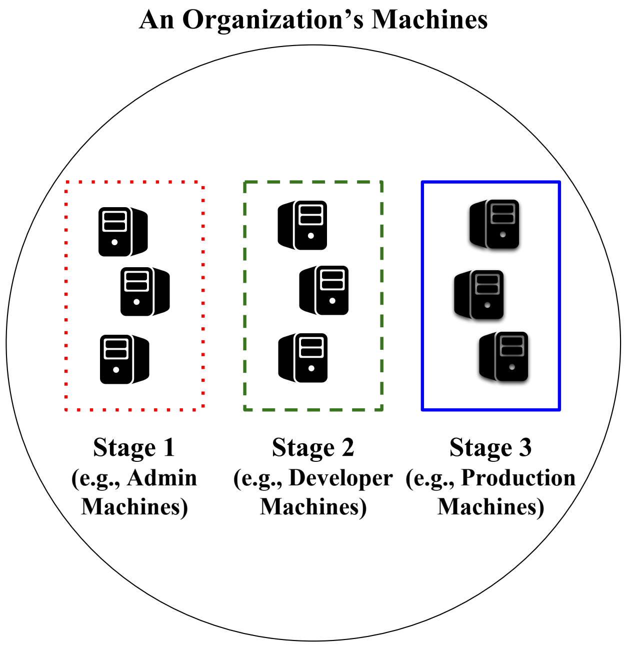

An Organization's Machines
Stage 1
(e.g., Admin Machines)
Stage 2
(e.g., Developer Machines)
Stage 3
(e.g., Production Machines)

Figure 4.2: **Staggered Deployment Testing:** The system administrator allocates machines into stage groups, and updates stage by stage, waiting between each stage for update issues to manifest. If they arise, the administrator halts deployment and investigates the issues. For example, an administrator at a software company might first group only machines that they use as the first stage, then group developer machines as a second stage, and form a final stage of production machines.

Our survey participants typically did not indicate how they monitored for update problems during staggered deployment, although four respondents mentioned gathering user feedback from those who piloted updates. Interview participants told us that they monitored how well updates were applied through monitoring software (6/17), lack of error messages (6/17), checking the machines for compliance (2/17), and user feedback (1/17).

2. Dedicated testing environments. Our survey participants often mentioned a dedicated testing setup, where they used machines provisioned specifically for testing (30/102 survey respondents) or relied on a testing or quality assurance (QA) team (9/102). (Five survey respondents used both approaches.) Figure 4.3 illustrates this process. Among interview participants, 8/17 used dedicated test servers, with two also having a QA team. S29 captured the gist of this approach: *"We test in a lab/test environment that has similar functions as our production environment. We do this to ensure we get accurate and reliable results that won't break our end users' applications."* Similarly, S19 gave an example of how QA teams conducted testing: *"For some third-party software (issue tracking, artifact management, etc.), our QA department has scripts to exercise business-critical functionality."* We note that 16 survey respondents and 5 interview subjects with dedicated testing also used staggered deployment, suggesting that often participants felt that dedicated testing was not sufficient by itself.

Most of our study participants did not elaborate on the specific evaluation done in dedicated

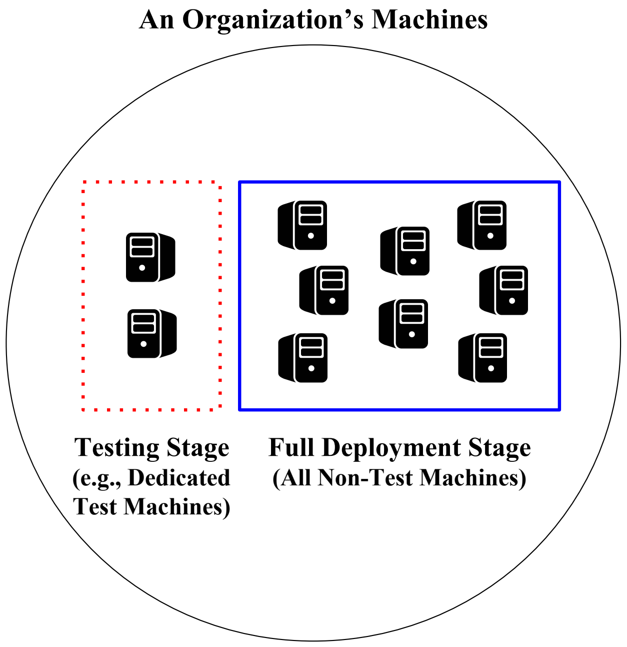

An Organization's Machines
Testing Stage (e.g., Dedicated Test Machines)
Full Deployment Stage (All Non-Test Machines)

Figure 4.3: **Dedicated Testing Environments:** The system administrator evaluates an update in an environment configured specifically for testing (e.g., test servers). If they do not discover update issues, they fully deploy the update (potentially via staggered deployment).

testing setups, although some mentioned automated software testing and manual investigation to confirm that critical software functionality remained. We note that no more than three survey or interview participants explicitly mentioned any particular method though, so further exploration on dedicated testing details is warranted.

4. No testing at all. A minority of survey respondents (10/102) did not test updates at all, and an additional two respondents indicated that they skipped testing on some of their systems as it was infeasible, without discussing testing on other systems. No interview subjects avoided testing. Three of the survey respondents who skipped testing did not provide their reasons. However, two survey respondents indicated they lacked the time, and three others deemed updates in their environment to be low-risk enough to deploy without testing. For example, S43 acknowledged, *"It is a poor habit but I don't ever experience any issues with Microsoft updates, so I see no reason to wait before applying them."* In another instance, a survey respondent skipped updates because testing on a diverse set of hosts seemed impractical, stating that with *"Too many different environments, would need to test a dozen different ways before deployment"* (S34). The final test-less respondent S37 stated that they skipped testing because *"security patches are a requirement, if it breaks something it gets fixed downstream."*

4.6.2 Impact on Updating Effectiveness

Those participants who used the staggered deployment testing strategy avoided the need for dedicated testing resources (although some used both strategies). However, we note that an important downside of staggered deployment for participants was that it could significantly delay updates to hosts in later stages. Some study participants indicated this delay could be on the order of weeks or months. Notably, administrators often spoke of updating production machines last, which is particularly concerning as these servers often directly interacted with external entities and hence, potential attackers.

Some of our administrators preferred dedicated testing environments for evaluating updates in a low-risk setting. However, we note this strategy requires additional computing resources or employees specifically for testing. In addition, we heard from participants about the challenges in replicating nuances of real-world deployments in testing environments. Ultimately, update testing was a challenging endeavor for most of the administrators in our study, driving some to even bypass testing.

4.7 Stage 4: Deploying Updates

Our study participants had to develop methods for deploying updates across the many machines under their purview.

4.7.1 Update Processes

Specifically, our study participants had to determine how to deploy updates and when to do it.

1. How to Deploy? In a closed-ended survey question, we asked survey participants whether they deployed updates manually, wrote their own programs or scripts to deploy updates, used third-party update management software, enabled automatic updates, or deployed in an alternate fashion (with a free-form response). Based on the 99/102 survey respondents who answered, we observed that administrators often lacked a single unified system for deploying updates. While 34 survey participants used a single method, the rest used multiple, with a median of 2 methods. We asked interview subjects an open-ended question on how they install updates, and interview participants also reported a mixture of deployment methods.

A majority of survey respondents used third-party update managers[1] (64/102), as did 12/17 of the interview subjects. P14 described their use of the update management software Ansible [4], explaining *"with Ansible you would just specify a list or subsection of a list of machines to run a particular command or update and it would run all of those in parallel on each of the machines and return the status of the request."* Some interview participants felt these tools could be improved to take snapshots of their systems and better indicate missing updates for specific machines.

[1]Tools mentioned included Ansible [4], SCCM [129], Chef [9], Spiceworks [23], Puppet [21], Terraform [26], and WSUS [130].

Almost half of our participants (50/102 survey respondents and 7/17 interview subjects) created custom scripts or programs to automate the deployment process, while 44/102 survey participants and 2/17 interviewees enabled automatic updates for some software packages. Manual updates were still frequent though, conducted by 40/102 survey respondents and 4/17 interview subjects. One consequence of the heavy use of scripting and manual actions was the issue of legacy systems and processes. For example, P7 illustrated one scenario, saying *"If there's a legacy system in place and Jeff the sysadmin is the only dude who even knows how to run the scripts for that, or whatever service is running on there, you know, God forbid Jeff gets hit by a bus."*

On Automation: In response to open-ended survey questions on what aspects of an administrator's update management process work well and which are challenging, many study participants spoke of the importance of automation in the update deployment process. In a representative quote, S62 explained: *"Automating the process is essential for any environment with more than 10 endpoints as it greatly reduced the time involved and also improves the frequency of patch application."* S19 agreed in their response to the same survey question, stating *"There is no way our small team could manage this many machines without [automation]."* However, implementing automation often required significant effort. P15 stated they did not initially automate due to *"just the amount of time it would take to implement all the automation."* This participant did later deploy automation, stating it took them *"three months, to get it right."*

Even with the benefits of automation, our survey participants also highlighted that many situations still required manual actions, some in preparation for update installation (as mentioned in Section 4.6). For example, S14 sometimes still performed manual updates because *"Major OS updates require more manual intervention, such as updating custom scripts, updating or rewriting configuration files, or updating third-party tools."* In the interviews, some subjects mentioned that automation was not always desirable since update issues could arise unexpectedly.

Dependency and compatibility concerns posed particular problems for automation. In a prototypical example, S62 struggled with *"Maintaining compatibility with software that depends on platforms like Java/.Net/etc. Vendors tend to lag behind the platform by at least 1-2 release cycles preventing us from updating to the latest version."* Additionally, host heterogeneity (e.g., different software versions) complicated update deployment as illustrated by the following typifying example: S86 found deploying updates difficult when *"pushing to multiple versions of Linux with only one tool."*

Thus, while automatic updates and deployment automation was helpful and important for our study participants, they often could not fully automate updates across their machines due to some of the above reasons.

2. When to Deploy? In open-ended interview questions on how administrators deployed updates and whether they had to notify anyone about the update, our interview subjects frequently discussed the need to minimize disruptions for users and updated machines. (Our survey did not contain equivalent questions.) One strategy for mitigating disruptions (used by 13/17 interview subjects) was to update along a predictable schedule, such as P10's weekly patching program, so that users were not caught off-guard by the update timing. Another strategy mentioned by 12/17 interview participants was to update during off-hours. We also observed that organization and management decisions could dictate when updates occurred (described in Section 4.9).

In many cases, communication and coordination with those affected by an update were vital. This sentiment is best exhibited by P10's (who followed a weekly update schedule) discussion of their coordination efforts: *"On a given week, your machine might get software and it might reboot. We have a communication program that goes along with that, that we send out to the units about what's happening this week."* In a contrasting but similar example of coordination, P5 told us that they based update timing on user preferences: *"You send out an email to people and see what time works best for them. Usually, they can identify a time that is going to be idle for them or lower use than regular."*

4.7.2 Impact on Updating Effectiveness

Challenges in implementing automation for update deployment forced many of our participants to perform manual updates. In addition, administrators in our study often eschewed automatic updates so they could make proper preparations. We note that these manual actions could result in slower update rollouts leaving machines exposed to bugs and vulnerabilities for a longer duration. Also, manual updates may require further effort and be more prone to human error, potentially resulting in misconfigurations or functionality regressions. For our participants, the need to time updates in coordination with organization members or policies further widened the vulnerability window for machines.

4.8 Stage 5: Handling Update Issues After Deployment

Unfortunately, update testing did not always prevent issues from arising post-deployment. As found in Chapter 3 and by Sliwerski et. al. [176], a non-trivial fraction of security and general bug fixes are indeed problematic. We asked our survey participants an open-ended question on how they became aware of problems caused by installed updates. In total, 56/102 survey participants found out about some update issues through user or client complaints, while 21 discovered problems through monitoring updated hosts. We further asked both our survey and interview participants open-ended questions about how they handled these post-deployment problems.

4.8.1 Update Processes

Of the 93/102 survey respondents that answered, only 3 indicated they lacked a process for managing post-deployments issues. From the interviews, 11/17 subjects reported recently running into post-deployment problems.

For the administrators that did deal with update complications, the most common approach was to uninstall an update. In total, 48/102 survey participants used this strategy, with 6 mentioning that they did so with custom scripts and 20 using third-party software or an update manager to do so (the rest did not specify). Similarly, 6/17 interviewees mentioned having to uninstall updates to resolve update problems. Another common approach was to revert to a previous snapshot or backup of the software or system. This strategy, used by 35/102 survey respondents and

7/17 interview subjects, did require proactive steps in preparation for update installation (namely, making a backup), as mentioned in Section 4.6. In an example of the forethought required of administrators, S5 discussed their backup strategy: *"I take an image of the entire disk once a month for non-critical machines and daily for critical machines."* Other rollback strategies mentioned less frequently during the surveys and interviews included downgrading to an earlier version of the software (possibly undoing several update cycles), manually negating an update's changes, or reverting to a mirrored/parallel environment.

The prior strategies all involve returning to a pre-update state, which can leave machines without patches for new vulnerabilities. Some administrators preferred to keep the updates in place, with 15/102 survey participants and 1/17 interviewees saying they attempted to find workarounds for problematic updates. Of these, 4 survey participants said they never roll back, focusing on keeping updates in place while managing any issues. Also, 7/102 of our survey participants relied on vendor assistance in resolving update issues.

4.8.2 Impact on Updating Effectiveness

After deploying an update, if problems arose, our study participants tended to revert to a functional but insecure prior state, demonstrating that they prioritized functionality over security. This behavior also suggests that system administrators found it difficult to identify workarounds or fixes for update problems, whether by themselves or via the software vendors.

4.9 Across Stages: Organization and Management Influence

A significant theme that emerged from our study participants was the important role that an organization's internal policies and management could play in update decisions. This theme provides new evidence extending the work by Dietrich *et al.* [58], who also observed that organizational factors impacted how administrators handled system misconfigurations.

We briefly note that we explored whether organizational structure, such as the number of employees or machines managed, affected our participant's update management practices, particularly related to different testing and deployment strategies. To do so, we compared the distributions of the organization size and the number of machines managed between those adopting different updating behaviors. We used the Mann-Whitney-Wilcoxon test [120], with a p-value threshold of $\alpha = 0.05$, to determine if the distributions statistically significantly differed. However, we did not identify any significant differences; thus, the organizational structure did not appear strongly correlated with any particular update process.

4.9.1 Update Processes

Across responses to various open-ended questions, our study participants discussed situations where organizational policies and management affected updating practices.

1. Free Rein. In some organizations, administrators had decision-making authority and could apply updates as they saw fit. However, this put the onus on the administrator solely to keep machines secure. P11's company exemplified this approach: *"I don't have to run junk through a bunch of red tape to do anything. I just do it, knowing the consequences; things could break, could cause a lot of problems and lose a lot of money, but that's just part of having the responsibilities of that job I have. If I want to push out updates to all 1,800 machines, I don't have to really answer to anybody."*

2. Organizational Oversight. In other cases, administrators in our study told us they had to get management buy-in before taking certain update actions. A quote from S26 characterizes this setup, as they talked about applying updates only after management approval because *"I will be fired if I do so before I can convince management."* Similarly, in another representative example, S70 discussed that their update promptness was often delayed because *"Mostly the business being incompetent and not approving the work to go ahead. If it was up to me, [updates would be installed] as soon as they are released and after testing."* This setup often made updating challenging for participants. For example, S14 had to fight for maintaining Windows updates, as management felt that those updates were not trustworthy. These disagreements between administrators and management appeared to result in updating practices that the administrators in our study did not always support.

In some cases, organizational policies dictated the actions of the administrators. A canonical example from S37 illustrates the pressure on their update deployment timeline: *"Policy and compliance require deploying them within 5/10/30 days depending on severity."* In another example quote, P15 explained that their organization's requirements determined the priority of different updates: *"We have compliance implications around getting security updates out, so that's one. We have an organizational mandate to deliver a stable platform, so stability updates set prioritized as well."* With a potentially less secure outcome, P12's organization decided to reduce the frequency of machine updating, because *"that's just more of a decision that we've made as a business that...it's just better not to introduce a problem."*

Several study participants also commented on another important organizational decision: the budget allocated for system administrator operations. For example, S21 said they lacked the time for managing updates but *"My company won't let me buy anything to help with automatically deploying."* Similarly, P16 said that they lacked the budget for obtaining good software to handle updates until demonstrating their network's insecurities to management.

4.9.2 Impact on Updating Effectiveness

Organizational freedom allowed some of our study participants to more effectively apply updates, but placed the burden of security on their shoulders alone. We note that such freedom could result in ad hoc decision making by administrators, potentially resulting in poor practices, or decisions that could negatively impact other aspects of an organization, such as the reliability or availability of an organization's production systems.

By contrast, requiring management approval complicated the update process for many system administrators and could delay or prevent the application of updates. Such barriers also drove down the updating frequency for those administrators who told us they can only request approval for the most severe updates, and often, some skipped less severe updates to avoid the hassle of getting approval.

4.10 Discussion

Our study of system administrator software updating identified how administrators perform updates and the security implications of their behaviors. Future user studies on administrator software updating could extend our work to develop a richer model of update decision-making processes, investigate how updating differs for different types of updates, organizations, and machines, explore the effects of organizational policies on updates in more depth, and identify concrete steps for improving updating tools and interfaces. In this section, we synthesize our findings to identify how software updating differs between system administrators and end users, and how we can help administrators better keep machines updated through recommendations grounded in our results.

4.10.1 Comparison with End User Software Updating Practices

Prior work on software updating behavior has primarily studied end users. From synthesizing and comparing with the results from existing studies [71,123–125,193,194], we find that end users follow similar stages of the updating process, but with differing considerations at each stage. Overall, we observe that administrators performed more sophisticated tasks (e.g., testing) and had unique aspects of their workflows as a result of managing numerous heterogeneous machines within an organizational context (e.g., staggered deployment, organizational influences). For each of our five updating stages (summarized in Figure 4.1), we highlight the salient differences between end user and system administrator considerations.

- **Stage 1 (Learning):** Administrators relied on a diverse set of update information sources, including those from proactive searching. In comparison, end users primarily learned about updates through notifications or alerts from within their software and rarely sought updates by themselves [124, 194].
- **Stage 2 (Deciding):** Like end users, administrators in our study considered the benefits and risks of an update [71, 124, 125, 193, 194]. However, our participants had the additional facet of determining if and which updates affected the potentially heterogeneous hosts in their organization. Some administrators also had to abide by organization policies.
- **Stage 3 (Preparing):** We observed that update-induced issues concerned both our study participants and end users [71, 123, 124, 193, 194]. As a result, end users either avoided updating, updated after making backups, or dealt with update issues only after applying [194]. In comparison, administrators took more extensive preparatory steps, including backing up and snapshotting systems, modifying software configurations and dependencies, and testing updates before applying them.
- **Stage 4 (Deploying):** As administrators in our study deployed updates at scale, unlike end users, they had to consider the interruptions and downtime on machines they served, often

requiring coordination with other organization members or organization approval to take actions. They also often employed automation to scale up their updating tasks.
- **Stage 5 (Remedying):** When updates caused issues, both populations employed similar high-level remedies (e.g., uninstalling updates, finding workarounds) [194]. However, administrators in our study had to contend with the challenge of identifying update issues across numerous machines that they updated, requiring them to consider monitoring systems and feedback from these machines' users. Additionally, as these issues could affect organizational operations, organization factors influenced how administrators handled these situations.

4.10.2 Reducing the Burden of Update Information Retrieval

In Section 4.4, we learned that information on software updates is widely dispersed across various sources. Our findings suggest that helping administrators more easily identify relevant updates for their machines would simplify their updating efforts and increase the likelihood of prompt updating. One solution could be to standardize and consolidate update information at a centralized repository (similar to efforts on aggregating vulnerability information [191]), providing a singular destination for identifying available updates.

Another intriguing approach is through outreach campaigns that inform administrators about severe vulnerabilities and promote updating to patch the security holes. This topic is explored in Part II of this dissertation.

4.10.3 Simplifying Update Decision-Making

Our findings in Section 4.5 indicate that administrators prioritize updates with certain characteristics (e.g., update severity). Standardizing update information (as has been done with vulnerabilities [191]) to consistently include such characterizations would aid them in their decision-making. In particular, administrators differentiate update types. Thus, there is value in splitting all-inclusive updates into updates specific to one type of patch, as also recommended by prior work on end user updates [124, 194]. For example, software vendors could bundle security patches separately from feature patches. With this segregation, administrators can better prioritize the updates they apply (e.g., security fixes). However, we recognize that splitting updates could complicate software development and release. Future work could therefore explore how best to separate and enable updates of different types, from both the software developer and administrator standpoints.

4.10.4 Improving Update Deployment Processes

There remains a salient need for advancements in the update tools that system administrators rely upon, as we observed that administrators encountered various hurdles throughout the preparation and deployment of updates (Sections 4.6 and 4.7), and the handling of post-deployment problems (Section 4.8). For example, the notion that automatic updates would solve the patching problem is overly simplistic, as our findings demonstrate the complexities of the updating process (particularly with situations still requiring manual actions, as discussed in Section 4.7).

While technical developments are certainly needed, we also lack a deep understanding of the usability of these tools. Therefore, the usable security community could contribute explorations into how administrators use update tools and how their interfaces could be improved. For example, our findings (in Section 4.7) indicate that many administrators use third-party update managers. What information do they display before, during, and after update deployment, and what missing information (such as on dependencies or affected configurations) would streamline administrator workflows if provided?

One notable deployment issue our administrators faced was timing updates to avoid interruptions to crucial system operations. We believe that dynamic software updating [94] (DSU), a method that allows for live updates without restarts or downtime, could help with side-stepping update timing concerns. While it has not yet been widely deployed, the approach is promising as some major systems have adopted it, such as with the Linux kernel extension Ksplice [19]. However, we have little understanding so far of how using DSU systems affect developers writing patches and administrators operating such systems. For example, the use of DSU systems can result in complex data representations and less readable code, potentially impacting the software development process. Similarly, DSU systems may not serve as a complete solution for system administrators if they still require approval or coordination before initiating updates, even without system downtime. Research into the usability of dynamic updating systems and avenues for improvement could potentially eliminate update timing concerns for administrators in the future.

4.10.5 Shifting Organizational Culture on Software Updates

In Section 4.9, we identified that organization management and policies can impact administrator actions, often impeding secure updating practices. A culture shift at organizations to recognize the importance of expedient updates (particularly for security issues) would help administrators perform their jobs more successfully. If end users and management do not readily accept that updates should be routinely applied, it becomes difficult to balance system maintenance and security with minimizing operational interruptions. Similarly, if organizations do not devote enough resources for administrators to adequately perform update tasks or have some oversight for security operations, security lapses can occur (e.g., Equifax [146]).

Resolutions to this problem are not straightforward. Existing recommendations such as NIST SP 800-40 [144] provide some guidance on organizational structures that promote updating. However, investigating how administrators deal with data breaches (similar to studies on end users facing breaches [213]) could provide insights into how to better facilitate practices that enable, not hinder, security, beyond solely relying on organizational security education. Such studies could also inform regulatory policies on security oversight. For instance, Equifax currently reports to 8 US states about their security overhaul [147]. The usable security community could offer insights into whether such audits fit into administrator workflows and improve security overall, or whether other policy approaches may better incentivize organizations to implement and prioritize security best practices.

4.11 Conclusion

System administrators play a vital role in securing machines on behalf of their organizations. One of their primary tasks is to manage the updates on numerous hosts to counter emergent vulnerabilities. However, prior work has paid less attention to how exactly they do so. In this chapter, we examined how administrators manage software updates, determining five primary stages of updating and the various considerations and actions associated with each stage. We identified pain points in administrator updating processes, such as when learning about updates, testing for and handling update-caused issues, deploying updates without causing operation disruptions, and dealing with organizational and management oversight. Based on our findings, we developed recommendations grounded in our results, and provided research directions for better support of administrators in keeping their hosts updated and secure.

Chapter 5

Remediation Discussion and Conclusion

Progressing towards effective solutions to a problem as complex as Internet-scale security remediation requires understanding how and why that problem manifests. In Part I, we focused on the security patches that in theory should address the flood of software vulnerabilities constantly being discovered. Our work established empirical grounding on how security patches are developed and applied for Internet systems in practice, and the impacts on the remediation process. We started with the creation of the patches. In Chapter 3, we developed a method to collect a large dataset pertaining to security patches for open-source software projects, and analyzed the characteristics of their development process as well as of the resulting fixes. However, a security patch is only effective if applied to vulnerable systems before attacker exploitation. In Chapter 4, we turned our attention to how system administrators applied security fixes through software updates to Internet systems. Through a multi-part user study, we identified administrator software updating workflows and some of the security implications of their decisions.

These studies have shed light on where we come up short when developing and applying security patches, as well as potential paths forward for the security (as well as the broader computer science) community. Salient conclusions from Chapter 3 on patch development include:

- Approximately a fifth of CVE vulnerabilities are not patched prior to public disclosure, leaving systems exposed without a complete remedy. These security issues skew towards less severe ones potentially requiring less immediate attention, but still over 10% of high severity CVEs were unpatched when publicly announced. Thus, in many cases, the timely creation of a vulnerability fix is the root issue, indicating we either need advancements in software development technologies to reduce the costs from developing patches or further resources for open-source organizations to better address emergent security concerns.

- The majority of security fixes developed prior to disclosure appear publicly in open-source software repositories on the order of weeks to months in advance of their announcements, providing attackers with an opportunity to get a head start on learning of and exploiting a vulnerability before defenders can take action. Attackers have effectively leveraged this observation in practice [89], suggesting that perhaps open-source software projects in general should control information surrounding security issues and their fixes (as some organizations have, such as Mozilla [137]).

- A non-trivial fraction (about 10%) of patches are either incomplete or introduce new issues, justifying the concerns users (and system administrator) have about patch reliability. Without improvements to the production quality of security patches, we will continue to struggle to incentivize prompt patching, particularly with fostering the widespread use of automatic software updates. Our observation that security patches tend to be smaller and more localized than other types of software fixes may guide the design of mechanisms that specifically cater to the quick development of reliable security patches.

Similarly, the insights below from systematically exploring system administrator software updating practices in Chapter 4 provide some explanations for the continued vulnerability of Internet systems in practice, and areas we must focus on in future work.

- While administrators and end users use the same software mechanisms for the actual software update and thus share similar high-level stages of the updating process, the detailed considerations are different between the populations. These distinctions highlight the importance of studying specialized populations. In the context of Internet systems, administrators are particularly relevant as they manage many of the publicly accessible systems.
- Administrators struggle with an information problem when dealing with updates (much more so than end users). They must collect and process information from a disparate set of sources, overcoming both missing helpful information as well as information overload. As a community, we can work towards reducing the burden of information retrieval, whether it is through standardizing and centralizing update information, or through proactive outreach campaigns (as explored in Part II of this dissertation).
- Fear of problematic updates (justified from our security patch analysis in Chapter 3) drove administrators to extensively evaluate updates before complete deployment. Such efforts were still imperfect and often resulted in significant delays in patch application, particularly to the most critical and exposed systems (e.g., public-facing production servers). As discussed earlier with patch development, the need to drive more reliable patch production and testing is urgent if we hope to shift updating behavior en masse.
- Existing software updating technology has limitations we must continue to work towards removing. For example, update deployment automation was vital to many administrators, but they often still had to perform manual actions to safely and effectively apply updates to the heterogeneous systems under their purview. Similarly, recovering from update issues typically involved completely undoing the update's changes, resulting in a functional but again insecure state.
- Software updating for administrators involved more than purely technical considerations, including organizational and economic aspects. In particular, administrators do not operate

in isolation, but rather engage with others in the organization, including management entities. Improving administrator updating will require factoring in these influences.

We recognize that the work discussed so far does not itself solve security problems directly. However, we consider the work as a vital step in the scientific process. In particular, the value of this work is in providing systematic, evidenced-based understanding of remediation behavior in the real world, identifying areas needing improvement, and suggesting promising and/or urgent directions to focus on (backed by data). In Part II, we investigate some of these directions to drive better remediation behavior. However, we aim for our results to help guide the community more widely in advancing along other dimensions in the pursuit of a more secure Internet.

On a closing note, the two chapters of Part I are both empirical security studies, but are drastically different in their methods. Chapter 3's analysis of security patch development involved data mining of software repositories and patch analysis. In contrast, Chapter 4's investigation of system administrator software updating required user studies to assess the human aspects. The diversity of these studies highlights the complexity of the remediation problem space. There are numerous factors at play beyond traditional computer technical considerations, including human, organizational, and economic facets. To tackle this problem space, the solution space must similarly encompass these various dimensions, indicating that future work must recognize, respect, and pursue progress along these various directions.

Part II

Promoting Remediation through Internet-Scale Outreach

Chapter 6

Internet Outreach Introduction and Related Work

6.1 Introduction

A secure Internet ecosystem requires continual discovery and remediation of software vulnerabilities and critical misconfigurations. Security researchers discover thousands of such issues each year, across a myriad of platforms [191]. Once a problem is uncovered, this process proceeds through (1) identifying remedies, (2) informing those affected about the issue and the fixes, and (3) applying the remedies to affected systems. In Part I of this dissertation, we focused on the remedies themselves and investigated the first and last phases of this process, exploring the development of patches that remedy vulnerabilities (Chapter 3) and how system administrators apply software updates that remedy security concerns (Chapter 4). In Part II, we turn our focus to the middle phase of this process, where information about remedies is disseminated. Thus, we conclude with our treatment of the complete remediation life cycle.

Today, the process for informing those affected about a security issue remains at best ad hoc. Unlike the public health community, which has carefully studied and developed best practices for patient notification (e.g., [45, 143]), the security community lacks significant insight into the kinds of notification procedures that produce the best outcomes.[1] Instead, for most software, the modern practice of vulnerability notification remains broadcasting messages via well-known mailing lists or websites that administrators must periodically poll and triage (as seen by the software update information sources used by system administrators, as discussed in Chapter 4).

Developments in high-speed scanning [66, 86] and network monitoring (e.g., [181, 206]) have significantly advanced the ease with which investigators can today often determine the affected parties across the Internet. Thus, the question then arises of how they should best utilize that

[1] An exception concerns the development of online software update systems that explicitly tie together notification and remediation, allowing precise and automated updating targeting the affected parties. Unfortunately, the vast majority of software lacks such systems; even for those that do, operators may disable it in a number of scenarios, as observed in our system administrator user study in Chapter 4.

information. In the early days of the commercial Internet, performing large-scale notifications was often assumed to be both ineffective and impractical [54, 60, 96, 136]. However, we have little empirical grounding on the impact of such outreach efforts, particularly in today's Internet. With the potential to drive better remediation behavior at an Internet-scale, it behooves the security community to investigate if these notifications can be effective, and if so, how best to conduct the outreach efforts. At the same time, we must balance the benefits to the ecosystem (and the associated ethical responsibilities to notify) against the burden this imposes on the reporter, which calls for determining notification regimens that will not prove unduly taxing.

In the studies forming Part II, we describe the first systematic investigation into this techno-social approach at improving global remediation behavior, striving to inform and drive the development of "best practices" for the Internet community. Across these studies, we explore a multitude of solution dimensions, although the solution space is simply too wide to comprehensively explore in a single dissertation. Here, we aim to develop soundly supported results for the most salient basic issues, with an eye towards then facilitating follow-on work that builds on these findings to further map out additional considerations.

The studies described here were conducted over multiple years, one after another. They share many similarities and goals, as they build on top of each preceding work. Thus, we will describe these works in chronological order. We start with the first foray into evaluating Internet-scale outreach (Chapter 7). At this initial point, we lacked a concrete understanding of whether security notifications have any impact. To establish some empirical grounding, we conducted a controlled experiment on Internet-wide notifications for the critical 2014 OpenSSL Heartbleed vulnerability [128], observing for the first time that notifications drove a significant increase (almost 50%) in the vulnerability's patching rate. These results were promising but limited, as the Heartbleed incident was an unprecedented security event. To explore outreach in other contexts, we analyzed the efforts of Google Safe Browsing and Search Quality in reaching out to operators of compromised websites (Chapter 8), and found that direct communication with webmasters increased the likelihood of cleanup by over 50%, and reduced infection durations by more than 60%. Despite the complexity of compromise recovery, over 75% of all notified webmasters successfully recovered their websites. Building on these positive results, we explored how best to conduct notifications through randomized multivariate controlled experiments with vulnerability and security misconfiguration notifications (Chapter 9), providing data on the efficacy of different notification practices.

6.2 Related Work

Beyond the studies in Part II, several other works have considered large-scale security notifications. Concurrent with our study of the Heartbleed bug in Chapter 7, Kührer et al. identified over 9 million public NTP amplifiers and performed a large-scale notification campaign to reduce the number of NTP servers with the *version* and *monlist* commands enabled [110]. They posted public advisories about both NTP commands in collaboration with a number of security organizations, including MITRE, CERTs, and PSIRT. As the amplification factor of NTP *monlist* is orders of magnitude higher than other NTP features, they also directly notified ISPs, hosting providers, data clearing houses, and public NTP awareness projects with regards to NTP *monlist* amplifiers. After one year, they found the number of NTP *version* and *monlist* amplifiers decreased by 33.9% and 92.4%, respectively. However, unlike our work, the study lacked a control to understand the precise impact of notification.

Others have conducted experiments to more systematically evaluate notifications and their impact, many building on top of the work described in Part II. Prior to our investigations, Vasek et al. conducted a small-scale notification experiment involving 161 infected websites [195] and found that after 16 days, 55% of notified sites cleaned up compared to 45% of unnotified sites. They further note that more detailed notifications outperformed reports with minimal information by 13%, resulting in a 62% cleanup rate.

Expanding on our work, Cetin et al. also performed a small-scale notification experiment measuring the role of sender reputation when notifying the owners of hijacked websites [50]. They emailed the WHOIS contacts of 240 infected sites from email addresses belonging to an individual independent researcher (low reputation), a university research group (medium reputation), and an anti-malware organization (high reputation). While nearly twice as many notified sites cleaned up within 16 days compared to unnotified ones, they found no significant differences across the various senders. In a follow up study, Cetin et al. [49] investigated whether providing a mechanism to verify vulnerability could encourage further remediation, focusing on a security weakness affecting DNS authoritative name servers. They set up a website that demonstrated the vulnerability, and found that those who visited their demonstration web site were more likely to fix, but only 10% of contacts visited. They observed high bounce rates when attempting to contact the nameserver owners, indicating a significant fraction of their notifications did not reach anyone.

Concurrent to our study on the effectiveness of different vulnerability practices (Chapter 9), Stock et al. investigated the feasibility of large-scale notifications for web vulnerabilities [179, 180], experimentally evaluating the effectiveness of different communication channels, including WHOIS email contacts and service providers. Additionally, they analyzed the reachability and viewing behavior of their messages. Their results largely accord with ours, providing a complementary study of notifications in a separate context (namely, vulnerable websites). Notably, they likewise observed that notifications could induce a statistically significant increase in patching, although the raw impact may be limited.

A couple of studies have looked at notifying service providers rather than end hosts directly, ultimately observing mixed success. In 2013, Canali et al. [44] studied how website hosting providers handle malware reports. For 20 hosting providers, they hosted Web sites serving malware and self-reported each site to the corresponding hosting provider, observing that 64% of the complaints were ignored. Similarly, Nappa et al. [142] notified 19 hosting providers about malicious servers in their network; however, only 7 actually took steps to take down the affected servers.

Our work described in Part II complements these works, expanding our knowledge about Internet-scale security outreach campaigns through systematic empirical evaluations.

Chapter 7

First Foray: The Matter of Heartbleed

7.1 Introduction

In March 2014, researchers found a catastrophic vulnerability in OpenSSL, the cryptographic library used to secure connections in popular server products including Apache and Nginx. While OpenSSL has had several notable security issues during its 16 year history, this flaw—the *Heartbleed* vulnerability—was one of the most impactful. Heartbleed allows attackers to read sensitive memory from vulnerable servers, potentially including cryptographic keys, login credentials, and other private data. Exacerbating its severity, the bug is simple to understand and exploit.

In this chapter, we analyze the impact of the vulnerability and track the server operator community's responses. Using extensive active scanning, we assess who was vulnerable, characterizing Heartbleed's scope across popular HTTPS websites and the full IPv4 address space. We also survey the range of protocols and server products affected. We estimate that 24–55% of HTTPS servers in the Alexa Top 1 Million were initially vulnerable, including 44 of the Alexa Top 100. Two days after disclosure, we observed that 11% of HTTPS sites in the Alexa Top 1 Million remained vulnerable, as did 6% of all HTTPS servers in the public IPv4 address space. We find that vulnerable hosts were not randomly distributed, with more than 50% located in only 10 ASes that do not reflect the ASes with the most HTTPS hosts. In our scans of the IPv4 address space, we identify over 70 models of vulnerable embedded devices and software packages. We also observe that both SMTP+TLS and Tor were heavily affected; more than half of all Tor nodes were vulnerable in the days following disclosure.

Our investigation of the operator community's response finds that within the first 24 hours, all but 5 of the Alexa Top 100 sites were patched, and within 48 hours, all of the vulnerable hosts in the top 500 were patched. While popular sites responded quickly, we observe that patching plateaued after about two weeks, and 3% of HTTPS sites in the Alexa Top 1 Million remained vulnerable almost two months after disclosure.

In an attempt to drive remediation of this critical vulnerability, starting three weeks after disclosure, we undertook a large-scale notification effort and contacted the operators responsible for the remaining vulnerable servers. By contacting the operators in two waves, we could conduct a controlled experiment and measure the impact of notification on patching. We report the effects of our notifications, observing a surprisingly high 47% increase in patching by notified operators.

We draw upon these observations to discuss both what went well and what went poorly in the aftermath of Heartbleed. By better understanding the lessons of this security disaster, the technical community can respond more effectively to such events in the future.

This chapter's work was published in the ACM Internet Measurement Conference (IMC) [65], and was awarded Best Paper.

7.2 Background

On April 7, 2014, the OpenSSL project publicly disclosed the Heartbleed vulnerability, a bug in their implementation of the TLS Heartbeat Extension. The vulnerability allowed attackers to remotely dump protected memory—including data passed over the secure channel and private cryptographic keys—from both clients and servers. In this section, we provide a brief history of OpenSSL, the Heartbeat Extension, and details of the vulnerability and its disclosure.

7.2.1 OpenSSL: A Brief History

OpenSSL is a popular open-source cryptographic library that implements the SSL and TLS protocols. It is widely used by server software to facilitate secure connections for web, email, VPN, and messaging services. The project started in 1998 and began tracking vulnerabilities in April 2001.

Over the last 13 years, OpenSSL has documented six code execution vulnerabilities that allowed attackers to compromise private server data (e.g., private cryptographic keys and messages in memory) and execute arbitrary code. The project has faced eight information leak vulnerabilities, four of which allowed attackers to retrieve plaintext, and two of which exposed private keys. Two of the vulnerabilities arose due to protocol weaknesses; the remainder came from implementation errors.

The Heartbleed bug reflects one of the most impactful vulnerabilities during OpenSSL's history for several reasons: (1) it allowed attackers to retrieve private cryptographic keys and private user data, (2) it was easy to exploit, and (3) HTTPS and other TLS services have become increasingly popular, resulting in more affected services.

7.2.2 TLS Heartbeat Extension

The Heartbeat Extension allows either end-point of a TLS connection to detect whether its peer is still present, and was motivated by the need for session management in Datagram TLS (DTLS). Standard implementations of TLS do not require the extension as they can rely on TCP for equivalent session management.

Peers indicate support for the extension during the initial TLS handshake. Following negotiation, either end-point can send a `HeartbeatRequest` message to verify connectivity. The

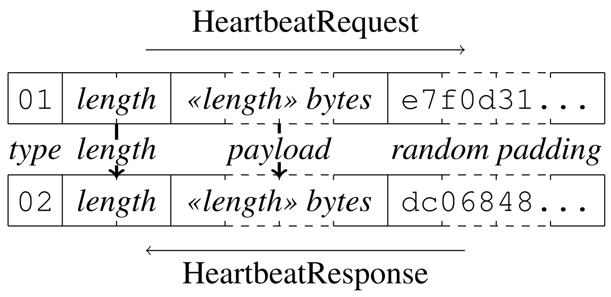


Figure 7.1: **Heartbeat Protocol.** Heartbeat requests include user data and random padding. The receiving peer responds by echoing back the data in the initial request along with its own padding.

extension was introduced in February 2012 in RFC 6520 [173], added to OpenSSL on December 31, 2011, and released in OpenSSL Version 1.0.1 on March 14, 2012.

`HeartbeatRequest` messages consist of a one-byte type field, a two-byte payload length field, a payload, and at least 16 bytes of random padding. Upon receipt of the request, the receiving end-point responds with a similar `HeartbeatResponse` message, in which it echoes back the `HeartbeatRequest` payload and its own random padding, per Figure 7.1.

7.2.3 Heartbleed Vulnerability

The OpenSSL implementation of the Heartbeat Extension contained a vulnerability that allowed either end-point to read data following the payload message in its peer's memory by specifying a payload length larger than the amount of data in the `HeartbeatRequest` message. Because the payload length field is two bytes, the peer responds with up to 2^{16} bytes (~64 KB) of memory. The bug itself is simple: the peer trusts the attacker-specified length of an attacker-controlled message.

The OpenSSL patch adds a bounds check that discards the `HeartbeatRequest` message if the payload length field exceeds the length of the payload. However, while the bug is easy to conceptualize and the fix is straight-forward, the potential impact of the bug is severe: it allows an attacker to read private memory, potentially including information transferred over the secure channel and cryptographic secrets [68, 161, 184].

7.2.4 Heartbleed Timeline

The Heartbleed vulnerability was originally found by Neel Mehta, a Google computer security employee, in March 2014 [91]. Upon finding the bug and patching its servers, Google notified the core OpenSSL team on April 1. Independently, a security consulting firm, Codenomicon, found the vulnerability on April 2, and reported it to National Cyber Security Centre Finland (NCSC-FI). After receiving notification that two groups independently discovered the vulnerability, the OpenSSL core team decided to release a patched version.

Date	Event
03/21	Neel Mehta of Google discovers Heartbleed
03/21	Google patches OpenSSL on their servers
03/31	CloudFlare is privately notified and patches
04/01	Google notifies the OpenSSL core team
04/02	Codenomicon independently discovers Heartbleed
04/03	Codenomicon informs NCSC-FI
04/04	Akamai is privately notified and patches
04/05	Codenomicon purchases the `heartbleed.com` domain
04/06	OpenSSL notifies several Linux distributions
04/07	NCSC-FI notifies OpenSSL core team
04/07	OpenSSL releases version 1.0.1g and a security advisory
04/07	CloudFlare and Codenomicon disclose on Twitter
04/08	Al-Bassam scans the Alexa Top 10,000
04/09	University of Michigan begins scanning

Table 7.1: **Timeline of Events in March and April 2014.** The discovery of Heartbleed was originally kept private by Google as part of responsible disclosure efforts. News of the bug spread privately among inner tech circles. However, after Codenomicon independently discovered the bug and began separate disclosure processes, the news rapidly became public [91, 153].

The public disclosure of Heartbleed started on April 7, 2014 at 17:49 UTC with the version 1.0.1g release announcement [153], followed by the public security advisory [152] released at 20:37 UTC; both announcements were sent to the OpenSSL mailing list. Several parties knew of the vulnerability in advance, including CloudFlare, Akamai and Facebook. Red Hat, SuSE, Debian, FreeBSD and ALT Linux were notified less than 24 hours before the public disclosure [91]. Others, such as Ubuntu, Gentoo, Chromium, Cisco, and Juniper were not aware of the bug prior to its public release. We present a timeline of events in Table 7.1.

7.3 The Impact of Heartbleed

Heartbleed had the potential to affect any service that used OpenSSL to facilitate TLS connections, including popular web, mail, messaging, and database servers (Table 7.2). To track its damage, we performed regular vulnerability scans against the Alexa Top 1 Million domains [1] and against 1% samples of the public, non-reserved IPv4 address space. We generated these samples using random selection with removal, per ZMap's existing randomization function [66]. We excluded hosts and networks that previously requested removal from our daily HTTPS scans [64]. In this section, we analyze the impact on those services—particularly HTTPS. We have publicly released all of the data used for this analysis at `https://scans.io/study/umich-heartbleed`.

7.3.1 Scanning Methodology

We tested for the Heartbleed bug by modifying ZMap [66] to send Heartbeat requests with no payload nor padding, and the length field set to zero. Per the RFC [173], these requests should be rejected. However, vulnerable versions of OpenSSL send a response containing only padding, rather than simply drop the request. The patched version of OpenSSL—as well as other popular libraries, including GnuTLS, NSS, Bouncy Castle, PolarSSL, CyaSSL and MatrixSSL—correctly discard the request (or do not support the Heartbeat Extension).

We emphasize that this approach does not exploit the vulnerability or access any private memory—only random padding is sent back by the server. While it was later found that Heartbleed scanning caused HP Integrated Lights-Out (iLO) devices to crash [15], we received no reports of our scans disrupting these devices—likely because our approach did not exploit the vulnerability. We have publicly released our scanner at `https://zmap.io`.

7.3.2 False Negatives

Our Heartbleed scanner contained a bug that caused vulnerable sites to sometimes appear safe due to a timeout when probing individual hosts. The root cause was that the scanner labelled each host's vulnerability as false by default, rather than null or unknown. If a Heartbleed test timed out, the scanner returned the host's vulnerability status as the default false, providing no indication of a failed test. The result is a potential false negative, where the scan reports the system as immune. Note that our scanner does not however err when reporting a system as vulnerable. As we develop in this section, the likelihood of a given scan exhibiting such as false negative fortunately does not appear to depend on the particular address being scanned, and this allows us to estimate the false negative rate.

We first assessed whether some addresses were more prone to manifest false negatives than others. To do so, we compared three complete IPv4 scans and examined systems reported as vulnerable in one scan but immune in previous scans. Since the scanner does not err in reporting a host as vulnerable, any prior report of immunity reflects a false negative (assuming no one unpatches systems). We found the IP addresses associated with such false negatives spread evenly across the address space, without any apparent correlations. This observation leads us to believe the false negatives manifest in an address-independent manner.

Although our initial scanner did not fetch the web page itself, a subsequent change in the comprehensive scan (but not the incremental scans) attempted to fetch the server's home page. As the home page fetch occurred after the Heartbleed check, any reported home page data implicitly indicates that the Heartbleed test successfully completed without a timeout.

To investigate the false negative rate, we use two full scans of the IPv4 address space, one with and one without home page fetching. The full scan conducted on April 24 did not grab server home pages, while the May 1 scan did, hence we know the validity of scan results from the May 1 scan. To soundly conduct this comparison we need to remove servers that may have switched IP addresses between the two scans. To do so, we only considered servers that presented identical

TLS certificates between the two scans. While this restriction potentially introduces a bias because some sites will have both patched and changed their TLS certificates, the address-independent nature of the false negatives should cause this effect to even out.

Our scanner failed to grab the server home page for 24% of the hosts in the May scan. Of these 24% of hosts, we observe 44% appear immune. False negatives could only have occurred when testing these hosts. The remaining 56% of hosts appeared vulnerable (and hence are correctly labelled). From this, we conclude that at most $(0.24 \cdot 0.44) = 0.105$, or 10.5%, of hosts were incorrectly labelled in the May 1 scan.

For the April scan, the only observable signal of a false negative is if a host was reported immune and then reported vulnerable in the May scan. We find 6.5% of hosts exhibit this behavior. Assuming that people do not unpatch their systems, this provides an estimated lower bound of 6.5% for the April scan false negative rate. This estimate represents a lower bound because we cannot determine the vulnerability in April of a host that appears immune in both scans. In that case, a false negative is a host vulnerable in April but reported as immune, and patched by May. However, we do observe that of hosts reported as vulnerable in the April scan and successfully tested in May (so the server page was retrieved), only 0.36% appeared immune in May, indicating a very modest overall patching rate between the two scans (which accords with Figure 7.3 below). Given that our false negatives are address-independent, we expect a similarly low patch rate for all vulnerable April hosts. Thus, while a 6.5% false negative rate is a lower bound for the April scan, the true rate should not be significantly higher.

Given the similarity of these two false negative estimates using two different scans, ultimately we conclude that the scanner exhibited a false negative rate between 6.5% and 10.5%, but that these manifest independently of the particular server scanned. Due to this address-independent behavior, we can assume a similar false negative rate for sampled scans. We attempt to account for this error whenever possible. In particular, the bias implies that any population-based survey based on a single scan underestimates the vulnerable population. Finally, for our assessment of the impact of notifications (Section 7.5), we only consider a given server as non-vulnerable when it consistently reports as immune in repeated scans, which would require multiple (presumably independent) false negatives to occur before introducing a bias.

7.3.3 Impact on Popular Websites

Determining which websites were initially vulnerable poses significant difficulties. Little attention was paid to the Heartbeat Extension prior to the vulnerability announcement, and many popular sites patched the vulnerability within hours of the disclosure. Codenomicon, one of the groups that discovered Heartbleed, speculated that 66% of HTTPS sites were vulnerable [128]. However, this number represented the Apache and Nginx market share and may well reflect an overestimate, because some operators may have disabled the extension, deployed dedicated SSL endpoints, or used older, non-vulnerable versions of OpenSSL.

Web Servers		Mail Servers		Database Servers		XMPP Servers		Other Servers	
Apache (mod_ssl) [128]	Yes	Sendmail [165]	Yes	MySQL [165]	Yes	OpenFire [28]	No	OpenVPN [154]	Yes
Microsoft IIS [131]	No	Postfix [165]	Yes	PostgreSQL [165]	Yes	Ejabberd [16]	Yes	OpenLDAP [168]	Yes
Nginx [75]	Yes	Qmail [165]	Yes	SQL Server [131]	No	Jabberd14 [203]	Yes	Stunnel [172]	Yes
Lighttpd [165]	Yes	Exim [88]	Yes	Oracle [155]	No	Jabberd2 [105]	Yes	Openswan [148]	Yes
Tomcat [29]	Yes	Courier [93]	Yes	IBM DB2 [98]	No			Telnetd-ssl [25]	Yes
Google GWS [149]	Yes	Exchange [131]	No	MongoDB [135]	Yes			OpenDKIM [17]	Yes
LiteSpeed [116]	Yes	Dovecot [88]	Yes	CouchDB [35]	No			Proftpd [171]	Yes
IBM Web Server [98]	Yes	Cyrus [139]	Yes	Cassandra [7]	No			Bitcoin Client [40]	Yes
Tengine [145]	Yes	Zimbra [157]	Yes	Redis [88]	No				
Jetty [151]	No								

Table 7.2: **Vulnerable Server Products.** We survey which server products were affected by Heartbleed.

7.3.3.1 Top 100 Websites

All of the Alexa Top 100 websites were patched within 48 hours of disclosure—prior to the start of our scans. To document the impact on these websites, we aggregated press releases, other's targeted scans, and quotes provided to Mashable, a news site that hosted one of the popular lists of sites for which users should change their passwords due to possible exposure via Heartbleed [122].

Al-Bassam completed a vulnerability scan of the Alexa Top 10,000 domains on April 8, 2014 at 16:00 UTC (22 hours after the vulnerability disclosure) [32]. His scan found 630 vulnerable sites, 3,687 supporting HTTPS but not vulnerable, and 5,683 not supporting HTTPS. Several prominent sites, including Yahoo, Imgur, Stack Overflow, Flickr, Sogou, OkCupid, and DuckDuckGo, were found vulnerable. We investigated other sites in the Alexa Top 100 and found that half made a public statement regarding vulnerability or provided information to Mashable [30, 32, 33, 37, 56, 61, 77, 90, 99, 100, 102, 122, 149, 159, 163, 186, 188, 204, 209, 210]. Combining these press releases, Mashable's report, and Al-Bassam's scan, we find that at least 44 of the Alexa Top 100 websites were vulnerable. However, this figure reflects a lower bound, as we were unable to find information for some sites. Table 7.3 lists the vulnerability status of the top 30 HTTPS-enabled sites in the US.

7.3.3.2 Estimating Broader Impact

Within 48 hours of the initial disclosure, we conducted our first vulnerability scan of the Alexa Top 1 Million. At that point, we found that 45% of all sites supported HTTPS. 60% of those supported the Heartbeat Extension, and 11% of all HTTPS sites were vulnerable. While 60% of HTTPS sites supported the extension, 91% of these were powered by known vulnerable web servers (e.g., Nginx or Apache Web Server), as shown in Table 7.4. If all of these servers were initially vulnerable and operators installed a patched OpenSSL version (rather than rebuilding OpenSSL with Heartbeat disabled), at most about 55% of the HTTPS sites in the Alexa Top 1 Million were initially vulnerable.

While disabling the largely unused extension would appear to provide an obvious solution, it is not possible to disable the extension through a configuration file. Instead, this change requires

Site	Vuln.	Site	Vuln.	Site	Vuln.
Google	Yes	Bing	No	Wordpress	Yes
Facebook	No	Pinterest	Yes	Huff. Post	?
Youtube	Yes	Blogspot	Yes	ESPN	?
Yahoo	Yes	Go.com	?	Reddit	Yes
Amazon	No	Live	No	Netflix	Yes
Wikipedia	Yes	CNN	?	MSN.com	No
LinkedIn	No	Instagram	Yes	Weather.com	?
eBay	No	Paypal	No	IMDB	No
Twitter	No	Tumblr	Yes	Apple	No
Craigslist	?	Imgur	Yes	Yelp	?

Table 7.3: **Vulnerability of Top 30 US HTTPS-Enabled Websites.** We aggregate published lists of vulnerable sites, press releases, and news sources to determine which of the top sites were vulnerable before the discovery of Heartbleed.

recompiling OpenSSL with a specific flag—an option likely more laborious than updating the OpenSSL software package.

Some sites may possibly have used an older version of OpenSSL that was not vulnerable. To estimate a lower bound for the number of vulnerable sites, we considered sites that used vulnerable web servers and supported TLS 1.1 and 1.2—features first introduced in OpenSSL 1.0.1 along with the Heartbeat Extension. Such sites would have been vulnerable unless administrators had recompiled OpenSSL to explicitly disable the extension.

To estimate the number of sites that supported TLS 1.1 and 1.2 prior to the Heartbleed disclosure, we analyzed the data collected by the Trustworthy Internet Movement's SSL Pulse [24], which provides monthly statistics of SSL-enabled websites within the Alexa Top 1 Million. We find that 56,019 of the 171,608 (32.6%) sites in the SSL Pulse dataset supported TLS 1.1 or 1.2. Of these sites, 72.7% used known vulnerable web servers, yielding an estimated lower bound of 23.7% of the sites being vulnerable.

In summary, we can reasonably bound the proportion of vulnerable Alexa Top 1 Million HTTPS-enabled websites as lying between 24–55% at the time of the Heartbleed disclosure.

7.3.4 Pre-Disclosure Patching

Google, Akamai, and other sites disabled the Heartbeat Extension prior to public disclosure. To detect when services disabled the Heartbeat Extension, we examined data from the ICSI Certificate Notary, which passively monitors TLS connections from seven research and university networks (approximately 314K active users) [34].

The Notary data shows that Google disabled Heartbeat starting at least 12 days prior to public disclosure, with all servers Heartbeat-disabled by April 15. While some servers still had Heartbeat enabled after disclosure, they may not have been exploitable. Google may have already patched

Web Server	Alexa Sites	Heartbeat Ext.	Vulnerable
Apache	451,270 (47.3%)	95,217 (58.4%)	28,548 (64.4%)
Nginx	182,379 (19.1%)	46,450 (28.5%)	11,185 (25.2%)
Microsoft IIS	96,259 (10.1%)	637 (0.4%)	195 (0.4%)
Litespeed	17,597 (1.8%)	6,838 (4.2%)	1,601 (3.6%)
Other	76,817 (8.1%)	5,383 (3.3%)	962 (2.2%)
Unknown	129,006 (13.5%)	8,545 (5.2%)	1,833 (4.1%)

Table 7.4: **Alexa Top 1 Million Web Servers.** We classify the web servers used by the Alexa Top 1 Million Sites, as observed in our first scan on April 9, 2014. Percentages represent the breakdown of server products for each category. We note that while Microsoft IIS was not vulnerable to Heartbleed, a small number of IIS servers used vulnerable SSL terminators.

those servers, and decided afterwards to disable the Heartbeat Extension as a company-wide policy. Similarly, Akamai began disabling Heartbeat at least 4 days prior to disclosure, completing the process by April 18.

7.3.5 Internet-Wide HTTPS Vulnerability

We began performing daily 1% scans of the IPv4 address space on April 9, 48 hours after the disclosure. Our first scan found that 11.4% of HTTPS hosts supported the Heartbeat Extension and 5.9% of all HTTPS hosts were vulnerable. Combining these proportions from random sampling with our daily scans of the HTTPS ecosystem [64] (which do not include Heartbleed vulnerability testing), we estimate that 2.0 million HTTPS hosts were vulnerable two days after disclosure.

Surprisingly, 10 ASes accounted for over 50% of vulnerable HTTPS hosts but represented only 8.6% of all HTTPS hosts (Figure 7.2). With the exception of Comcast Cable Communications, the ASes all belonged to web hosting companies or cloud providers (Table 7.5). The vulnerable hosts in the Comcast AS were Fortinet devices. In the case of Strato Hosting, vulnerable addresses were hosting Parallels Plesk Panel, a web hosting management software. The vulnerable addresses of Minotavar Computers, ZeXoTeK IT-Services, Euclid systems, Vivid Hosting, and ACCESSPEOPLE-DE all served the default Apache page, likely reflecting named-based virtual hosts. In the case of the two Amazon ASes and Hetzner Online, a large number of the vulnerable hosts served public facing websites, and used Apache or Nginx.

7.3.6 Vulnerable Devices and Products

Heartbleed also affected many embedded systems, including printers, firewalls, VPN endpoints, NAS devices, video conferencing systems, and security cameras. To understand the embedded systems that were affected, we analyzed the self-signed certificates employed by vulnerable hosts. We clustered these by fields in the certificate Subject and manually inspected large clusters. From this, we developed device “fingerprints”. We took a conservative approach and chose the most

AS	% of Vulnerable	% of HTTPS
Minotavar Computers EOOD	18.5%	1.7%
ZeXoTeK IT-Services GmbH	13.0%	0.9%
ACCESSPEOPLE-DE ISP-Service	7.4%	0.7%
Amazon.com, Inc.	4.6%	0.8%
Amazon.com, Inc.	4.1%	0.9%
Hetzner Online AG	2.6%	0.4%
Comcast Cable Communications	2.3%	2.8%
Vivid Hosting	2.0%	0.1%
Euclid Systems	1.5%	0.1%
Strato Hosting	1.4%	0.1%
Total	57.4%	8.6%

Table 7.5: **Top ASes with Most Vulnerable Hosts.** We aggregate hosts by AS and find that 57% of vulnerable hosts in the April 9 scan were located in only 10 ASes.

restrictive fingerprints in order to minimize false positive identifications. This, and the manual effort required, means that our fingerprints lack comprehensive coverage. However, we still identified 74 distinct sets of vulnerable devices and software packages that fall into a number of broad categories:

Communication Servers: IceWarp messaging, Zimbra collaboration servers, iPECS VoIP systems, and Polycom and Cisco video conference products.

Software Control Panels: Puppet Enterprise Dashboard, IBM System X Integrated Management Modules, Kloxo Web hosting control panel, PowerMTA, Chef/Opscode management consoles, VMWare servers, and Parallels control panels for Plesk and Confixx.

Network Attached Storage: QNAP, D-Link, ReadyNAS, LaCie, Synology, and Western Digital NAS devices.

Firewall and VPN Devices: Devices from Barracuda Networks, Cisco, SonicWALL, WatchGuard, OpenVPN, pfSense, TOPSEC Network Security (a Chinese vendor), and Fortinet.

Printers: Dell, Lexmark, Brother, and HP printers.

Miscellaneous: Hikvision and SWANN security cameras, AcquiSuite power monitors, IPACCT (a management tool used by Russian ISPs), Aruba Networks WiFi login portals, INSYS VPN access for industrial controllers, and SpeedLine Solutions (the "#1-rated Pizza POS System").

7.3.7 Other Impacts

While our study focuses on Heartbleed's impact on public HTTPS web services, Heartbleed also affected mail servers, the Tor network, Bitcoin, Android, and wireless networks, as we briefly assess in this section.

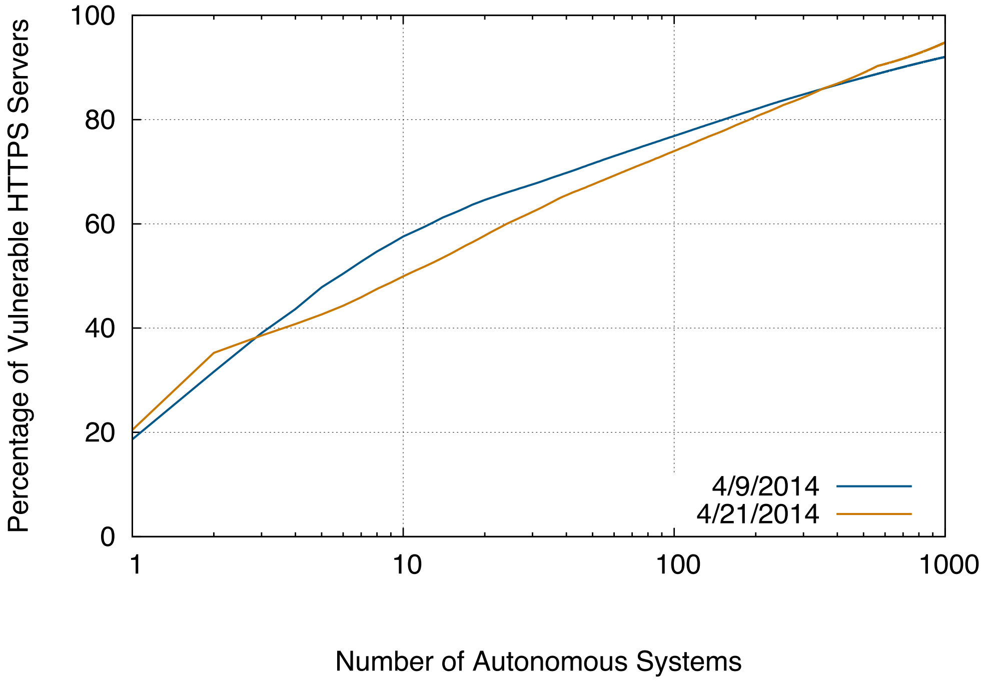


Figure 7.2: **Vulnerable Servers by AS.** We aggregate vulnerable hosts by AS and find that over 50% of vulnerable hosts are located in ten ASes.

Mail Servers. SMTP, IMAP, and POP3 can use TLS for transport security via use of a `StartTLS` directive within a plaintext session. As such, if mail servers used OpenSSL to facilitate TLS connections, they could have been similarly vulnerable to Heartbleed. On April 10, we scanned a random 1% sample of the IPv4 address space for vulnerable SMTP servers. We found that 45% of those providing SMTP+TLS supported the Heartbeat Extension, and 7.5% were vulnerable to Heartbleed.

These estimates only provide a lower bound, because similar to HTTPS, our scanner sometimes timed out, causing false negatives. (We also scanned for IMAP and POP3 servers, but later analysis of the data found systematic flaws that rendered the results unusable.)

Tor Project. Tor relays and bridges use OpenSSL to provide TLS-enabled inter-relay communication. In our April 10 scan, we found that 97% of relays supported Heartbeat and could have been vulnerable. 48% of the relays remained vulnerable at that time, three days after announcement of the vulnerability. The vulnerability could have allowed an attacker to extract both short-term onion and long-term identity keys, ultimately allowing an attacker to intercept traffic and impersonate a relay. In the case of a hidden service, the bug could have allowed an entry guard to locate a hidden service. The Tor client was similarly affected, potentially allowing entry guards to read sensitive information from a client's memory, such as recently visited websites [59].

Bitcoin Clients. Heartbleed affected both Bitcoin clients and exchanges, and in the most severe case, allowed an attacker to compromise the accounts on a popular exchange, BTCJam [42]. The value of Bitcoin did not change drastically on April 7, the date of public disclosure, falling only 3.1% from the previous day (a figure within its regular volatility) and returned to its April 6 value by April 14.

All versions of the Bitcoin software from May 2012 to April 2014 used a vulnerable OpenSSL version [6]. Immediately after Heartbleed's disclosure, a new Bitcoin version was released linking to the newly patched OpenSSL version. Because clients were also affected by the bug, attackers could potentially compromise wallets or retrieve private keys if a susceptible user followed a payment request link [40]. However, we have not found any reports of the theft of Bitcoins from local wallets.

Several companies, including Bitstamp and Bitfinex, temporarily suspended transactions on April 8 until they could patch their servers. In the most severe case, 12 customers had a total of 28 BTC ($\approx$ $6,500) stolen from BTCJam after account credentials were compromised, though with all funds subsequently reinstated by the exchange [42].

Android. Heartbleed only affected Android version 4.1.1 [149]. It is unclear how many devices currently run the affected version, but Google recently estimated that 33.5% of all Android devices currently run Android 4.1.x [3]. A vulnerable device would have been susceptible to having memory read by a malicious server.

Wireless Networks. Several variants of the Extended Authentication Protocol, a commonly used framework for wireless network authentication, use TLS, including EAP-PEAP, EAP-TLS, and EAP-TTLS. For implementations based on OpenSSL, Heartbleed would have allowed attackers to retrieve network keys and user credentials from wireless clients and access points [87]. We do not know of any statistics regarding what sort of vulnerable population this potentially represents.

7.4 Patching

In Section 7.3, we estimated the initial impact of Heartbleed. In this section, we discuss the patching behavior that occurred subsequent to the disclosure.

7.4.1 Popular Websites

Popular websites did well at patching. As mentioned above, only five sites in the Alexa Top 100 remained vulnerable when Al-Bassam completed his scan 22 hours after disclosure. All of the top 100 sites were patched by the time we started our scans, 48 hours after disclosure. As discussed in Section 7.3, our first scan of the Alexa Top 1 Million found that 11.5% of HTTPS sites remained vulnerable. The most popular site that remained vulnerable was `mpnrs.com`, ranked 689th globally and 27th in Germany. Similarly, all but seven of the vulnerable top 100 sites replaced their certificate in the first couple of weeks following disclosure. Most interestingly, `godaddy.com`, operator of the largest commercial certificate authority, did not change their certificates until much later. The other six sites are `mail.ru`, `instagram.com`, `vk.com`, `sohu.com`, `adobe.com`, and `kickass.to`.

As shown in Figure 7.3, while many Alexa Top 1 Million sites patched within the first week, the patch rate quickly dropped after two weeks, with only a very modest decline between April 26 and June 4, 2014. While top sites in North America and Europe were initially more vulnerable than Asia and Oceania (presumably due to more prevalent use of OpenSSL-based servers), they all followed the same general patching pattern visible in Figure 7.3.

7.4.2 Internet-Wide HTTPS

As can be seen in Figure 7.3, the patching trend for the entire IPv4 address space differs from that of the Alexa Top 1 Million. The largest discrepancy occurs between April 22, 14:35 EDT and April 23, 14:35 EDT, during which the total number of vulnerable hosts fell by nearly a factor of two, from 4.6% to 2.6%. This dropoff occurred because several heavily affected ASes patched many of their systems within a short time frame. The shift from 4.6% to 3.8% between 14:35 and 22:35 on April 22 reflects Minotavar Computers patching 29% of their systems, ZeXoTeK IT-Services patching 60%, and Euclid Systems patching 43%. Between April 22, 22:35 and April 23, 06:35, Minotavar Computers continued to patch systems. The last major drop from 3.4% to 2.6% (06:35–14:35 on April 23) was primarily due to Minotavar Computers patched remaining systems, and to a lesser extent, Euclid Systems and Vivid Hosting.

7.5 Notification

Three weeks after the initial disclosure a large number of hosts remained vulnerable, and we began notifying the system operators responsible for unpatched systems. This endeavor provided us with an opportunity to study the impact of large-scale vulnerability notification. In this section we describe our notification methodology and analyze the reactions to our notifications and its impact on patching.

7.5.1 Methodology

In order to find the appropriate operators to contact, we performed WHOIS lookups for the IP address of each vulnerable host appearing in our April 24, 2014 scan of the full IPv4 address space. We used the “abuse” e-mail contact extracted from each WHOIS record as our point of notification. We chose to use WHOIS abuse emails because they struck us as more reliable than emails from other sources. There also appeared to be less risk in offending a network operator through contacting the abuse contact. For example, many emails extracted from certificate Subject fields were not valid emails, and we observed several WHOIS records with comments instructing anything related to spam or abuse be sent to the abuse contact rather than the technical contact.

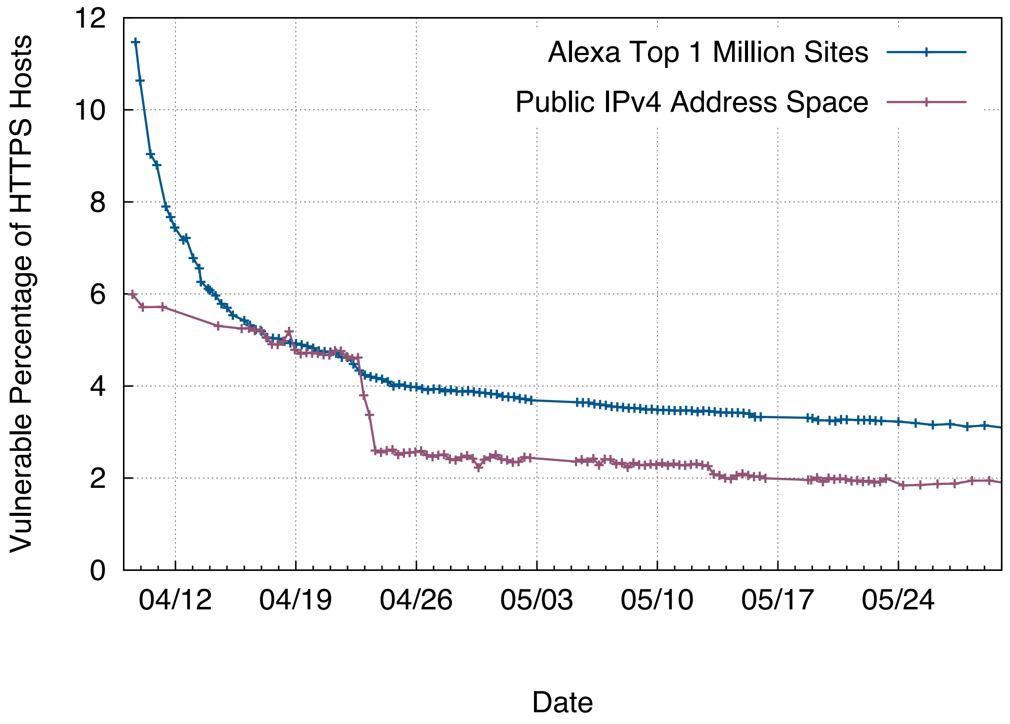


Figure 7.3: **HTTPS Patch Rate.** We track vulnerable web servers in the Alexa Top 1 Million and the public IPv4 address space. We track the latter by scanning independent 1% samples of the public IPv4 address space every 8 hours. Between April 9 and June 4, the vulnerable population of the Alexa Top 1 Million shrank from 11.5% to 3.1%, and for all HTTPS hosts from 6.0% to 1.9%.

Our scan found 588,686 vulnerable hosts. However, we excluded embedded devices—which accounted for 56% of vulnerable hosts—because administrators likely had no avenue for patching many of these devices at the time. These devices were detected using the fingerprints described in Section 7.3.6. The remaining 212,805 hosts corresponded to 4,648 distinct abuse contacts. Approximately 30,000 hosts belonged to RIPE and Amazon each. Because neither of these organizations directly manage hosts, we omitted them from our notifications.

To measure the impact of our notifications, we randomly split the abuse contacts into two groups, which we notified in stages. We sent notifications to the first half (Group A) on April 28, 2014, and the second half (Group B) on May 7, 2014. Our notification e-mail introduced our research and provided a list of vulnerable hosts, information on the vulnerability, and a link to the EFF's Heartbleed recovery guide for systems administrators.

7.5.2 Patching Behavior

To track patching behavior, we conducted regular scans of the known vulnerable hosts every eight hours. We considered a contact as having reacted and begun patching if we found at least one host in the list we sent to the contact as patched. Figure 7.4 shows a comparison of the patch rates

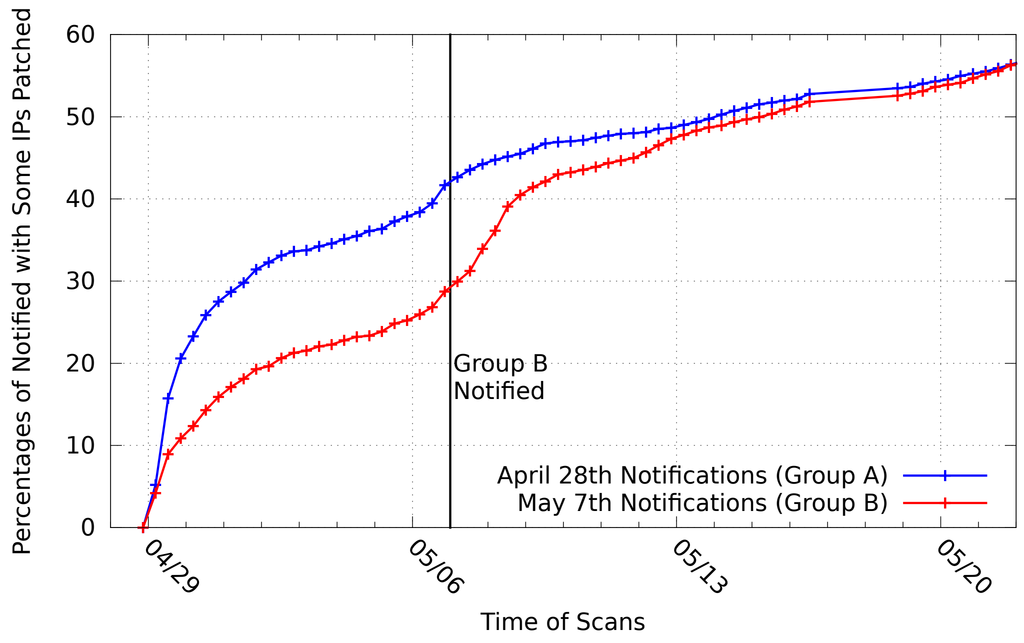


Figure 7.4: **Patch Rates of Group A vs Group B.** The patch rates for our two notification sets show that notification had statistically significant impact on patch rate.

between the two groups. Within 24 hours of the initial notification, 20.6% of the Group A operators had begun to patch, whereas only 10.8% of Group B contacts (not yet notified) had reacted. After eight days (just before the second group of notifications), 39.5% of Group A contacts had patched versus 26.8% in Group B. This is a 47% increase in patching for notified operators.

Fisher's Exact Test yields a one-sided p-value of very nearly zero for the null hypothesis that both groups reflect identical population characteristics. We thus conclude that our notification efforts had a statistically significant positive effect in spurring notified sites to patch. Our second round of notifications followed a similar pattern as the first. As Group A's rate of patching had decreased at that point, Group B caught up, resulting in both converging to around 57% of contacts having reacted within a couple of weeks of notification.

We also investigated the relationship between the reactions of network operators (per Section 7.5.3) and their patching behavior. First, we sent our notification message in English, possibly creating a language barrier between us and the contact. We analyzed the Group A responses and found that email responses entirely in English had no statistically significant difference in the corresponding patching rate than for responses containing non-English text (Fisher's Exact Test yielded a two-sided p-value of 0.407).

We did, however, find statistically significant differences between each of the categories of responses framed below in Section 7.5.3, as shown in Figure 7.5, with human responders patching at the highest rate. Within the first day post-notification, 48% of human responders had begun patching, while none of the other categories had a patch rate higher than 32%.

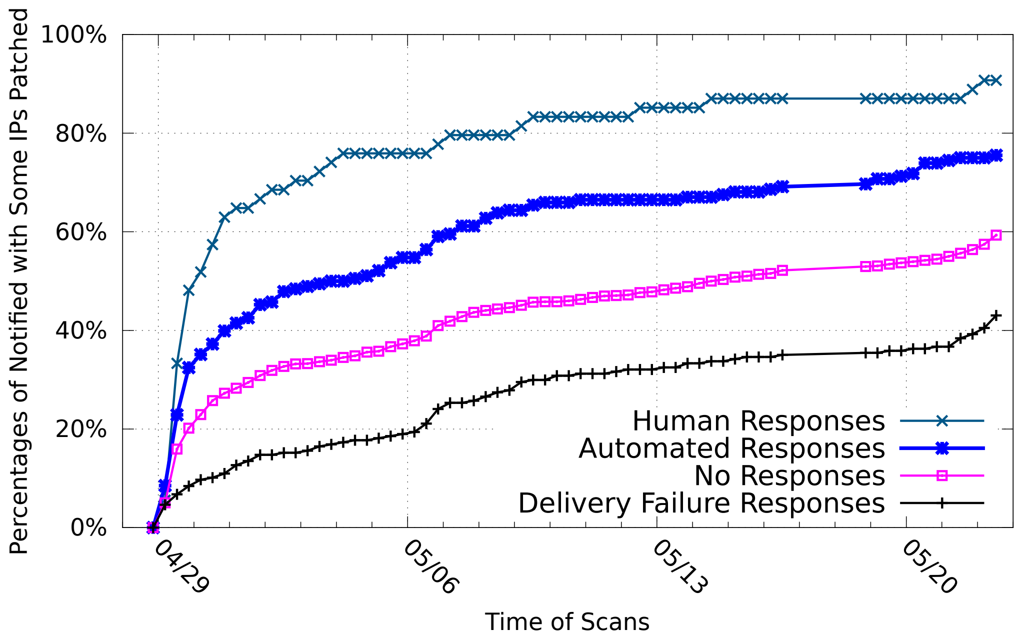


Figure 7.5: **Patch Rates for Different Response Types.** Conditioning on the sort of reply we received for a given notification reveals statistically significant differences.

The second strongest reactions came from contacts configured to send automated responses. 32% had reacted after one day, and 75% had reacted after three weeks. This indicates that operators using a system to automatically process notifications and complaints will still often react appropriately.

Over 77% of the contacts never responded. After one day, 20% of such contacts had conducted some patching; after three weeks, 59% had. Right before Group B's notifications, the patch rate of these contacts was statistically significantly higher than Group B's patch rate. This shows that even when system operators do not respond when notified, they often still patch vulnerable systems.

7.5.3 Responses

In our first group of notifications, on April 28, 2014, we contacted 2,308 abuse contacts and received email responses from 514 contacts. Of these 59 (11%) were clearly human-generated, 197 (38%) were automated, and 258 (50%) were delivery failures. We received 16 automated emails where we subsequently received a human response; these we classified as human (thus, in total we received 530 emails). The vast majority of responses (88%) were in English; other common languages included Russian, German, Portuguese, and Spanish.

We classified a positive response as one that thanked us or stated their plan to remedy their vulnerable hosts. The human-generated responses were overall very positive (54/59), with only three that we deemed neutral, and two negative. The two negative responses questioned the legality of our scan despite our explicit explanation that we did not exploit the vulnerability.

Automated messages came in four forms: confirmations (24%), tickets (44%), trackers (23%), and miscellaneous bounces (9%; primarily out-of-office notices and "no longer working here" messages). Confirmation emails confirmed the receipt of our notification; tickets provided a reference or ticket identifier to associate with our notification message; and trackers were tickets that also explicitly provided a link to a support site to track progress on opened tickets. Curiously, 21 of the 45 trackers did not provide the credentials to log into the support website, 2 provided invalid credentials, and 3 did not have our ticket listed on their support site. In the week following our notification, we were informed that 19 tickets were closed, although only 4 provided any reasoning.

Out of the 258 delivery failure replies, 197 indicated the recipient did not receive our notification. Other error messages included full inboxes or filtering due to spam, and several did not describe a clear error. We observed 30 delayed and retrying emails, but all timed-out within three days.

7.5.4 Network Operator Survey

We sent a brief survey to positive human responders, where all questions were optional, and received anonymous submissions from 17 contacts. Surprisingly, all 17 expressed awareness of the vulnerability and stated their organizations had performed some remediation effort prior to our notification, typically through informing their clients/customers and patching machines if accessible. When we asked why might the hosts we detected still be vulnerable, the most common responses were that they did not have direct control over those servers, or their own scans must have missed those hosts. It appears ignorance of the vulnerability and its threat did not play a factor in slow patching, although our sample size is small. When asked if they replaced or revoked vulnerable certificates, nine said yes, two said no, and one was unaware of the recommendation. Finally, we asked if these contacts would like to receive notifications of similar vulnerabilities in the future. Twelve said yes, two said no, and the others did not respond. This again demonstrates that our notifications were in general well-received.

7.6 Discussion

Heartbleed's severe risks, widespread impact, and costly global cleanup qualify it as a security disaster. However, by analyzing such events and learning from them, the community can be better prepared to address major security failures in the future. In this section, we use our results to highlight weaknesses in the security ecosystem, suggest improved techniques for recovery, and identify important areas for future research.

Support for Critical Projects. While not a focus of our research, many in the community have argued that this event dramatically underscores shortcomings in how our community develops, deploys, and supports security software. Given the unending nature of software bugs, the Heartbleed vulnerability raises the question of why the Heartbeat extension was enabled for popular websites. The extension is intended for use in DTLS, an extension unneeded for these sites. Ultimately, the inclusion and default configuration of this largely unnecessary extension precipitated the Heartbleed catastrophe. It also appears likely that a code review would have uncovered the vulnerability. Despite the fact that OpenSSL is critical to the secure operation of the majority of websites, it receives negligible support [121]. Our community needs to determine effective support models for these core open-source projects.

Vulnerability Disclosure. With the exception of a small handful, the most prominent websites patched within 24 hours. In many ways, this represents an impressive feat. Several factors indicate that patching was delayed because the Heartbleed disclosure process unfolded in a hasty and poorly coordinated fashion. Several major operating system vendors were not notified in advance of public disclosure, ultimately leading to delayed user recovery. As discussed in Section 7.3, a number of important sites remained vulnerable more than 24 hours after initial disclosure, including Yahoo, the fourth most popular site on the Internet. The security community needs to be better prepared for mass vulnerability disclosure before a similar incident happens again. This includes addressing difficult questions, such as how to determine which software maintainers and users to notify, and how to balance advance disclosure against the risk of premature leaks.

Notification and Patching. Perhaps the most interesting lesson from our study of Heartbleed is the surprising impact that direct notification of network operators can have on patching. Even with global publicity and automatic update mechanisms, Heartbleed patching plateaued two weeks after disclosure with 2.4% of HTTPS hosts remaining vulnerable, suggesting that widespread awareness of the problem is not enough to ensure patching. However, as discussed in Section 7.5, when we notified network operators of the unpatched systems in their address space, the rate of patching increased by 47%. Many operators reported that they had intended to patch, but that they had missed the systems we detected.

Although Internet-wide measurement techniques have enabled the mass detection of vulnerable systems, many researchers (including us) had assumed that performing mass vulnerability notifications for an incident like Heartbleed would be either too difficult or ineffective. Our findings challenge this view. Future work is needed to understand what factors influence the effectiveness of mass notifications and determine how best to perform them. For instance, was Heartbleed's infamy a precondition for the high response rate we observed? Can we develop systems that combine horizontal scanning with automatically generated notifications to quickly respond to future events? Can we standardize machine-readable notification formats that can allow firewalls and intrusion detection systems to act on them automatically? What role should coordinating bodies such as CERT play in this process? With additional work along these lines, automatic, measurement-driven mass notifications may someday be an important tool in the defensive security arsenal. We explore answers to some of these questions in the subsequent chapters.

7.7 Conclusion

In this chapter, we analyzed numerous aspects of the recent OpenSSL Heartbleed vulnerability, including who was initially vulnerable and global patching behavior. We found that the vulnerability was widespread, and estimated that between 24–55% of HTTPS-enabled servers in the Alexa Top 1 Million were initially vulnerable, including 44 of the Alexa Top 100. Sites patched heavily in the first two weeks after disclosure, but patching subsequently plateaued, and 3% of the HTTPS Alexa Top 1 Million sites remained vulnerable after two months. In an attempt to reduce the vulnerable population, we also conducted a mass notification of vulnerable hosts, finding a significant positive impact on the patching of hosts to which we sent notifications, indicating that this type of notification helps reduce global vulnerability. Ultimately, we drew upon our analyses to frame what went well and what went poorly in our community's response, providing perspectives on how we might respond more effectively to such events in the future.

Chapter 8

Remedying Web Hijacking: Notification Effectiveness and Webmaster Comprehension

8.1 Introduction

The proliferation of web threats such as drive-by downloads, cloaked redirects, and scams stems in part from miscreants infecting and subverting control of vulnerable web servers. One strategy to protect users from this dangerous content is to present a warning that redirects a client's browser to safer pastures. Examples include Safe Browsing integration with Chrome, Firefox, and Safari that each week alerts over 10 million clients of unsafe webpages [81]; Google Search labeling hacked websites in search results [83]; and Facebook and Twitter preventing users from clicking through malicious URLs [104, 187]. While effective at reducing traffic to malicious pages, this user-centric prioritization ignores long-term webmaster cleanup, relegating infected pages to a dark corner of the Internet until site operators notice and take action.

Inspired by our positive results from notifying about the Heartbleed bug, as discussed in Chapter 7, we could consider turning to notifications as a tool to alert site operators of ongoing exploits or vulnerable software, to encourage them to secure their sites. This new emphasis on webmaster hygiene mitigates the risks associated with users clicking through security indicators [31]. Our Heartbleed study (Chapter 7), as well as concurrent related work [50, 195], suggested that notifying webmasters of hijacked or at-risk properties might expedite remediation. However, a question remains as to the best approach to reach webmasters and whether they comprehend the situation at hand. This comprehension is critical: while browser warnings can overcome the lack of security expertise among users via "opinionated" designs that use visual cues to steer a user's course of action to safe outcomes [70], that same luxury does not exist for notifications, where webmasters must be savvy enough to contend with a security incident, or reach out to someone who can.

In this chapter, we present the first large-scale measurement study on the effectiveness of notifications at inducing quick recovery from website hijacking. As part of our study, we explore:

(1) how various notification techniques and external factors influence the duration of compromise; (2) whether site operators can comprehend and address security threats; and (3) the likelihood that previously exploited websites quickly fall victim again. To conduct our study, we rely on an 11-month dataset from July, 2014–June, 2015 of 760,935 hijacking incidents as identified by Safe Browsing (drive-bys) and Google Search Quality (blackhat SEO). Each of these incidents triggers a combination of an interstitial browser warning, a search result warning, or a direct email notification that alerts webmasters to hijacked properties with concrete examples of injected code.

We find that 59.5% of alerted sites redress infections by the end of our data collection window, with infections persisting for a median of two months. Breaking down this aggregate behavior, we find Safe Browsing interstitials, paired with search warnings that "this site may contain harmful content", result in 54.6% of sites cleaning up, compared to 43.4% of sites flagged with a search warning alone. Above all, direct contact with webmasters increases the likelihood of remediation to over 75%. We observe multiple other influential factors: webmasters are far more efficient at cleaning up harmful content localized to a single directory compared to systemic infections that impact every page on a site; while popular sites (as captured by search ranking) are three times more likely to clean up in 14 days compared to unpopular sites. Our results illustrate that webmasters benefit significantly from detailed, external security alerts, but that major gaps exist between the capabilities of small websites and major institutions.

To better understand the impact of webmaster comprehension on overall remediation, we decouple the period before webmasters receive a notification from the time spent cleaning a site. We capture this behavior based on appeals logs that detail when webmasters request Google to remove hijacking warnings from their site, a request verified by a human analyst or crawler. We find that 80% of site operators successfully clean up on their first appeal attempt. The remaining 20% require multiple appeals, spending a median of one week purging attacker code. Equally problematic, many site operators appear to address only symptoms rather than the root cause of compromise: 12% of sites fall victim to hijacking again within 30 days, with 10% and 20% of re-infections occurring within one day for Safe Browsing and Search Quality respectively. We distill these pain points into a path forward for improving remediation. In the process, we highlight the tension between protecting users and webmasters and the decentralized responsibilities of Internet security that ultimately confound recovery.

In summary, we frame our key contributions as follows:

- We present the first large-scale study on the impact of diverse notification strategies on the outcome of 760,935 hijacking incidents.
- We model infection duration, finding that notification techniques outweigh site popularity or webmaster knowledge as the most influential factor for expediting recovery.
- We find that some webmasters struggle with the remediation process, with over 12% of sites falling victim to re-compromise in 30 days.
- We present a path forward for helping webmasters recover and the pitfalls therein.

This chapter's content appeared in the World Wide Web Conference (WWW) [113].

8.2 Background & Related Work

Web services rely on *notifications* to alert site operators to security events after a breach occurs. In this study, we focus specifically on website compromise, but notifications extend to account hijacking and credit card fraud among other abuse. We distinguish notifications from *warnings* where visitors are directed away from unsafe sites or decisions (e.g., installing an unsigned binary). With warnings, there is a path back to safety and minimal technical expertise is required; breaches lack this luxury and require a more complex remedy that can only be addressed by site operators. Here, we outline approaches for identifying compromise, having discussed related notification work in Chapter 6.

8.2.1 Detecting Compromise

As a precursor to notification, web services must first detect compromise. The dominant research strategy has been to identify side-effects injected by an attacker. For example, Provos et al. detected hacked websites serving drive-by downloads based on spawned processes [162]; Wang et al. detected suspicious redirects introduced by blackhat cloaking [199]; and Borgolte detected common symptoms of defacement [41]. These same strategies extend beyond the web arena to detecting account hijacking, where prior work relied on identifying anomalous usage patterns or wide-scale collusion [67, 185]. More recently, Vasek et al. examined factors that influenced the likelihood of compromise, including the serving platform (e.g., nginx) and content management system (e.g., Wordpress) [196]. They found that sites operating popular platforms such as Wordpress, Joomla, and Drupal faced an increased risk of becoming compromised, primarily because miscreants focused their efforts on exploits that impacted the largest market share. Soska et al. extended this idea by clustering websites running the same version of content management systems in order to predict likely outbreaks of compromise [177]. We sidestep the issue of detection, instead relying on a feed of known compromised pages involved in drive-bys, spam, or cloaking (§ 8.3).

8.2.2 Webmaster Perspective of Compromise

Given a wealth of techniques to detect compromise, the preeminent challenge that remains is how best to alert webmasters to security breaches, assuming webmasters are incapable of running detection locally. StopBadware and CommTouch surveyed over 600 webmasters of compromised websites to understand their process for detecting compromise and remedying infection [183]. They found that only 6% of webmasters discovered an infection via proactive monitoring for suspicious activity. In contrast, 49% of webmasters learned about the compromise when they received a browser warning while attempting to view their own site; another 35% found out through other third-party reporting channels, such as contact from their web hosting provider or a notification from a colleague or friend who received a browser warning.

Equally problematic, webmasters rarely receive support from their web hosting providers. Canali et al. created vulnerable websites on 22 hosting providers and ran a series of five attacks that simulated infections on each of these websites over 25 days [44]. Within that 25-day window, they found that only one hosting provider contacted them about a potential compromise of their website, even though the infections they induced were detectable by free, publicly-available tools. Similarly, the StopBadware and CommTouch study found that 46% of site operators cleaned up infections themselves, while another 20% reached out for professional help [183]. Only 34% of webmasters had the option of free help from their hosting provider (irrespective of using it). These two studies provide qualitative evidence of the struggles currently facing webmasters and the potential value of third-party notifications.

8.3 Methodology

Our study builds on data gathered from two unique production pipelines that detect and notify webmasters about compromise: Safe Browsing, which covers drive-by attacks [162]; and Search Quality, which protects against scams and cloaked gateways that might otherwise pollute Google Search [79]. We describe each pipeline's detection technique, the subsequent notification signals sent, and how each system determines when webmasters clean up. A high level description of this process appears in Figure 8.1. We note that our measurements rely on *in situ* data collection; we cannot modify the system in any way, requiring that we thoroughly account for any potential biases or limitations. We provide a breakdown of our final dataset in Table 8.1, collected over a time frame of 11 months from July 15th, 2014–June 1st, 2015.

8.3.1 Compromised Sites

When Safe Browsing or Search Quality detect a page hosting harmful or scam content, they set a flag that subsequently triggers both notifications and warnings. We group flags on the level of registered domains (e.g., *example.com*), with the exception of shared web hosting sites, where we consider flags operating on subdomains (e.g., *example.blogspot.com*). Both Safe Browsing and Search Quality distinguish purely malicious pages from compromised sites based on whether a site previously hosted legitimate content; we restrict all analysis to compromised sites. For the purposes of our study, we denote a *hijacked website* as any registered or shared domain that miscreants compromise. We use the term *hijacking incident* to qualify an individual attack: if miscreants subvert multiple pages on a website, we treat it as a single incident up until Safe Browsing or Search Quality verify the site is cleaned (discussed in § 8.3.3).[1] We treat any subsequent appearance of malicious content as a new hijacking incident.

As detailed in Table 8.1, we observe a total of 760,935 such incidents, with the weekly breakdown of new incidents shown in Figure 8.2. Our dataset demonstrates that miscreants routinely compromise new websites, with a median of 8,987 new sites detected by Search Quality and 5,802 sites by Safe Browsing each week. We also find evidence of rare, large-scale outbreaks of compromise that impact over 30,000 sites simultaneously.

[1]In the event multiple attackers compromise the same site simultaneously, we will mistakenly conflate the symptoms as a single hijacking incident.

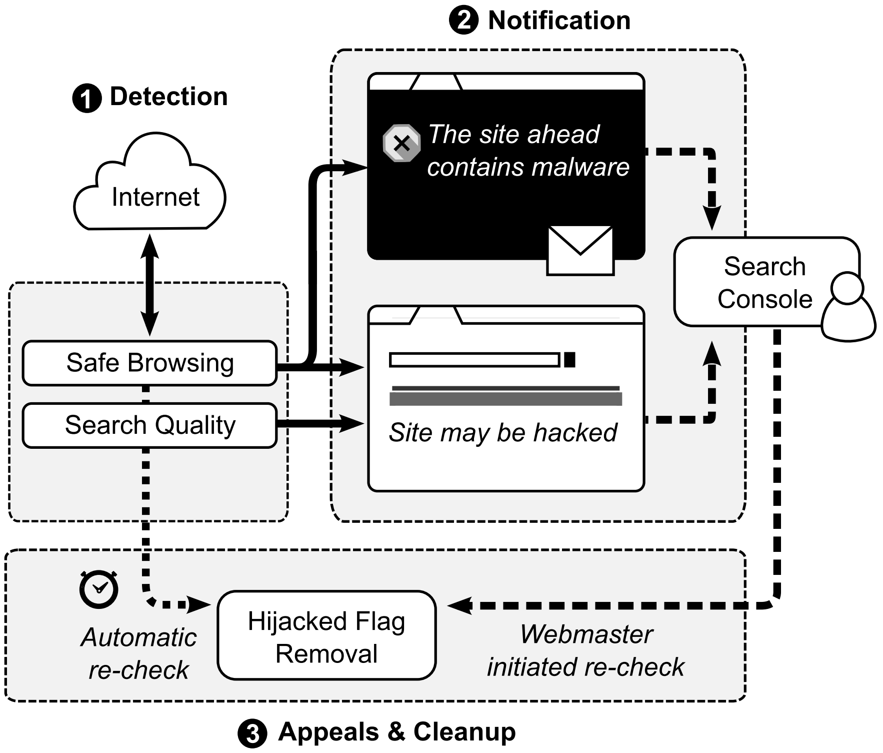


Figure 8.1: Google's hijacking notification systems. Safe Browsing and Search Quality each detect and flag hijacked websites (❶). From there, Safe Browsing shows a browser interstitial and emails WHOIS admins, while both Safe Browsing and Search Quality flag URLs in Google Search with a warning message (❷). Additionally, if the site's owner has a Search Console account, a direct notification alerts the webmaster of the ongoing incident. The hijacking flag is removed if an automatic re-check determines the site is no longer compromised. Webmasters can also manually trigger this re-check via Search Console (❸).

We caution that our dataset is biased to hijacking threats known to Google and is by no means exhaustive. From periodic manual reviews performed by Google analysts of a random sample of hijacking incidents, we estimate the false positive rates of both pipelines to be near zero, though false negatives remain unknown. That said, our dataset arguably provides a representative cross-section of hijacked webpages around the globe. We provide a detailed demographic breakdown later in § 8.4.

8.3.2 Notification Mechanisms

Safe Browsing and Search Quality each rely on a distinct combination of notification and warning mechanisms to alert webmasters either directly or indirectly of hijacking incidents. We detail each of the possible combinations in Figure 8.1, spanning browser interstitials, search page warnings, webmaster help tools (called Search Console), and WHOIS emails. For all of these notification

Dataset	**Safe Browsing**	**Search Quality**
Time frame	7/15/14–6/1/15	7/15/14–6/1/15
Hijacked websites	313,190	266,742
Hijacking incidents	336,122	424,813
Search console alerts	51,426	88,392
WHOIS emails	336,122	0
Webmaster appeals	124,370	48,262

Table 8.1: Summary of dataset used to evaluate notification effectiveness and webmaster comprehension.

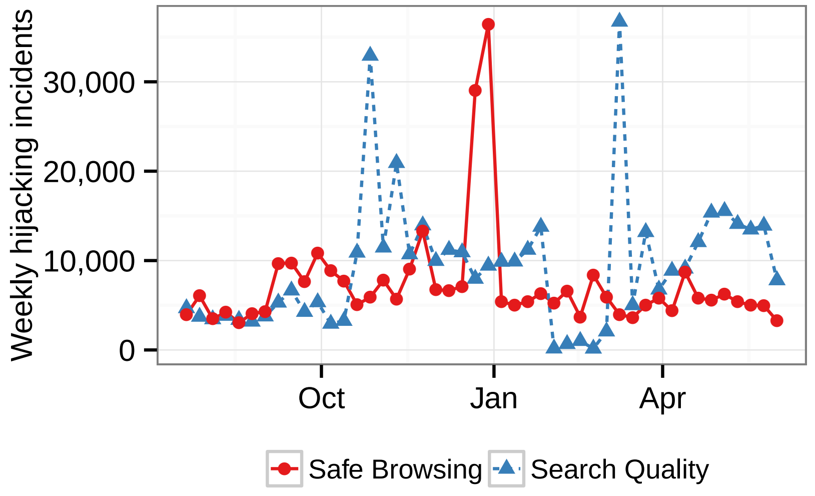


Figure 8.2: Weekly breakdown of new website hijacking incidents as detected by Safe Browsing (drive-bys) and Search Quality (scams, blackhat SEO, cloaking).

mechanisms, we lack visibility into whether webmasters observe the alert (e.g., received and read a message or viewed a browser interstitial). As such, when we measure the effectiveness of notifications, we couple both the distribution of the signal and the subsequent webmaster response.

Browser Interstitials Safe Browsing reports all compromised websites to Chrome, Safari, and Firefox users that opt for security warnings. While browser interstitials primarily serve to warn visitors of drive-by pages that cause harm, they also serve as an indirect alerting mechanism to site owners: webmasters (or their peers) that encounter warnings for their compromised sites can take action. However, this approach lacks diagnostic information about how exactly to clean up the infection.

Search Result Annotation Compromised sites detected by Search Quality receive an annotation in search result pages with a warning "This site may be hacked" [83]. Similarly, sites identified by Safe Browsing all receive a flag "This site may harm your computer" [84]. Additionally, sites may lose search ranking. As with browser interstitials, these modifications primarily serve to protect inbound visitors, but also serve as an indirect channel for alerting webmasters of compromise.

Search Console Alerts Both Safe Browsing and Search Quality provide concrete details about infected pages, examples, and tips for remediation via Google's Search Console [82]. Only webmasters who register with Search Console can view this information. Notifications come in the form of alerts on Search Console as well as a single message to the webmaster's personal email address (provided during registration). This approach alleviates some of the uncertainty of whether contacting the WHOIS `abuse@` will reach the site operator. One caveat is that a Search Console notification is sent only if the Safe Browsing or Search Quality algorithm is configured to do so.[2] This behavior results in the absence of a Search Console alert for some hijacking incidents, which we use as a natural control to measure their effectiveness.

Of hijacking incidents identified by Search Quality, 95,095 (22%) were sites where webmasters had registered with Search Console prior to infection. The same is true for 107,581 (32%) of Safe Browsing hijacking incidents. Of these, 48% of incidents flagged by Safe Browsing triggered a Search Console alert, as did 93% of incidents identified by Search Quality. Because notified and unnotified sites are flagged by different detection subsystems, differences in the subsystems may result in a biased control population. However, the bias is likely modest since the differences in detection approaches are fairly subtle compared to the overall nature of the compromise.

WHOIS Email In an attempt to contact a wider audience, Safe Browsing also emails the WHOIS admin associated with each compromised site. Since Safe Browsing simultaneously displays warnings via browser interstitials and search, we can only measure the aggregate impact of both techniques on notification effectiveness.

8.3.3 Appeals & Cleanup

Safe Browsing and Search Quality automatically detect when websites clean up by periodically scanning pages for symptoms of compromise until they are no longer present. This approach is limited by the re-scan rate of both pipelines. For Safe Browsing, sites are eligible for re-scans 14 days after their previous scan. This metric provides an accurate signal on the eventual fate of a hijacking incident, but leaves a coarse window of about 14 days during which a site may clean up but will not be immediately flagged as such. Search Quality reassesses symptoms each time Google's crawler revisits the page, enabling much finer-grained analysis windows.

As an alternative, both Safe Browsing and Search Quality provide a tool via Search Console where webmasters can appeal warnings tied to their site. Safe Browsing webmasters have an

[2]Each detection pipeline combines a multitude of algorithms, of which the majority also generate Search Console alerts.

additional channel of appealing through StopBadware [182], which submits requests for site review to Safe Browsing on behalf of the webmasters who chose not to register with Search Console. The appeals process signals when webmasters believe their pages are cleaned, which analysts at Google (Search Quality) or automated re-scans (Safe Browsing) subsequently confirm or reject. We use this dataset both to measure the duration of compromise as well as the capability of webmasters to clean up effectively, as captured by repeatedly rejected appeals.

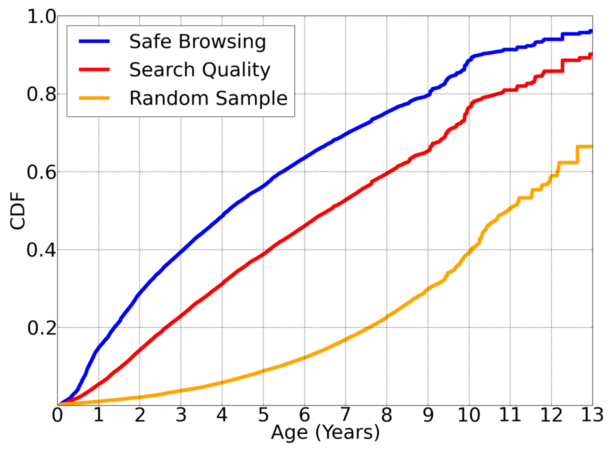
8.3.4 Limitations

We highlight and reiterate five limitations tied to our dataset. First, as a natural experiment our study lacks a true control: the systems under analysis notify all webmasters, where Google's policies arbitrate the extent of the notification. For example, browser interstitials are only shown for threats that may put site visitors at risk, namely Safe Browsing sites. Webmaster notifications via Search Console are sent only when a webmaster has registered for Search Console and the specific detection pipeline is integrated with the Search Console messaging platform. As such, we restrict our study to comparing the relative effectiveness of various notification approaches; Chapter 7 and other studies [50, 195] have already demonstrated the value of notifications over a control. Second, our coverage of hijacked websites is biased towards threats caught by Google's pipelines, though we still capture a sample size of 760,935 incidents, the largest studied to date. Third, when victims are notified, we lack visibility into whether the intended recipient witnesses the alerts. As such, when we measure the impact of notifications, we are measuring both the distribution of alerts and the ability of webmasters to take action. Fourth, the granularity at which we measure cleanup is not always per-day, but may span multiple days before a site is rechecked. This may cause us to overestimate the time a site remains compromised. We specifically evaluate compromise incidents as continuous blacklisting intervals. Finally, our study is not universally reproducible as we rely on a proprietary dataset. However, given the scale and breadth of compromised websites and notifications we analyze, we believe the insights gained are generalizable.

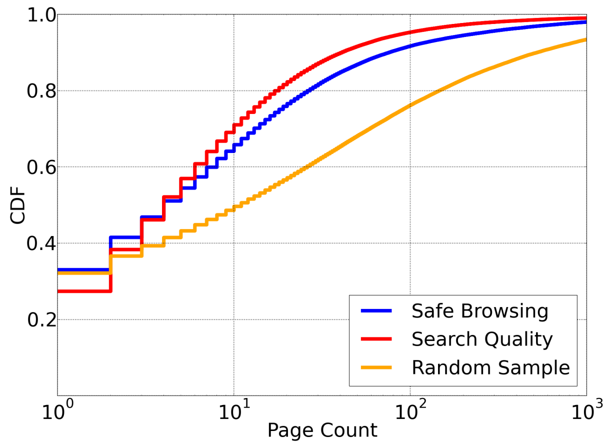
8.4 Website Demographics

As a precursor to our study, we provide a demographic breakdown of websites (e.g., domains) that fall victim to hijacking and compare it against a random sample of 100,000 non-spam sites indexed by Google Search. All of the features we explore originate from Googlebot, Google's search crawler [80]. We find that compromise affects webpages of all age, language, and search ranking, with attackers disproportionately breaching small websites over major institutions. We also observe biases in the sites that miscreants re-purpose for exploits versus scams and cloaked gateways.

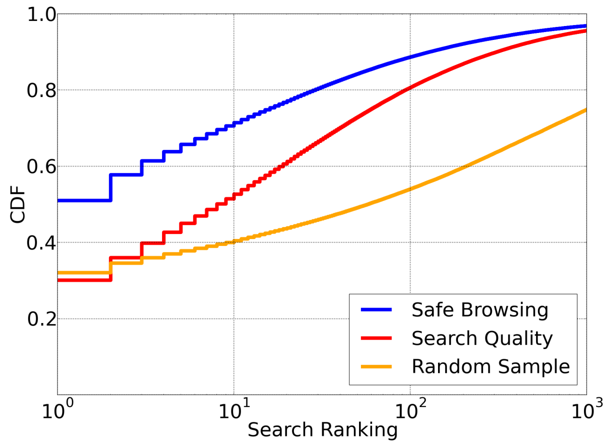
Site Age We estimate the age of a site as the time between Googlebot's first visit and the conclusion of our study in June, 2015. We detail our results in Figure 8.3a. We find over 85% of Search Quality sites are at least two years old at the time of compromise, compared to 71% of Safe



(a) Site Age



(b) Page Count



(c) Search Rankings

Figure 8.3: Demographic breakdowns of Safe Browsing and Search Quality flagged sites.

Browsing sites. Our results indicate that sites falling victim to compromise skew towards newer properties compared to the general population, more so for Safe Browsing.

Page Count We estimate a site's size as the total number of pages per domain indexed by Googlebot at the conclusion of our study. In the event a site was hijacked, we restrict this calculation to the number of pages at the onset of compromise so as not to conflate a proliferation of scam offers with prior legitimate content. Figure 8.3b shows a breakdown of pages per domain. Overall, if we consider the volume of content as a proxy metric of complexity, we observe that hijacking skews towards smaller, simpler sites compared to larger properties run by major institutions.

Search Page Rankings The search result ranking of a site represents a site's popularity, as captured by a multitude of factors including page rank and query relevance. We plot the distribution of search ranking among sites in Figure 8.3c. As an alternative metric to search ranking, we also calculate the Alexa ranking of hijacked sites. We observe that Search Quality sites skew towards higher search rankings compared to Safe Browsing, a bias introduced by cloaking and scam-based attacks targeting sites more likely to draw in visitors from Google Search. There are limits, however: we find less than 5% of compromised properties appear in the Alexa Top Million. Similarly, compared to our random sample, hijacking disproportionately impacts lowly-ranked pages. Overall, 30% of Search Quality sites and 50% of Safe Browsing sites rank low enough to receive limited search traction. This suggests that search result warnings may be ineffective for such properties due to limited visibility, a factor we explore later in § 8.5.

Language For each site, we obtain Googlebot's estimate of the site's primary language, shown in Table 8.2. We find that miscreants predominantly target English, Russian, and Chinese sites, but all languages are adversely affected. We observe a substantially different distribution of languages than the work by Provos et al., which studied exclusively malicious properties [162]. Their work found that miscreants served over 60% of exploits from Chinese sites and 15% from the United States. This discrepancy re-iterates that exploit delivery is often a separate function from web compromise, where the latter serves only as a tool for traffic generation. We observe other examples of this specialization in our dataset: 10% of Safe Browsing incidents affect Chinese sites, compared to only 1% of Search Quality incidents. Conversely, Search Quality is far more likely to target English sites. These discrepancies hint at potential regional biases introduced by the actors behind attacks.

Site Software Our final site annotation consists of the content management system (CMS) or forum software that webmasters serve content from, when applicable. Of hijacked sites, 9.1% from Safe Browsing and 13.8% from Search Quality include this annotation, compared to 10.3% of sampled sites. We present a breakdown of this subsample in Table 8.3. Wordpress and Joomla are both popular targets of hijacking, though this partly represents each system's market share. The popularity of DedeCMS and Discuz among Safe Browsing sites might be unexpected, but as

Language	Rnd. Sample	Safe Browsing	Search Quality
English	35.4%	46.9%	57.6%
Chinese	9.0%	10.0%	1.0%
German	7.2%	4.5%	2.5%
Japanese	6.0%	1.5%	4.6%
Russian	5.6%	9.9%	6.3%

Table 8.2: Top five languages for a random sample of websites versus sites falling victim to compromise. We find hijacking disproportionately impacts English, Russian, and Chinese sites.

Software	Rnd. Sample	Safe Browsing	Search Quality
WordPress	47.4%	36.9%	39.6%
Joomla	10.7%	11.6%	20.4%
Drupal	8.3%	1.7%	9.4%
Typo3	3.4%	0.5%	0.9%
Vbulletin	3.3 %	0.8%	0.4%
Discuz	1.0%	7.6%	0.4%
DedeCMS	0.2%	13.9%	1.4%

Table 8.3: Top five software platforms for hijacked websites and significant outliers, when detected. We find attacks target all of the top platforms. We note these annotations exist for only 10–13.8% of sites per data source.

Chinese platforms, this accords with the large proportion of Chinese Safe Browsing sites. Studies have shown that historically attackers prioritize attacking popular CMS platforms [196].

8.5 Notification Effectiveness

We evaluate the effect of browser warnings, search warnings, and direct communication with webmasters on remediation along two dimensions: the overall duration a site remains compromised and the fraction of sites that never clean up, despite best efforts to alert the respective webmaster. We also consider other influential factors such as whether symptoms of compromise are localized to a single page or systemic to an entire site; and whether language barriers impede effective notification. To avoid bias from webmasters with prior exposure to infection, we restrict our analysis to the first incident per site.[3] (We explore repeated hijacking incidents later in § 8.6.) We reiterate that our experiments lack a control, as outlined in § 8.3, due to our *in situ* measurements of a live

[3]We filter sites who have appeared on the Safe Browsing or Search Quality blacklists prior to our collection window. However, note it may be possible that sites were previously compromised and undetected. This filtering is best effort.

system that always sends at least some form of notification. As such, we restrict our evaluation to a comparison between notification approaches and contrast our findings with prior work.

8.5.1 Aggregate Remediation Outcomes

To start, we explore in aggregate the infection duration of all websites identified by Safe Browsing and Search Quality. We find that over our 11-month period, webmasters resolved 59.5% of hijacking incidents. Breaking this down into time windows, 6.6% of sites cleaned up within a day of detection, 27.9% within two weeks, and 41.2% within one month. Note that for the single-day cleanup rate, we exclude Safe Browsing sites that were automatically re-scanned and de-listed as we have no remediation information until two weeks after an infection begins. This does not impact manually appealed Safe Browsing incidents or any Search Quality incidents. The 40.5% of sites that remain infected at the end of our collection window have languished in that state for a median of four months, with 10% of persistent infections dating back over eight months. Our results indicate a slower remediation rate than observed by Vasek et al., who found 45% of 45 unnotified hijacked sites and 63% of 53 notified sites cleaned up within 16 days [195].

We recognize these numbers may be biased due webmasters not having enough time to redress infections occurring towards the tail end of our collection window. If we repeat the aggregate analysis only for hijacking incidents with an onset prior to January 1, 2015 (the first half of our dataset), we find the resulting cleanup rate is 7% higher, or 66.7%. Breaking this down into time windows, we find only a slight difference compared to our earlier analysis of the entire dataset: 6.4% of sites cleaned up within a day after detection, 28.2% within two weeks, and 42.1% within one month. As such, given an infection that is older than a few months, we find the likelihood of cleanup in the future tends towards zero.

8.5.2 Ranking Notification Approaches

We find that notification approaches significantly impact the likelihood of remediation. Browser warnings, in conjunction with a WHOIS email and search warnings, result in 54.6% of sites cleaning up, compared to 43.4% of sites only flagged in Google Search as “hacked”. For webmasters that previously registered with Search Console and received a direct alert, the likelihood of remediation increases to 82.4% for Safe Browsing and 76.8% for Search Quality. These results strongly indicate that having an open channel to webmasters is critical for overall remediation. Furthermore, it appears that browser interstitials (in conjunction with possible WHOIS email contact) outperform exclusive search warnings. Assuming the two compromise classes do not differ significantly in remediation difficulty, this suggests that browser warnings and WHOIS emails stand a better chance at reaching webmasters. This is consistent with our observation in § 8.4 that the majority of hijacked sites have a low search rank, impeding visibility of search warnings. Another possible explanation is that browser interstitials create a stronger incentive to clean up as Safe Browsing outright blocks visitors, while Search only presents an advisory.

For those sites that do clean up, we present a CDF of infection duration in Figure 8.4 split by the notifications sent. We omit the 37.8% sites that Safe Browsing automatically confirmed as

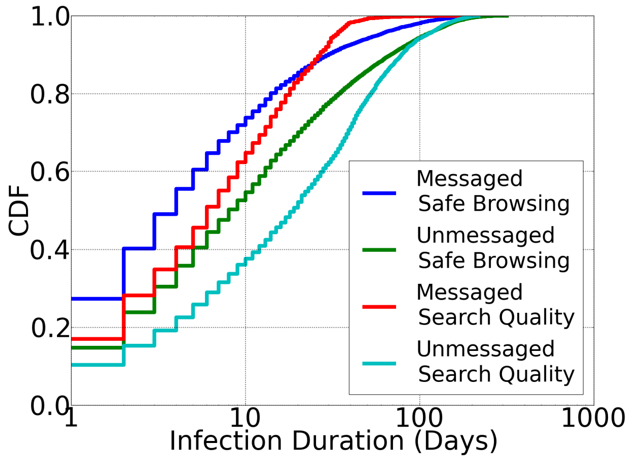


Figure 8.4: CDFs of the infection duration for Safe Browsing and Search Quality sites that eventually recover, dependent on whether they received Search Console messages or not. Direct communication with webmasters significantly reduces the duration of compromise.

cleaned from this analysis as we lack data points other than at a 14-day granularity. We note that after 14 days, 8% more manually appealed sites were cleaned compared to automatically appealed sites, indicating the latter population of site operators clean up at a slightly slower rate. As with remediation likelihood, we also observe that direct communication with webmasters decreases the time attackers retain access to a site. Within 3 days, 50% of Safe Browsing manually-appealed sites clean up when notified via Search Console, compared to 8 days in the absence of Search Console alerts. We observe a similar impact for Search Quality: 50% of webmasters resolve in 7 days, versus 18 days for unassisted sites. This indicates that incorporating direct, informative messages expedites recovery.

8.5.3 Localized vs. Systemic Infections

Given the positive influence of notifications, an important question remains whether the complexity of an infection influences the time necessary to clean up. While Search Console provides remediation tips, webmasters may naturally require more support (and detailed information) to contend with systemic infections. To assess this, we rely on detailed logs provided by Search Quality on whether harmful content for hijacked properties was systemic to an entire site or localized. In particular, Search Quality attempts to categorize incidents into three groups: *isolated incidents*, where harmful content appears in a single directory (N=91,472); *dispersed incidents* that affect multiple URL paths and subdomains (N=21,945); and *redirection incidents*, where sites become a gateway to other harmful landing pages (N=21,627).

Figure 8.5 provides a breakdown of infection duration by hijacking symptoms irrespective of whether webmasters received a Search Console alert. This measurement also includes sites that have yet to recover. We find that webmasters are both more likely and quicker to clean up isolated incidents. Of all resolved and ongoing isolated incidents, only 50% last longer than 27

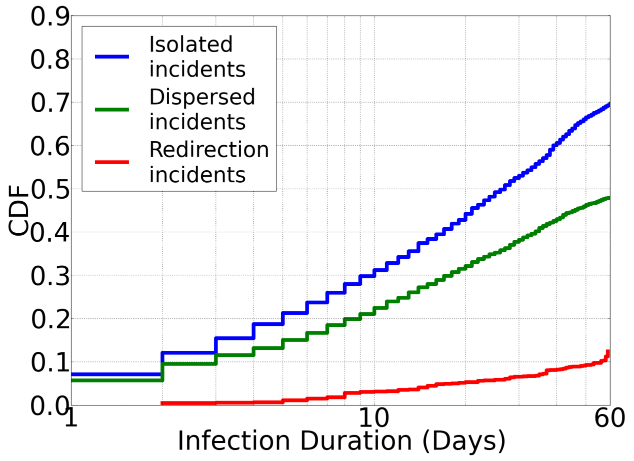


Figure 8.5: CDFs of the infection duration for different Search Quality incidents. We note curves do not reach 100% because we consider the percentage over all sites, including those not yet clean after 60 days. Cloaking (redirection) and systemic infections (dispersed) prove the most difficult for webmasters to contend with.

days. In contrast, over 50% of dispersed incidents persist for longer than 60 days. The most challenging incident type relates to redirect attacks where sites become cloaked gateways. Of these, only 12% recover within 60 days. One possible explanation for low cleanup behavior is that redirect-based attacks cloak against site operators, preventing webmasters from triggering the behavior [199]. Alternatively, redirection attacks require far less code (e.g., .htaccess modification, JavaScript snippet, meta-header) compared to attackers hosting entirely new pages and directories. As such, it becomes more difficult for webmasters to detect the modifications. In aggregate, our findings demonstrate that webmasters impacted by systemic or redirect incidents require far more effort to recover. We discuss potential aids later in § 8.7.

8.5.4 Notification Language

To avoid language barriers and assist victims of a global epidemic, Search Console supports over 25 popular languages [119]. We investigate the impact that language translations might have on recovery speeds, examining sites that received a Search Console message. Breaking down these hijacking incidents by site language reveals only relatively small differences in the fraction of sites cleaned after two weeks for the ten most popular languages. In particular, remediation rates vary between 10% for Safe Browsing and 7% for Search Quality, indicating that the message language does not drastically influence cleanup.

To evaluate the influence of language for the other notification vectors, we measure the cleanup rate for non-Search Console registrants. Table 8.4 lists the percentage of Search Quality and Safe Browsing sites recovered after two weeks. For Search Quality, remediation rates range 7%. Excluding Chinese sites, Safe Browsing sites are also similar, varying between 9%. Thus, language

Language	**Search Quality**	**Safe Browsing**
English	24.7%	29.5%
Chinese	20.1%	*2.7%*
Russian	19.4%	22.3%
Spanish	24.6%	30.4%
German	22.9%	24.6%
French	20.9%	26.9%
Italian	25.8%	29.1%
Polish	26.4%	28.9%
Portuguese (Brazil)	24.1%	27.6%
Japanese	26.1%	28.2%

Table 8.4: Top 10 languages and remediation rates for Safe Browsing and Search Quality, restricted to sites not registered with Search Console. We find consistent rates despite languages differences, with the exception of Chinese due to low Safe Browsing usage in the region.

or geographic biases do not play a major factor for browser interstitials and search warnings, with the exception of Chinese sites.

Given Chinese sites present an outlier, we explore possible explanations. Despite having the largest Internet population in the world, China ranks 45th in the number of daily browser interstitials shown, a discrepancy unseen for countries who speak the other top languages. Since we find a significant number of Chinese sites serving malware, we argue this discrepancy is due to lagging adoption of Safe Browsing in China, and potentially explains the slow Chinese remediation rate for Safe Browsing. However, it remains unclear why Chinese Search Quality sites are comparable to other languages. As there are popular search alternatives local to China, Google's search result warnings may have low visibility. Alternatively, it may be possible that Chinese sites flagged by Search Quality are a different population that those for Safe Browsing; for example, these may be Chinese language sites with a target audience outside of China and can benefit from external signals.

8.5.5 Site Popularity

We correlate a site's popularity (e.g., search ranking) with 14-day remediation rates for hijacking incidents. In particular, we sort sites based on search ranking and then chunk the distribution into bins of at least size 100, averaging the search ranking per bin and calculating the median cleanup time per bin. We present our results in Figure 8.6. We find more popular sites recover faster across both Safe Browsing and Search Quality. Multiple factors may influence this outcome. First, popular sites have a strong incentives to maintain site reputation and business. Additionally, with more visitors and a higher search ranking, it is more likely that a site visitor encounters a browser interstitial or search page warning and informs the webmaster. We find this reaches a limit, with Safe Browsing and Search Quality cleanup rates converging for search rankings greater than one

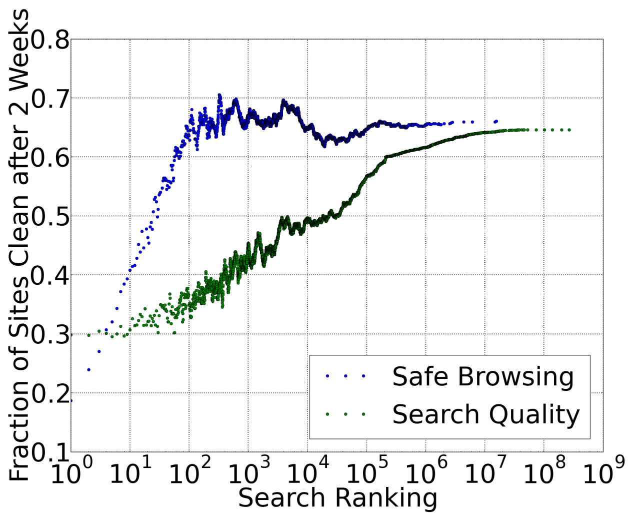


Figure 8.6: The percentages of sites clean after two weeks across different search rankings. Cleanup rate increases for higher ranked site, although the increase is stronger for Safe Browsing than Search Quality, possibly due to browser interstitials being stronger alert signals.

million. Finally, site popularity may also reflect companies with significantly higher resources and more technical administrators. Regardless the explanation, it is clear that less popular websites suffer from significantly longer infections compared to major institutions. As discussed in § 8.4, these lowly ranked sites comprise the brunt of hijacked properties.

8.5.6 Search Console Awareness

Webmasters who proactively register with Search Console may represent a biased, more savvy population compared to all webmasters. As such, we may conflate the improved performance of Search Console alerts with confounding variables. As detailed in § 8.3, not all hijacking incidents trigger a Search Console alert, even if the site owner previously registered with the service. We compare the likelihood of remediation for this subpopulation against the set of users who never register with Search Console. After two weeks, 24.5% of sites registered with Search Console and identified by Search Quality cleaned up, compared to 21.0% of non-registrants. Similarly for Safe Browsing, 28.4% of Search Console sites cleaned up compared to 24.3% of non-registrants. While Search Console registrants do react faster, the effect is small relative to the 15-20% increase in remediation rate from Search Console messaging.

8.5.7 Modeling Infection Duration

Given all of the potential factors that impact infection duration, we build a model to investigate which variables have the strongest correlation with faster recovery. We consider the following dimensions for each hijacking incident: detection source, Search Console usage, whether a Search

Console message was sent, and all of a site's demographic data including age, size, ranking, and language. For site size and ranking, we use the log base 10. We exclude a site's software platform as we lack this annotation for over 86% of sites. Similarly, we exclude systemic versus localized infection labels as these annotations exist only for Search Quality. Finally, we train a ridge regression model [95] (with parameter $\lambda = 0.1$) using 10-fold cross validation on the first hijacking incident of each site and its corresponding remediation time, exclusively for sites that clean up. Ridge regression is a variant of linear regression that applies a penalty based on the sum of squared weights, where λ is the penalty factor. This brings the most important features into focus, reducing the weights of less important features.

Overall, our model exhibits low accuracy, with an average fit of $R^2 = 0.24$. This arises due to the limited dimensionality of our data, where sites with identical feature vectors exhibit significantly different infection durations. Despite the poor fit, we find Search Console alerts, search ranking, and Search Console registration exhibit the largest magnitude weights, with weights of -10.3, -6.1, and -3.3 respectively. This suggests that receiving a Search Console alert reduces infection lengths by over 10 days on average, a stronger effect than from other factors. Interpreting these results, we argue that direct communication with webmasters is the best path to expedited recovery, while popular websites naturally benefit from increased warning visibility and potentially more technically capable webmasters. Conversely, we find other factors such as the corresponding notification's language or a site's age and number of pages do not correlate with faster recovery. We caution these are only weak correlations for a model with high error rate, but as our prior analysis shows, they have the highest discriminating power at determining the lifetime of an infection.

8.6 Webmaster Comprehension

Webmasters that receive a warning or notification must be technically savvy enough to contend with the corresponding security breach, or reach out to a third party that can help. Our dataset provides a lens into three aspects of webmaster comprehension: webmasters incorrectly requesting the removal of hijacking flags for their site when symptoms persist; sites repeatedly falling victim to new hijacking incidents in a short time window; and whether the duration that a site remains compromised improves after repeated incidents, a sign of learning over time.

8.6.1 Cleanup Attempts Before Successful

Both Search Console and Safe Browsing provide webmasters with a mechanism to manually appeal hijacking flags if webmasters believe their site is cleaned up. This triggers a re-scan or manual review by a Google analyst, after which the respective pipeline confirms a site is symptom-free or rejects the appeal due to an ongoing incident. We focus on webmaster-initiated appeals, as opposed to automated appeals that happen periodically, because the timestamp of a manual appeal denotes when a webmaster is aware their site is compromised and is taking action. Overall, 30.7% of Safe Browsing and 11.3% of Search Quality webmasters ever submit a manual appeal, of which 98.7% and 91.4% were eventually successful.

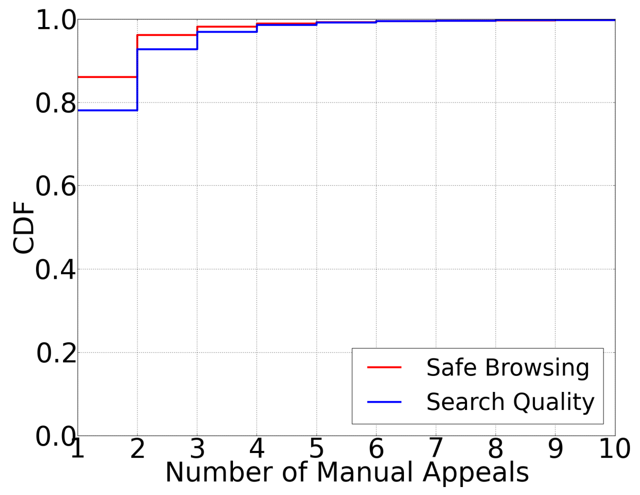


Figure 8.7: Number of manual appeals per hijacking incident. The majority of site operators successfully appeal on their first attempt.

Figure 8.7 captures the number of webmaster cleanup attempts per hijacking incident before a site was verified symptom-free. We find 86% of Safe Browsing sites and 78% of Search Quality sites successfully clean up on their first attempt, while 92% of all site operators succeed in cleaning up within two attempts. Our findings illustrate that webmasters in fact possess the technical capabilities and resources necessary to address web compromise as well as to correctly understand when sites are cleaned. However, a small portion of the webmasters struggle to efficiently deal with compromise. For both Safe Browsing and Search Quality, at least 1% of the webmasters required 5 or more appeals.

For webmasters that fail at least one appeal, we measure the total time spent in the appeals process in Figure 8.8. We find the median Safe Browsing site spends 22 hours cleaning up, compared to 5.6 days for Search Quality. The discrepancy between the two systems is partially due to Search Quality requiring human review and approval for each appeal, as opposed to Safe Browsing's automated re-scan pipeline. Another factor is the fact that webmasters addressing Search Quality incidents tend to require more appeals than Safe Browsing. Our findings illustrate a small fraction of webmasters require a lengthy amount of time to manage their way through the appeal process, with some still struggling after 2 months. In the majority of these cases, a long period of time elapses between subsequent appeals, suggesting that these users do not understand how to proceed after a failed appeal or they give up temporarily.

8.6.2 Reinfection Rates

After webmasters remedy an infection, an important consideration is whether the operator merely removed all visible symptoms of compromise or correctly addressed the root cause. Webmasters that fail to remove backdoors, reset passwords, or update their software are likely to fall victim to subsequent attacks. Along these lines, we measure the likelihood that a previously compromised

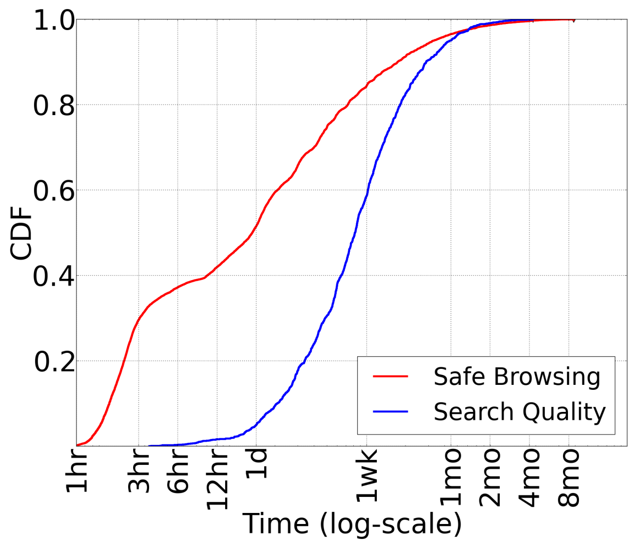


Figure 8.8: Total time webmasters spend cleaning up, exclusively for hijacking incidents where site operators submit multiple appeals.

site falls victim to a second, distinct hijacking incident. To capture this behavior, we analyze the subset of all hijacking incidents that webmasters successfully cleaned and calculate what fraction Google flagged as hijacked again within one month. Given Safe Browsing and Search Quality detection operates on a per-day granularity, our measurement requires a minimum of a one-day gap between infection incidents.

We find that 22.3% of Search Quality sites and 6.0% of Safe Browsing sites become reinfected within 30 days after their first compromise incident. Figure 8.9 provides a CDF of the time gap between infections, restricted to those sites that become reinfected. We observe attackers reinfect over 10% of Safe Browsing sites and over 20% of Search Quality sites within a single day. After 15 days, this increases to 77% of Safe Browsing sites and 85% of Search Quality sites. Some webmasters may act too hastily during the first infection incident and recover incompletely. To determine if this is a significant effect, we correlate the first infection duration with reinfection. We find minimal correlation, with a Pearson coefficient of 0.07. Our findings demonstrate a non-negligible portion of webmasters that successfully expunge hijacking symptoms fail to fully recover or improve their security practices. These oversights thrust them back into the remediation and appeals cycle yet again.

8.6.3 Learning and Fatigue

Webmasters facing reinfection can build on past experiences to recover quicker. However, they may also tire of the remediation struggles, and respond slower. We explore whether repeat victims of hijackings tend to improve (learn) or regress (fatigue) their response. A limitation is our inability to determine whether two incidents are caused by the same infection source and what actions were taken in recovery. Changes in remediation performance may be due to varying difficulties in

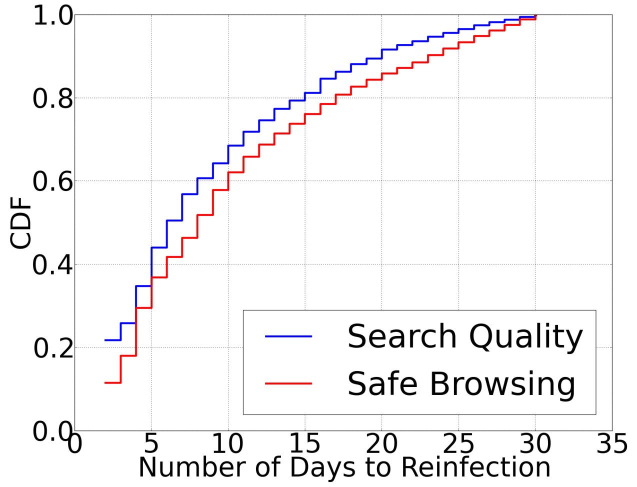


Figure 8.9: CDFs of fast reinfection times (within a month) for Safe Browsing and Search Quality sites.

addressing different infections, or conducting different cleanup procedures for the same infection source that vary in time consumption. Thus, our analysis of learning and fatigue indicates the tendency of webmasters to improve or regress their response times to subsequent infections, without identifying the causes of such changes.

For each site with multiple compromise incidents, we compute the rate at which remediation time increases or decreases for subsequent reinfections. Specifically, per site, we generate a list of infection durations, ordered by their occurrence number (e.g., first incident, second incident, etc.). For each site, we then compute the slope of the linear regression between occurrence numbers (x values) and infection durations (y values). A positive slope s indicates fatigue: each subsequent infection tends to take s days longer to redress than the previous infection. Conversely, a negative slope indicates learning: each subsequent infection lasts s days shorter.

Overall, we found that 52.7% of sites with multiple infections had a positive slope (indicating fatigue), while 36.1% had a negative slope (learning); the remaining 11.2% of sites had a slope of zero, suggesting that prior infections did not influence the remediation times of later reinfections. These results indicate that sites are more likely to fatigue than learn. Among the sites that exhibited remediation fatigue, the median increase in infection duration was 9.7 days per subsequent infection. For sites that exhibited learning, the median decrease in infection duration was 5.0 days. Thus, the effects of learning or fatiguing can be quite substantial.

8.7 Discussion

With over ten thousand weekly incidents, web compromise persists as a major problem for Internet safety. Our findings demonstrate that direct webmaster contact significantly improves remediation, but that establishing a communication channel remains a challenge. As a consequence, systems must fall back to less effective global warnings to reach a wide audience. We distill our measurements into a set of potential directions for improving detection and recovery, touching on the notion of responsible parties and user- vs. webmaster-oriented defenses.

8.7.1 Protecting Users vs. Webmasters

Safe Browsing and Search Quality both pursue a user-centric security approach that puts the safety of visitors above the interests of sites affected by hijacking incidents. Based on informal analysis of webmaster dialogues during the appeals and post-appeals process, we find that webmasters often find hijacking to be a traumatic experience. This is exasperated in part by browser interstitials and search warnings that potentially drive away visitors, reduce business, or mar site reputation. On the other hand, as we have demonstrated, these very warnings serve as the side-channels through which security services can communicate with webmasters, spurring remediation. Indeed, a majority of webmasters expressed that they were unaware of a hijacking incident until they were notified or saw a warning. Some webmasters requested that any site-level hijacking flag not take affect until one week after notification. However, such an approach both requires a direct notification channel (thus ruling out interstitials or search warnings) and also puts visitors at risk in the interim. These anecdotes highlight the duality of security when it comes to web compromise and the decision that web services must make in whose interests to prioritize.

8.7.2 Improving Remediation

Our study found that webmasters can significantly benefit from early notification of infections, but that webmasters fail to redress 40.5% of hijacking incidents. We offer three approaches for improving overall cleanup rates: increasing webmaster coverage, providing precise infection details, and equipping site operators with recovery tools.

Notification Coverage Our findings showed that direct Search Console notifications outperformed global warnings, but that only 22% of Search Quality incidents and 32% of Safe Browsing incidents had a corresponding webmaster registered with Search Console. This stems in part from the requirement that webmasters must both know about Search Console and proactively register an account. While email may seem like an attractive alternative channel, we argue that identifying a point of contact remains the most significant hurdle. The study in Chapter 7, as well as other works [50, 195], relied on WHOIS abuse contacts, though it is unclear what fraction of recipients received or opened the notification. For our study, we were unable to independently assess the efficacy of Safe Browsing's email notifications due to their tight coupling with browser interstitials. However, given that webmasters cleaned up 51% more Safe Browsing incidents when a Search Console email was triggered versus not, it is clear that WHOIS-based emails often fall into the void. While hosting providers are also well-positioned, the lack of a uniform protocol or API to alert hosted sites remains a barrier to adoption. Instead, we argue that notifications should expand to services with broader reach among webmasters such as social networks and analytics or ads platforms.

Notification Content A common theme among webmaster appeals was the desire for notifications with more detailed information about precisely what pages served harmful content. As our analysis of appeals found, 14–22% of webmasters lacked sufficient expertise or example code to clean up on their first attempt. As Vasek et al. previously showed, including more detailed notifications expedites remediation [195]. Potential improvements might include screenshots of rogue pages, a tool for accessing a crawler's perspective of injected content, or more detailed diagnostic information. Alternatively, in the absence of a direct notification, browser interstitials and search warnings could include information targeted directly at webmasters rather than merely visitors. This is a tacit recognition that global warnings play a key role in recovery. That said, detailed logs may steer webmasters towards addressing symptoms of compromise rather than the root cause, yielding an increase in repeated hijacking incidents. We investigate notification content further in subsequent chapters.

Recovery Tools While our findings demonstrate that many webmasters successfully recover, we are aware of few tools that help with the process. Such tools would decrease the duration of compromise and likelihood of reinfection. However, a question remains whether such tools generalize across attacks or not. Soska et al. found that attackers commonly exploit the same vulnerable software [177]. As such, it may be better to proactively notify sites of outdated software rather than wait till a hijacking incident, side-stepping the challenge of cleaning up specialized payloads.

8.7.3 Responsible Parties

The final challenge we consider is who should assume the responsibility for driving remediation. Site operators are best positioned to redress hijacking incidents, but our and prior work has shown that webmasters are often unaware their site is compromised until an outside alert. Alternatively, hosting providers own the serving infrastructure for compromised sites, of which security scanning could be a service. However, doing so comes at a financial cost or technical burden; today, few providers scan for harmful content or vulnerabilities [44]. Finally, ISPs, browser vendors, and search engine providers can enact incentives to spur action, but ultimately they possess limited capacity to directly help with remediation. These factors—representing the decentralized ideals of the Internet—make web compromise more challenging to address than account or credit card breaches where a centralized operator responds. The security community must determine which parties should bear the most responsibility for attending to compromised sites; what actions the parties should and should not pursue; and which mechanisms to employ to hold each accountable. Until then, compromise will remain an ongoing problem.

8.8 Conclusion

In this chapter, we explored the influence of various notification techniques on remediation likelihood and time to cleanup. Our results indicate that browser interstitials, search warnings, and direct communication with webmasters all play a crucial role in alerting webmasters to compromise and spurring action. Based on a sample of 760,935 hijacking incidents from July, 2014–June, 2015, we found that 59.5% of notified webmasters successfully recovered. Breaking down this aggregate behavior, we found Safe Browsing interstitials, paired with search warnings and WHOIS emails, resulted in 54.6% of sites cleaning up, compared to 43.4% of sites flagged with a search warning alone. Above all, direct contact with webmasters increased the likelihood of remediation to over 75%. However, this process was confounded in part by 20% of webmasters incorrectly handling remediation, requiring multiple back-and-forth engagement with Safe Browsing and Search Quality to re-establish a clean site. Equally problematic, a sizeable fraction of site owners failed to address the root cause of compromise, with over 12% of sites falling victim to a new attack within 30 days. To improve this process moving forward, we highlighted three paths: increasing the webmaster coverage of notifications, providing precise infection details, and equipping site operators with recovery tools or alerting webmasters to potential threats (e.g., outdated software) before they escalate to security breaches. These approaches address a deficit of security expertise among site operators and hosting providers. By empowering small website operators—the largest victims of hijacking today—with better security tools and practices, we can prevent miscreants from siphoning traffic and resources that fuel even larger Internet threats.

Chapter 9

Exploring Effective Vulnerability Notifications

9.1 Introduction

In Chapters 7 and 8, we discussed studies that revealed the effectiveness of security notifications across distinct security contexts. While the later work (Chapter 8) began exploring factors that influenced the impact of a notification campaign, it analyzed a live system *in situ* and thus did not execute fully controlled experiments. In this study, we perform Internet-scale multivariate controlled experiments to systematically explore not only if and when Internet-scale outreach is effective, but how best to conduct these efforts. In particular, we address (1) who to notify (e.g., WHOIS contacts versus national CERTs versus US-CERT), (2) the role of notification content (e.g., do reporters need to devise detailed messages or do short ones suffice), (3) the importance of localization (e.g., what role does native language play in notification response rates), and (4) how these considerations vary with the nature of the vulnerability (including whether for some vulnerabilities notification appears hopeless).

We evaluate these questions empirically in the context of notification campaigns spanning three different vulnerability categories: publicly accessible industrial control systems, misconfigured IPv6 firewalls, and DDoS amplifiers. Using large-scale Internet scanning to identify vulnerable hosts and then monitor their behavior over time post-notification, we infer the effects of different notification regimes as revealed by the proportion and timeliness of contacts remediating their vulnerable hosts.

Our results indicate that notifications can have a significant positive effect on patching, with the best messaging regimen being directly notifying WHOIS contacts with detailed information within the message itself. An additional 11% of contacts addressed the security issue when notified in this fashion, compared to a control. However, we failed to push the majority of contacts to take action, and even when they did, remediation was often only partial. Repeat notifications did not further patching. We additionally characterize the responses we received through our notification campaigns, of which 96% of human-sent responses were positive or neutral. Given these promising

Dataset	Hosts	WHOIS Abuse Contacts	Hosts with WHOIS Contacts
ICS	45,770	2,563	79.7%
IPv6	180,611	3,536	99.8%
Ampl.	83,846	5.960	92.4%

Table 9.1: **Vulnerable Hosts**—We notified network operators about three classes of vulnerabilities found in recent studies: publicly accessible industrial control systems (ICS), hosts with misaligned IPv4 and IPv6 firewall policies, and DDoS amplifiers (NTP, DNS, and Chargen).

yet modest findings, it behooves the security community to more deeply investigate vulnerability notifications and ways to improve their efficacy. Our methodology and results form the basis for establishing initial guidelines to help drive future efforts.

The work in this chapter appeared at the USENIX Security Symposium [112].

9.2 Methodology

To measure notification efficacy and to understand how to construct effective notifications, we notified network operators while varying aspects of the notification process. In this section, we detail the datasets of vulnerable hosts, the variables we tested, and how we tracked remediation.

9.2.1 Vulnerable Hosts

We notified operators about the three classes of vulnerabilities listed below. We show the population of vulnerable hosts in Table 9.1.

Publicly Accessible Industrial Control Systems Industrial control systems (ICS) are pervasive and control physical infrastructure ranging from manufacturing plants to environmental monitoring systems in commercial buildings. These systems communicate over a myriad of domain and manufacturer specific protocols, which were later layered on Ethernet and TCP/IP to facilitate long distance communication. Never designed to be publicly accessible on the Internet, these protocols lack important security features, such as basic authentication and encryption, but nonetheless are frequently found unsecured on the public Internet. To identify vulnerable ICS devices, Mirian et al. extended ZMap [66] and Censys [63] to complete full IPv4 scans for several ICS protocols: DNP3, Modbus, BACnet, Tridium Fox, and Siemens S7 [132]. In total, they found upwards of 46 K ICS hosts that were publicly accessible and inherently vulnerable.

We coordinated with Mirian et al. to complete daily scans for each protocol against the public IPv4 address space from January 22–24, 2016. We limited our study to the 45.8 K hosts that were present all three days to reduce the noise due to IP churn. To track the impact of our notifications, we continued the daily scans of these hosts using the same methodology.

Misconfigured IPv6 Firewall Policies Czyz et al. found that 26% of IPv4/IPv6 dual-stack servers and routers have more permissive IPv6 firewall policies compared to IPv4, including for BGP, DNS, FTP, HTTP, HTTPS, ICMP, MySQL, NTP, RDP, SMB, SNMPv2, SSH, and Telnet access [57]. For example, twice as many routers have SSH accessible over IPv6 compared to IPv4. Given the presumed rarity of IPv6-only services, this likely indicates a misconfiguration and potential security issue.

To identify dual-stack servers, Czyz et al. looked for hostnames in the Rapid7 DNS ANY dataset [167] that had both A and AAAA records. After filtering out automatically generated hostnames, they identified 520 K dual-stack servers. To find routers, the team performed reverse DNS lookups and subsequent A and AAAA lookups for hosts in the CAIDA Ark dataset [43], identifying 25 K routers. Czyz et al. then scanned these hosts using Scamper [118] to identify firewall inconsistencies.

We scanned the hosts that Czyz et al. identified over a 25 day period from December 31, 2015 to January 24, 2016. We limited our study to the 8.4 K routers and 172.2 K servers that were consistently available during that period. Similar to the ICS measurements, we continued to perform daily scans using the same methodology to track the impact of our notifications.

DDoS Amplifiers Several UDP protocols allow attackers to launch distributed denial of service attacks when improperly configured [170]. In this scenario, an attacker spoofs a small request to a misconfigured server, which then sends a large response to the victim. For example, an attacker can spoof a DNS lookup to a recursive DNS resolver, which will then send the full recursive lookup to the victim's machine. We identified 152 K misconfigured hosts that were actively being used to launch DDoS attacks over NTP, DNS, and Chargen by monitoring the sources of DDoS attacks against a university network between December 11–20, 2015.

We restricted our notifications to the vulnerable hosts that were consistently available during our daily scans from December 21, 2015 to January 26, 2016. In total, we discovered 5.9 K Chargen amplifiers, 6.4 K NTP amplifiers, and 71.5 K DNS amplifiers on 83.8 K distinct IP addresses. We continued to track these hosts by performing daily protocol scans (e.g., Chargen requests, NTP monlist commands, and DNS recursive lookups).

In each case, we coordinated with the studies' authors to ensure that they did not simultaneously notify operators. However, we do note that groups have previously sent notifications to DDoS amplifiers [110].

9.2.2 Experiment Variables

To understand how to best construct and route notification messages, we performed notifications using several methodologies and measured the differences in remediation. We specifically aimed to answer the following questions:

Who should researchers contact? Researchers have several options when deciding where they should report vulnerabilities, including directly contacting network operators, notifying national

CERTs, and asking their own country's CERT to disseminate the data to other CERT groups. We tested three options: (1) notifying the abuse contact from the corresponding WHOIS record, (2) geolocating the host and contacting the associated national CERT, and (3) asking our regional CERT (US-CERT) to propagate the information.

How verbose do messages need to be? It is not clear how much information researchers need to include when notifying operators. For example, are notifications more effective if researchers include detailed remediation steps or will such instructions go unheeded? We sent three types of messages: (1) a terse message that briefly explained that we discovered the vulnerability with Internet-wide scanning, and the impact of the vulnerability (e.g., for ICS notifications, we wrote "These devices frequently have no built-in security and their public exposure may place physical equipment at risk for attack."), (2) a terse message with a link to a website with detailed information, and (3) a verbose email that included text on how we detected the problem, vulnerability details, and potential remediation steps. We provide the full text of our different messages in Appendix B.2–B.7.

Do messages need to be translated? We tested sending messages in English as well as messages translated by native technical speakers to several local languages.

9.2.3 Group Assignment

To test the impact of our experiment variables, we randomly formed experiment groups that received different notification regimens. Here we describe our process for constructing these groups.

For each IP address, we extracted the abuse contact from the most specific network allocation's WHOIS record. For the 16.7% of dual-stack hosts with different contacts extracted from IPv4 and IPv6 WHOIS records, we used the contact with the deepest level of allocation, and preferred IPv6 contacts when all else was equal (4.3% of dual-stack hosts).

To test each variable, we split the abuse contacts from each vulnerability into treatment groups (Table 9.2). For the ICS and amplifier experiments, we randomly allocated one quarter of abuse contacts to the control group (Group 1), one quarter to the CERT groups (half US-CERT, half national CERTs), and the remaining half to the WHOIS groups. For IPv6, to act in a responsible manner we needed to complete *some* form of notification for all hosts to ensure adequate disclosure prior to the release of the corresponding study [57] in February 2016. This prevented us from using a true control group. Instead, we approximate the behavior of the control group using the 25 days of daily scans prior to our notifications. We allocated a third of the IPv6 contacts to the CERT groups, and the remainder to the WHOIS groups.

For the vulnerable hosts assigned to the CERT groups, we geolocated each IP using MaxMind [126] and identified the associated CERT. We note that not all countries have an established CERT organization. This was the case for 2,151 (17%) IPv6 hosts, 175 (8%) ICS devices, and 2,156 (19%) DDoS amplifiers. These hosts were located in 16 countries for IPv6, 26 countries for ICS, and 63 countries for DDoS. Many of these countries are in Africa or Central America (e.g.,

Group	**ICS**	**IPv6**	**Ampl.**
Control	657	3,527	1,484
National CERTs	174	650	379
US-CERT	493	578	1,128
WHOIS: English Terse	413	633	777
WHOIS: English Terse w/ Link	413	633	777
WHOIS: English Verbose	413	632	777
WHOIS: Language – Terse			
Germany: German		71	
Germany: English		72	
Netherlands: Dutch		32	
Netherlands: English		32	
Poland: Polish			37
Poland: English			37
Russia: Russian			123
Russia: English			123
WHOIS: Language – Verbose			
Germany: German		70	
Germany: English		72	
Netherlands: Dutch		32	
Netherlands: English		29	
Poland: Polish			36
Poland: English			36
Russia: Russian			123
Russia: English			123

Table 9.2: **Notification Groups**—We aggregated vulnerable hosts by WHOIS abuse contacts and randomly assigned these contacts to notification groups. Here, we show the number of contacts notified in each group. Note that for the language experiments, we tested terse and verbose messages for several countries, both translated and in English.

Botswana, Ethiopia, and Belize), or are smaller island states (e.g., American Samoa, Antigua and Barbuda, and the Bahamas). We did not include hosts without a CERT organization in the CERT experiment (although we later passed them along to US-CERT).

In total, 64 CERTs were responsible for IPv6 hosts, 57 for ICS, and 86 for amplifiers. To compare directly contacting national CERTs versus having US-CERT distribute information to them, we randomly divided the affected national CERTs into two halves. For national CERTs in the first half, we contacted them directly with vulnerable hosts in their region (Group 2). We sent the remaining hosts for CERTs in the second half to US-CERT (Group 3).

We obtained native translations of our WHOIS messages for several countries. We allocated contacts in the WHOIS groups that were in those countries (based on the WHOIS records) for our language experiment, further detailed in Section 9.3.3. The remaining contacts were randomly split into three groups based on message verbosity: terse (Group 4), terse with a link (Group 5), and verbose (Group 6).

9.2.4 Notification Process

We sent notification emails with the FROM and REPLY-TO header set to an institutional mailing list: *security-notifications@berkeley.edu*. In each message, we attached a CSV file that contained the list of vulnerable hosts along with the latest scan timestamp and the list of vulnerable protocols. We also included a link to an anonymous survey, which asked for the organization's perspective on the reported security issue and whether they found our detection and notification acceptable. The messages were sent from a server in UC Berkeley's network, which was listed as a valid mail server by UC Berkeley's SPF policy. We note that we also included a randomly generated identifier in each email subject that enabled us to match a reply to the original notification.

9.2.5 Tracking Remediation

We tracked the impact of different notification methodologies by scanning all hosts for several weeks following our notifications. As our scanning methods tested the reachability of several services, we may have falsely identified a host as patched due to random packet loss or temporary network disruptions. To account for this, we only designated a host as patched if it did not appear vulnerable in any subsequent scans. We leveraged the last day's scan data for this correction, but did not otherwise use it in our analysis as it lacked subsequent data for validation.

One limitation in our tracking is the inability to distinguish true patching from network churn, where the host went offline or changed its IP address. While we can still conduct a comparative analysis against our control group, we acknowledge that our definition of patching is a mixture of true patching and churn. We investigated whether we could better approximate true remediation by distinguishing between RST packets and dropped packets. We compared the proportion of RSTs and drops between our control group and our notified groups two days after notification and two weeks after notification. At both times, we observed nearly identical proportions between the control and notified groups—in all cases less than 20% of hosts sent RST packets. This indicates that RST packets are not a reliable signal for remediation, as most hosts did not send RST packets even when truly fixed.

Unless stated otherwise, we consider a host as having taken remediation steps for a particular vulnerability if any of its affected protocols were detected as fixed. Likewise, we say a notification contact has taken remediation steps if any of its hosts have patched. We define the remediation rate as the percentage of notification contacts that have taken remediation steps. This definition is over contacts rather than hosts as we are measuring the impact of notifying these contacts, and contacts differ in the number of affected hosts.

9.2.6 Ethical Considerations

We followed the guidelines for ethical scanning behavior outlined by Durumeric et al. [66]: we signaled the benign intent of our scans through WHOIS entries and DNS records, and provided project details on a website on each scanning host. We respected scanning opt-out requests and extensively tested scanning methods prior to their deployment.

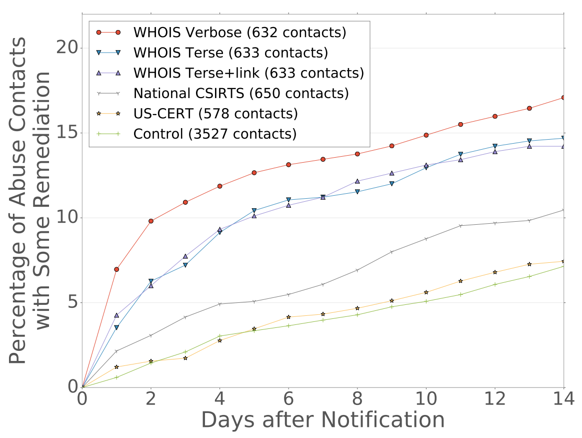
The ethics of performing vulnerability notifications have not been widely discussed in the security community. We argue that the potential good from informing vulnerable hosts outweighs the risks. To minimize potential harm, we only contacted abuse emails using addresses available in public databases. Additionally, we messaged all unnotified contacts at the conclusion of the study. We offered a channel for feedback through an anonymous survey with questions about the notified organization (described in Appendix B.1). We note that because we only collected data about organizational decisions and not individuals, our study did not constitute human subjects research (confirmed by consulting the UC Berkeley IRB committee). Nevertheless, we followed best practices, e.g., our survey was anonymous and optional.

9.3 Results

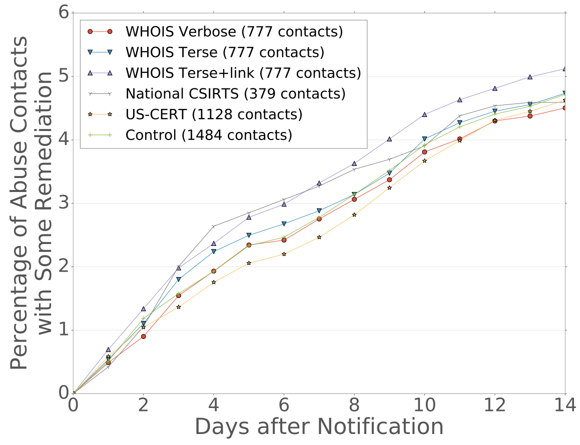
For both ICS and IPv6, our notifications had a significant impact on patch rates. In our most successful trial—verbose English messages sent directly to operators—the patch rate for IPv6 contacts was 140% higher than in the control group after two weeks. For ICS, the patch rate was 200% higher. However, as can be seen in Figure 9.1b, none of our notifications had significant impact on DDoS amplifiers. This is likely due to the extensive attention DDoS amplifiers have already received in the network operator community, including several prior notification efforts [170]. In addition, these amplifiers were already previously abused in DDoS attacks without administrative responses, potentially indicating a population with poor security stances. It is also important to note that our best notification regimen resulted in at most 18% of the population remediating. Thus, while notifications can significantly improve patching, the raw impact is limited. In the remainder of this section, we discuss the impact of each experiment variable and how this informs how we should construct future notifications.

To characterize the performance of our trial groups, we measure the area under the survival curve for each group, which captures the cumulative effect of each treatment. To determine if observed differences have statistical significance, we perform permutation tests with 10,000 rounds. In each round, we randomly reassign group labels and recompute the area differences under the new assignments. Intuitively, if the null hypothesis is true (i.e., no significant difference between two groups), then this random reassignment will only reflect stochastic fluctuation in the area difference. We assess the empirical probability distribution of this measure after the permutation rounds, allowing us to determine the probability (and significance) of our observations.

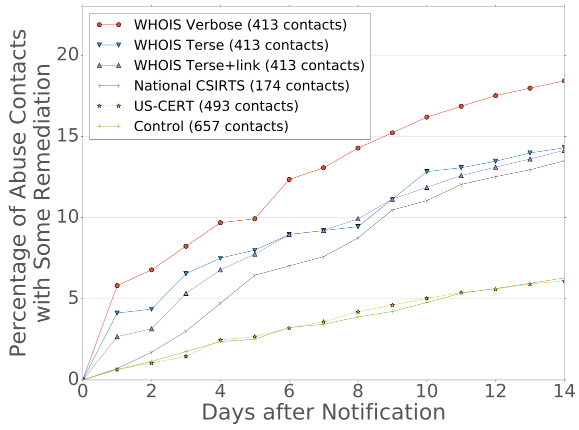
All reported p-values are computed via this permutation test. We use a significance threshold of $\alpha = 0.05$, corrected during multiple testing using the simple (although conservative) Bonferroni correction, where each test in a family of m tests is compared to a significance threshold of $\frac{\alpha}{m}$.



(a) Misconfigured IPv6



(b) DDoS Amplifiers



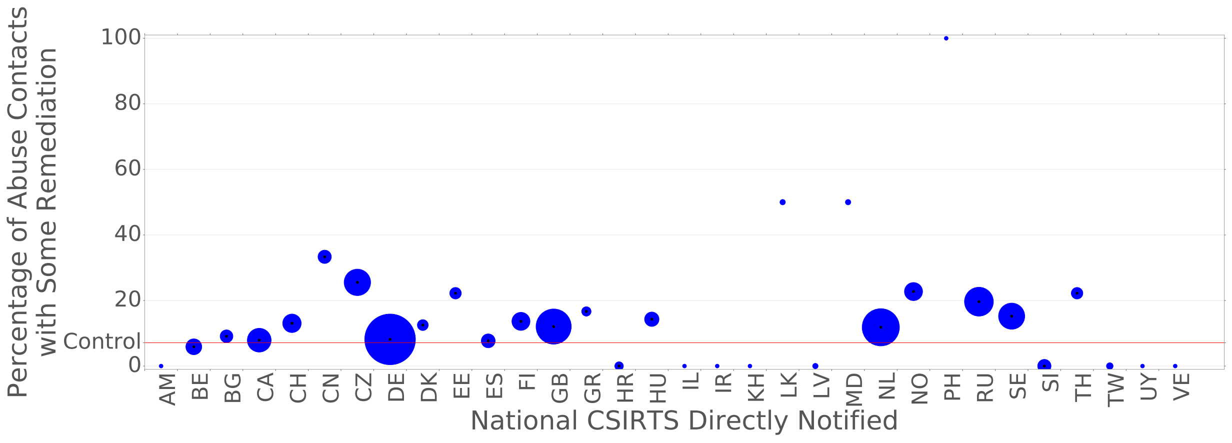
(c) ICS Services

Figure 9.1: **Remediation Rates**—We show the remediation rate for each variable we tested. We find that verbose English notifications sent to network operators were most effective for IPv6 and ICS. Note the varying Y axes.

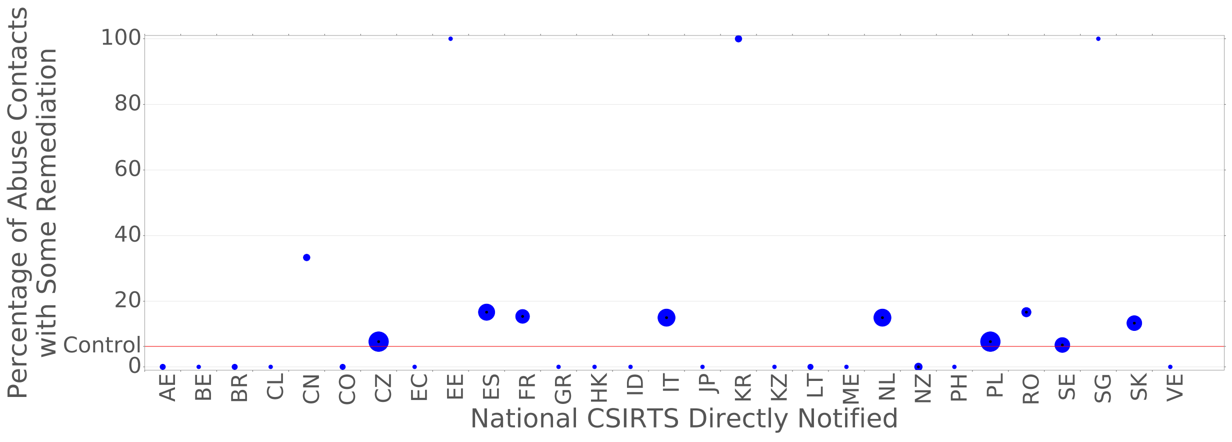
Ideally, we would have selected this procedure as part of our original experimental design. Unfortunately, we only identified its aptness *post facto*; thus, its selection could introduce a selection bias, a possible effect that we lack any practical means to assess.

9.3.1 Notification Contact

For both IPv6 and ICS notifications, directly notifying WHOIS abuse contacts was most effective—particularly early on. Two days after IPv6 disclosure, direct verbose notifications resulted in 9.8% of the population remediating, compared to 3.1% when contacting national CERTs and 1.4% by contacting US-CERT. For ICS, direct notifications promoted 6.8% of the population to patch, more than national CERTs (1.7%) and US-CERT (1.0%). In both cases, direct notifica-



(a) Misconfigured IPv6



(b) ICS Services

Figure 9.2: **Differences between National CERTs**—We show the remediation rate for each directly notified national CERT after two weeks. The size of a data point is proportional to the number of abuse contacts in the country. We directly contacted 32 CERTs for IPv6, and 29 CERTs for ICS. We observe notable differences between CERT groups. However, none are statistically significantly different than the control group. This may be because there are too few hosts for some countries, and that the Bonferroni correction is conservative.

tions were notably better than no notifications. As can be seen in Figures 9.1a and 9.1c, this gain was persistent. After two weeks, the patch rate of directly notified IPv6 contacts was 2.4 times as high as the control, and three times as high for ICS contacts.

To determine if these observations are statistically significant, we perform permutation tests using the Bonferroni correction. With six treatment groups, the family of pairwise comparisons includes 15 tests, giving an individual test threshold of $\alpha = 0.0033$. Under the permutation test, the gains that direct verbose notifications had on the CERTs and the control group are statistically significant for both IPv6 and ICS, with all p-values less than 0.0001 except when comparing ICS verbose notifications with national CERTs ($p = 0.0027$).

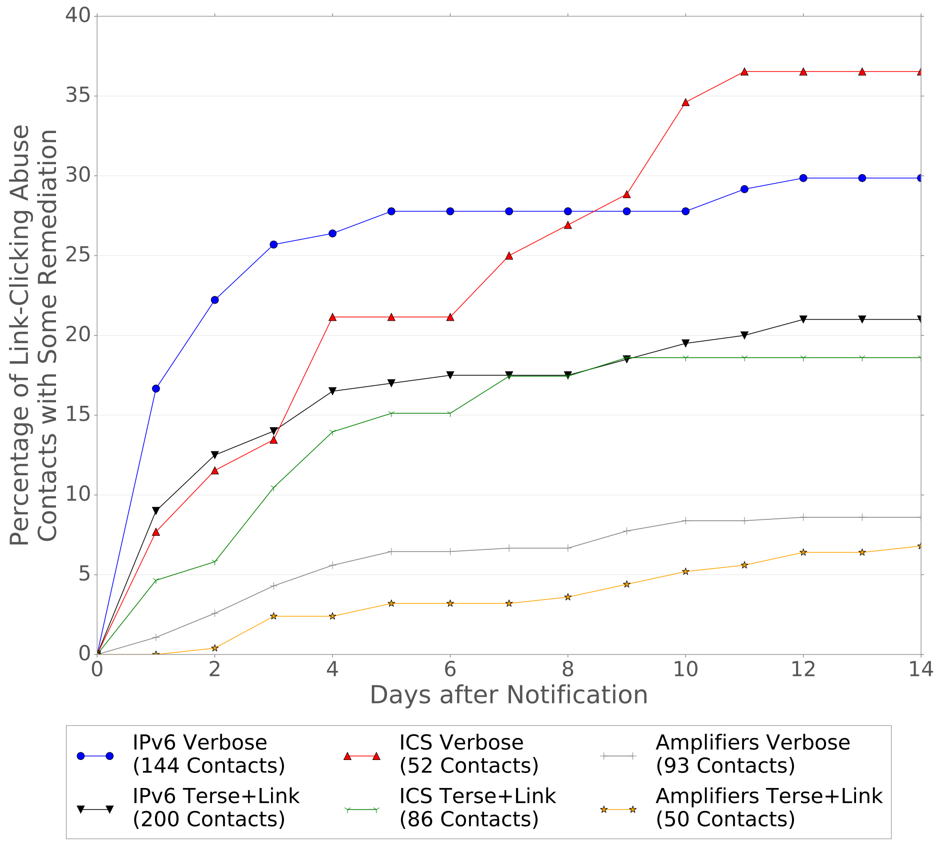


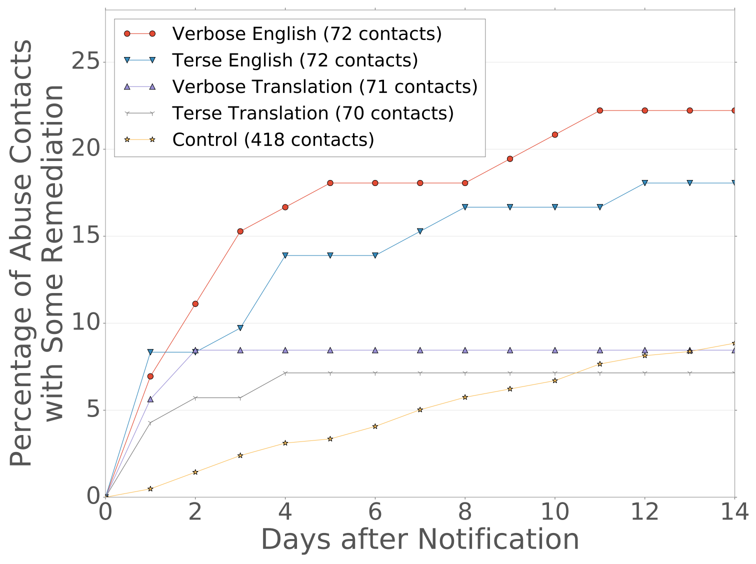
Figure 9.3: **Remediation Rates for Website Visitors**—The contacts who viewed our informational website remediated at a higher rate than those who received a verbose message. However, despite this, less than 40% of the contacts who visited the site fixed the vulnerability.

Notably, US-CERT—our local CERT who we asked to disseminate data to other CERT groups—had the lowest patch rate, which is statistically indistinguishable from the control group that had no notifications. We suspect that US-CERT did not disseminate the data to any other CERT groups or notify any US operators. One national CERT included in the report to US-CERT informed us they had not received any notices from US-CERT. As seen in Figure 9.2, there were stark differences between CERT groups—some duly notified operators, while others appear to have ignored our disclosures.

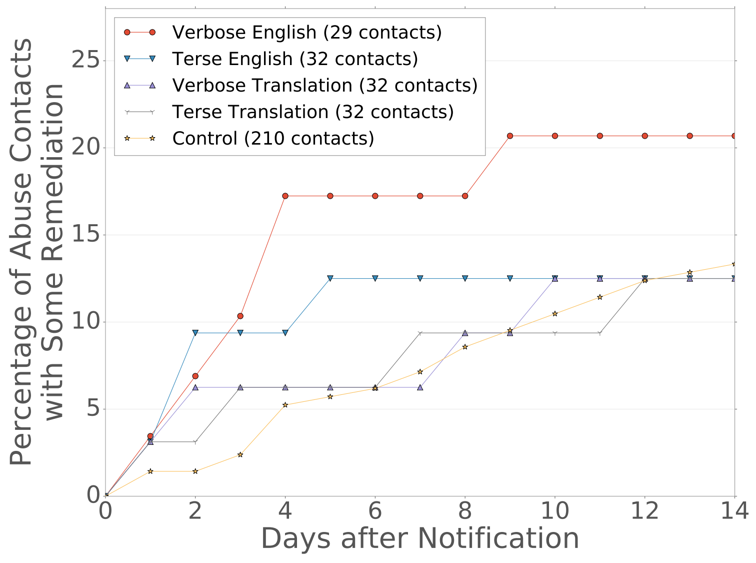
Overall, this suggests that the most effective approach—in terms of both the number of hosts patched and the rate of patching—is to directly notify network operators rather than contact CERT groups.

9.3.2 Message Verbosity

To determine what information needs to be included in notification messages, we sent three types of emails: (1) verbose, (2) terse, and (3) terse with a link to a website with additional details.



(a) Misconfigured IPv6 - German Contacts

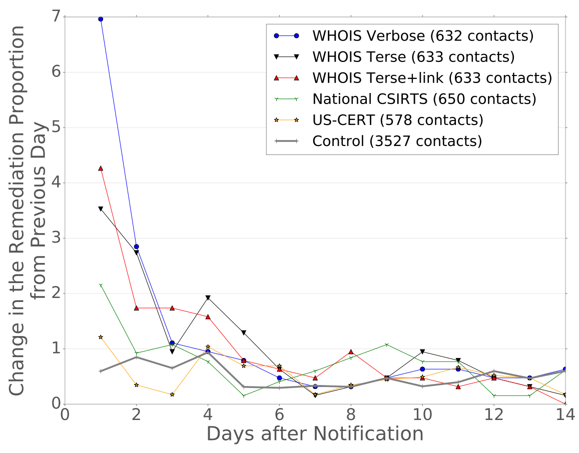


(b) Misconfigured IPv6 - Dutch Contacts

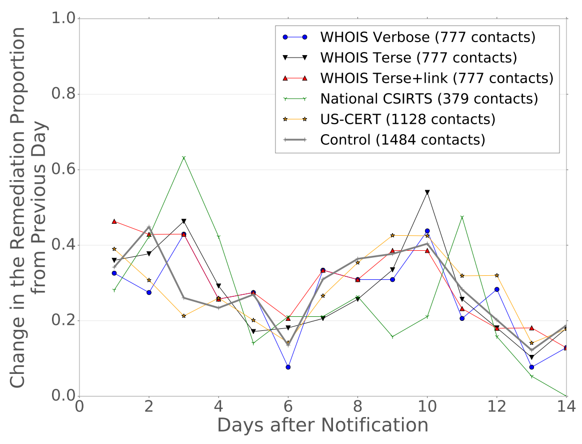
Figure 9.4: **Remediation Rates for Translated Messages—**We find that sending verbose English messages was more effective than translating notifications into the language of the recipient. Note, though, that this observation is limited to the small set of languages we were able to evaluate.

We observed the best remediation by contacts who received verbose messages. For IPv6, verbose messages were 56.5% more effective than either terse messages after two days and 55.5% more effective for ICS. However, as can be seen in Figure 9.1, the differences between verbose and terse messages decreased over time.

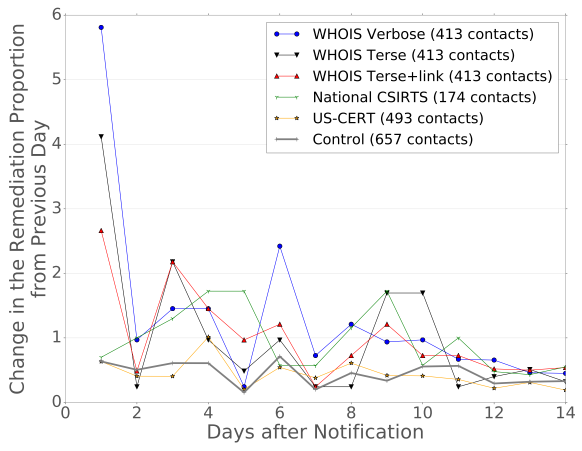
Using permutation testing and the Bonferroni correction, we find that the differences between



(a) Misconfigured IPv6



(b) DDoS Amplifiers

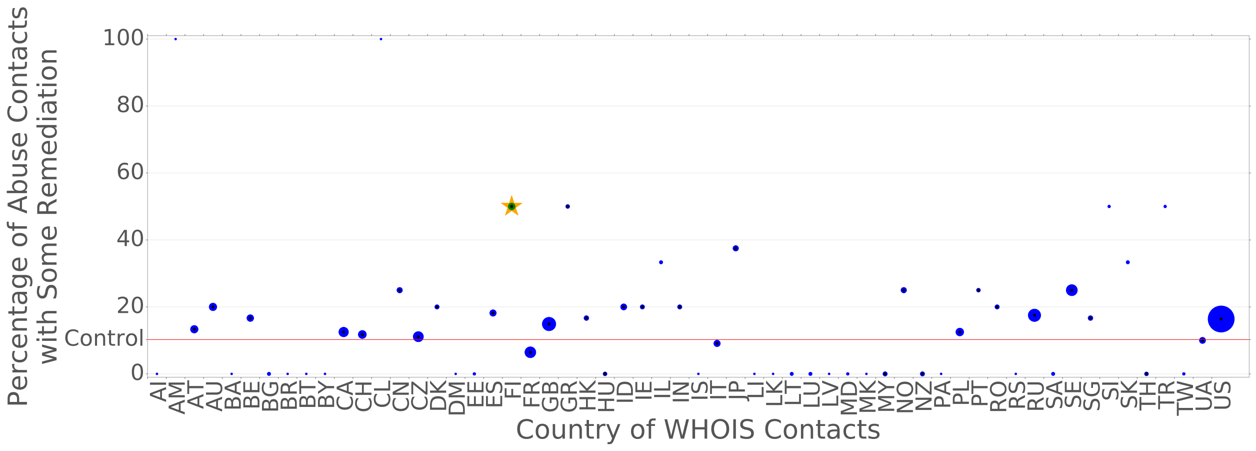


(c) ICS Services

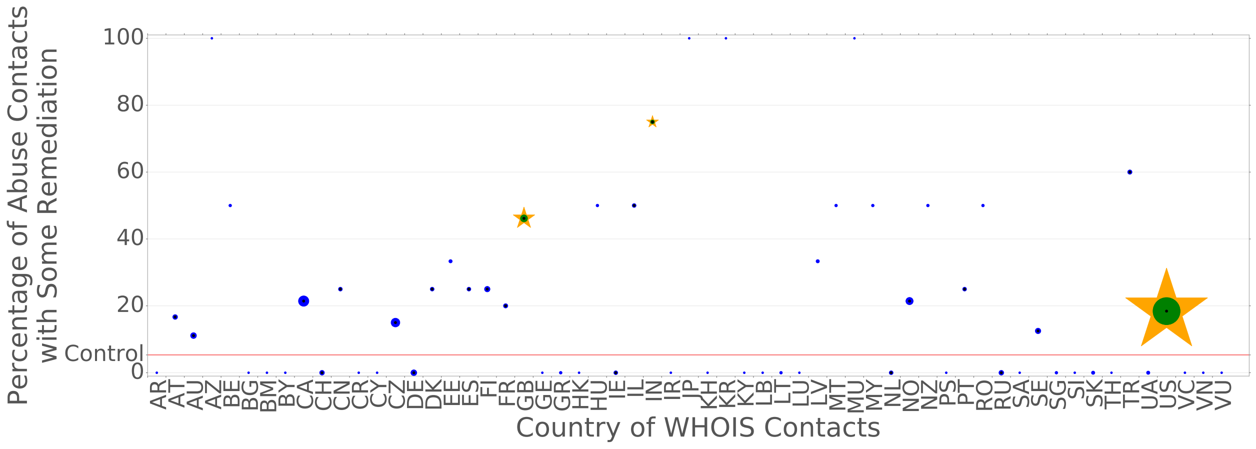
Figure 9.5: **Daily Changes in Remediation Proportions—**We show the differences in the proportions of remediated contacts from one day to the next. We find that most contacts that remediated fixed the problem immediately after disclosure. After a few days, contacts returned to remediating at the same rate as the control group.

the message types are not statistically significant for IPv6 and ICS. However, given the earlier benefits that verbose messages had for both data sets, we argue notifiers may still want to prefer verbose messages over terse ones. We discuss this effect further in Section 9.3.4 and note that further investigation of this variable is warranted.

We tracked the remediation rate of contacts who visited the linked website, as shown in Figure 9.3. We note that all of the information included in the verbose message was available on the linked website and that 16.8% of users who received an email with a link visited the site. This indicates that a sizable population of users engaged with our site, but many would not patch even after visiting the link. Specifically, no more than 40% of website visitors patched. Thus, even when our messages successfully reached contacts, the majority did not take action.



(a) Misconfigured IPv6



(b) ICS Services

Figure 9.6: **Contact Remediation per Country**—We show the percentage of contacts who remediated per country after two weeks. The data sizes are proportional to the number of contacts. Green data points surrounded by an orange star signify countries with a remediation rate statistically better than the control group's, under the permutation test using the Bonferroni correction.

9.3.3 Message Language

To investigate whether notifications need to be translated into recipients' local languages or can be sent in English, we distributed translated messages for two countries for DDoS and IPv6 notifications. For DDoS amplifiers, we obtained native Russian and Polish translations—for the countries with the third and fourth largest number of vulnerable organizations. For IPv6, we translated messages into German and Dutch, for the second and third largest countries. The population of contacts in non-English speaking countries for the ICS dataset was too low to provide significant meaning. We randomly split the WHOIS contacts in each country into four groups that vary language and verbosity.

We observe no significant effect from language for DDoS notifications. This is unsurprising given our notifications' overall lack of effect on DDoS amplifiers. For IPv6, as seen in Figure 9.4, we observe that translated messages resulted in *worse* patching than when left in English. Several survey respondents were surprised at receiving translated messages from United States institutions and initially suspected our notifications were phishing messages or spam, which may explain the lower patch rate. The additional overhead of translating messages paired with less successful disclosure suggests that it may be most effective to send notifications in English. However, we note that our results are limited to the small set of languages we were able to obtain reliable translations for, and deeper investigation into the effects of message language is warranted.

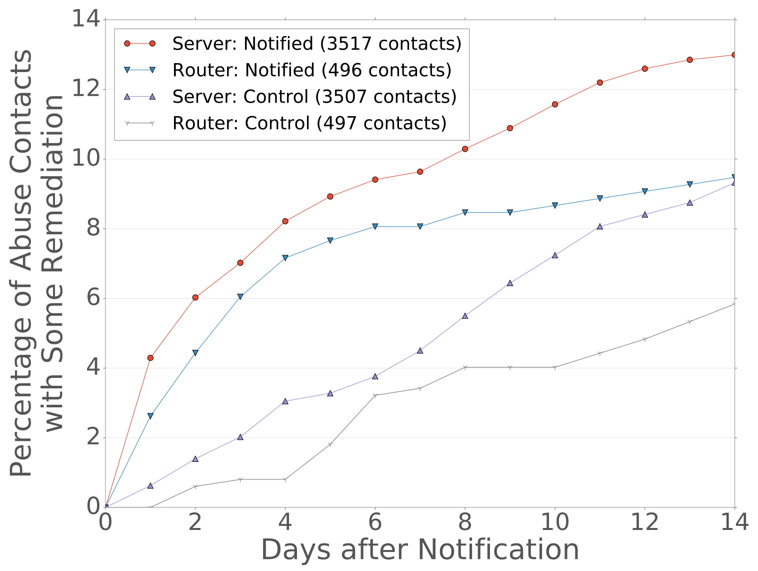
9.3.4 Staying Power of Notification's Effect

As can be seen in Figure 9.5, our notifications caused a near immediate increase in patching. However, this increased patching velocity did not persist. In other words, we find that the effects of notifications were short-lived—on the order of several days. The day after notifications were sent, we observe large increases in the remediation proportions for IPv6 and ICS notified groups, as operators responded to our reports. However, we also see that the daily changes in remediation proportions drastically dropped by the second day.

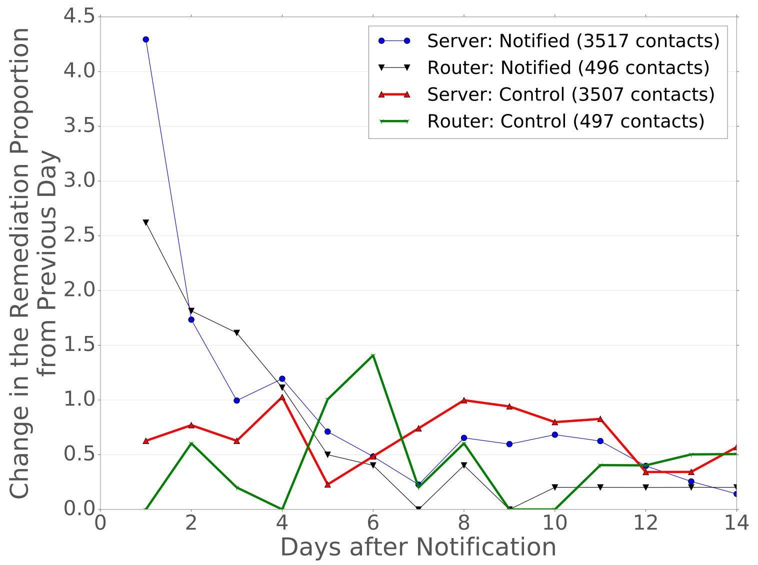
For IPv6, the daily changes in remediation proportions for all notified groups leveled off and matched that of the control group from the fifth day onward. We also witness a drop off in the daily remediation proportion changes for ICS, although a non-trivial amount of change continued throughout the first 10 days. Notably, the national CERTs first began accelerating remediation after two days, a delay compared to WHOIS experiment groups. For amplifiers, there was little change in the remediation rate over time, which is unsurprising given the limited effect of our notifications.

9.3.5 Geographic Variation

As with the national CERTs, we note variation in the patching rates between countries. This suggests that the geographic distribution of vulnerable contacts may influence a notification's outcome. As visible in Figure 9.6, the United States, Great Britain, India, and Finland were the only countries that patched significantly better than the control group. However, we note that some countries had too few hosts to be statistically significant, given the conservative nature of the Bonferroni correction.

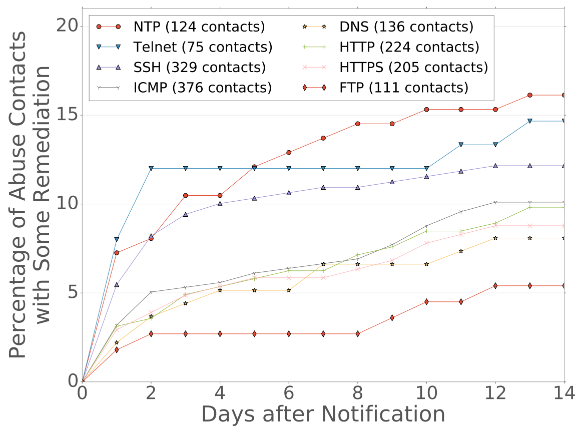


(a) Remediation rates

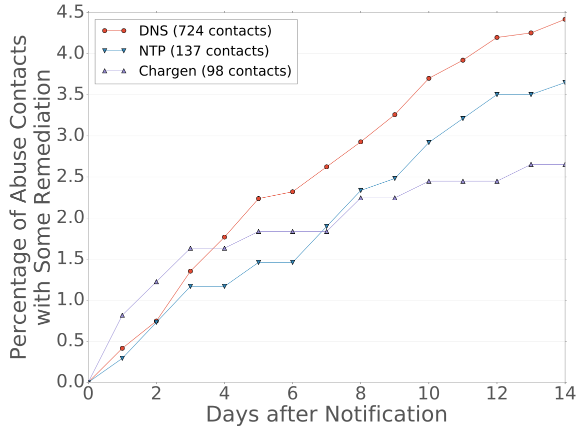


(b) Changes in the remediation proportions from one day to the next.

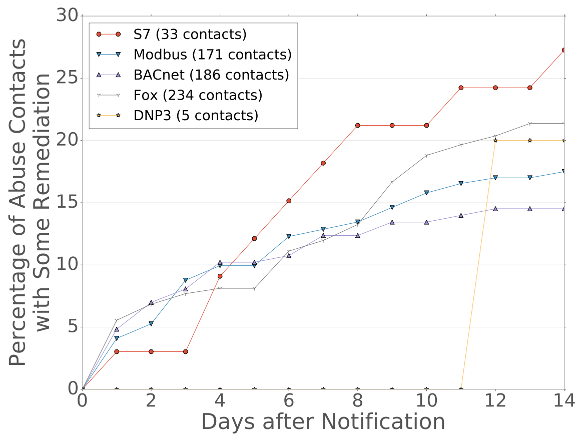
Figure 9.7: **Remediation Rates by Host Type—**We find no significant difference in the remediation rate between servers and routers.



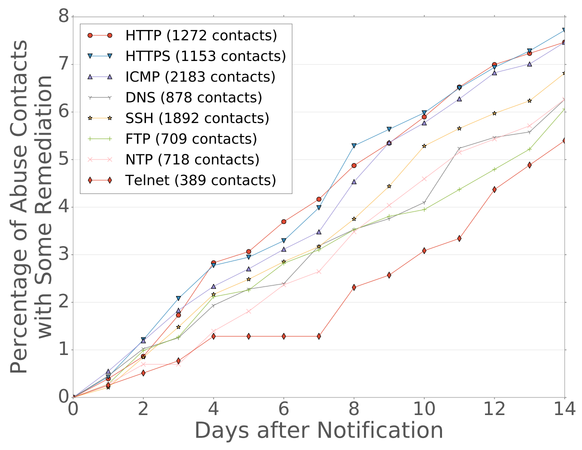
(a) Misconfigured IPv6 - WHOIS Verbose



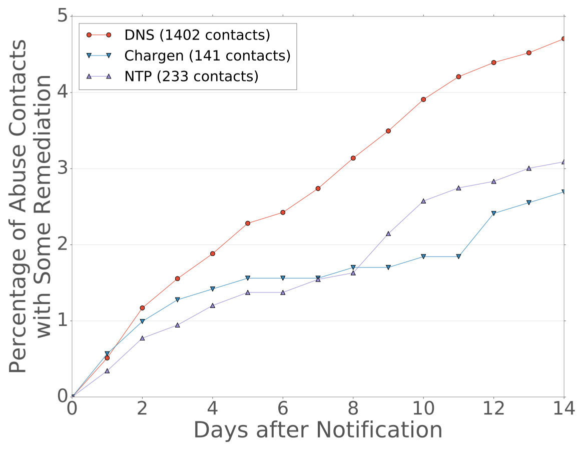
(b) DDoS Amplifiers - WHOIS Verbose



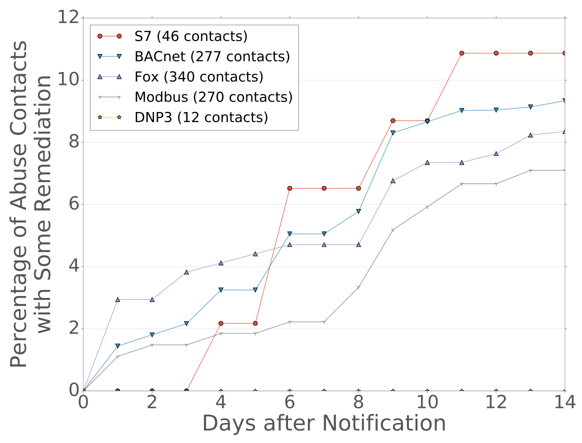
(c) ICS Services - WHOIS Verbose



(d) Misconfigured IPv6 - Control



(e) DDoS Amplifiers - Control



(f) ICS Services - Control

Figure 9.8: **Protocol Remediation Rates**—We track the remediation rate for each specific protocol within the WHOIS verbose group and the control group. We note that operators patched some protocols significantly faster than others (e.g., Telnet versus FTP).

9.3.6 Variation over Protocols

In Figure 9.8, we observe variation in the patch rates for different protocols within each vulnerability (e.g., Modbus versus S7 for ICS). As seen in Figure 9.8a, network administrators reacted most to open IPv6 NTP, Telnet, and SSH services, and least to FTP, with over a 200% difference in the remediation proportions. This variation is not reflected in the control group (Figure 9.8d), where all protocols exhibited similar behavior. This may reflect an increased likelihood that certain services were unintentionally left accessible, or that operators assessed different levels of risk for allowing different protocols to be reachable.

Operators also responded differently for the multiple ICS protocols (Figure 9.8c), but the variation is also reflected for contacts in the control group (Figure 9.8f). BACnet, Fox, and Modbus devices were fixed at similar rates. While the remediation of S7 systems initially lagged behind, there was a significant upswing in action after three days, with nearly 18% of contacts with vulnerable S7 systems patching after 8 days.

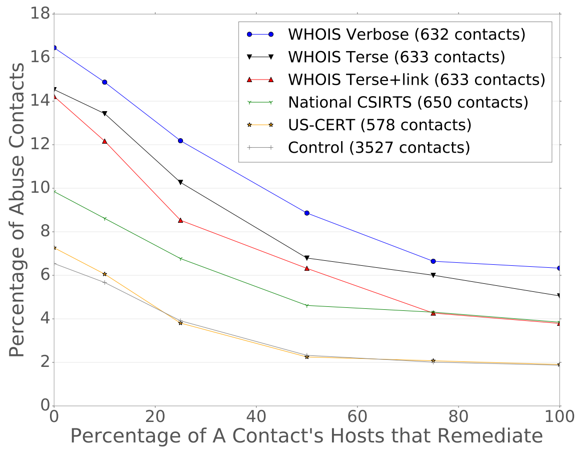
Surprisingly, no DNP3 systems had been patched within 10 days of notification (out of 5 contacts). We note that these five contact groups belonged to Internet service providers—not individual organizations. We similarly note that DNP3 differs from the other protocols and is specifically intended for power grid automation. These devices may be remote power stations which require more complex changes than local devices (e.g., installation of new hardware versus a configuration change).

While we observe variation between amplifier protocols, these fluctuations are similar in both the notified and control group. Given the limited effect of our DDoS amplifier notifications, these differences likely reflect the varying natural churn rates of these hosts.

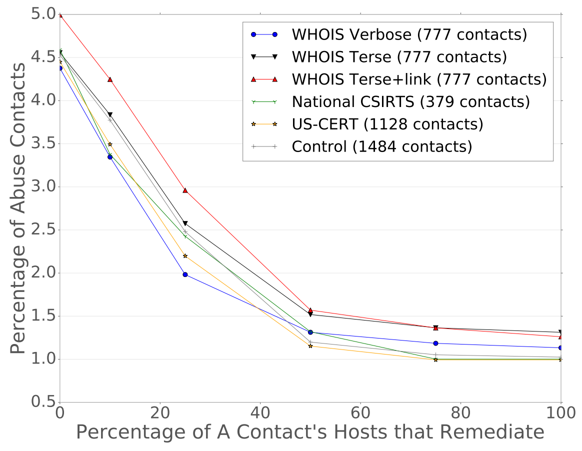
9.3.7 Host Type

When notifying IPv6 operators, we were able to distinguish between servers and routers. To assess the difference between device types, for each type, we only consider contacts with a vulnerable host of that type. We count a contact as having performed some remediation if that contact fixed at least one host of that type.

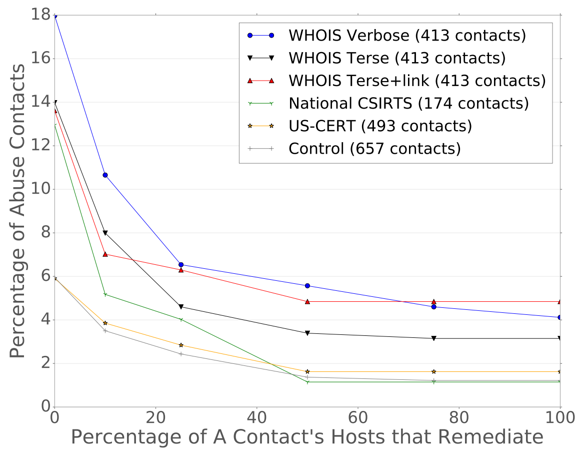
We observe that servers and routers remediated at similar rates for the first four days, after which router remediation dropped off and fell significantly below that of servers (Figure 9.7a). However, servers also naturally patched at a higher rate than routers in the control group. This difference accounts for the gap between notified servers and routers after four days. This is also visible in Figure 9.7b, where the daily changes in the remediation proportions converged after four days. After 14 days, notified contacts with servers fixed at a rate 44% higher than notified contacts with routers. The divergence in the control group was similar at 48%. This indicates that overall, network administrators respond to vulnerabilities in servers and routers about equally.



(a) Misconfigured IPv6



(b) DDoS Amplifiers



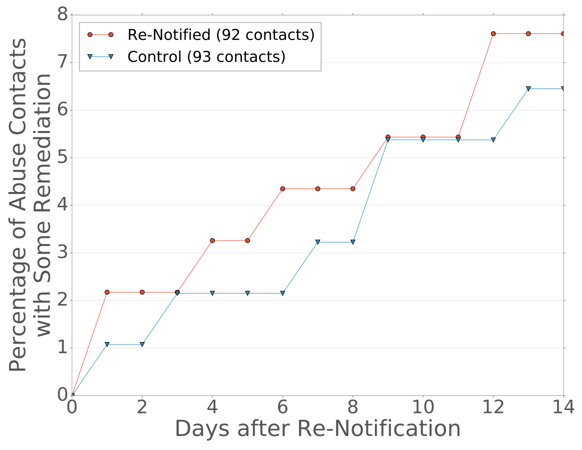
(c) ICS Services

Figure 9.9: **Remediation Completeness**—We find that most operators only fixed a subset of their vulnerable hosts. For example, only 40% of the operators that fixed a single host fixed all hosts in their purview.

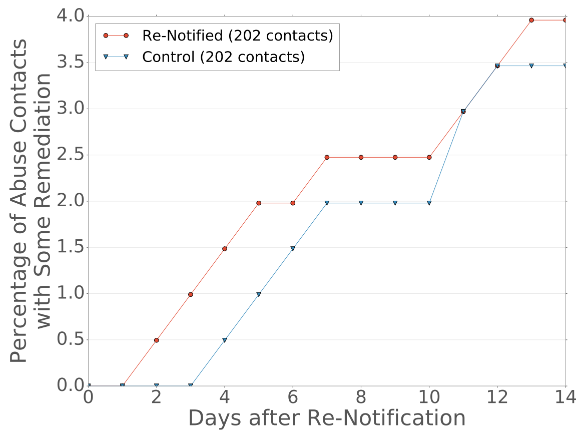
9.3.8 Degree of Remediation

Up to this point, we designated a contact as having patched if any host under its purview was patched. We now consider how well operators patched their hosts.

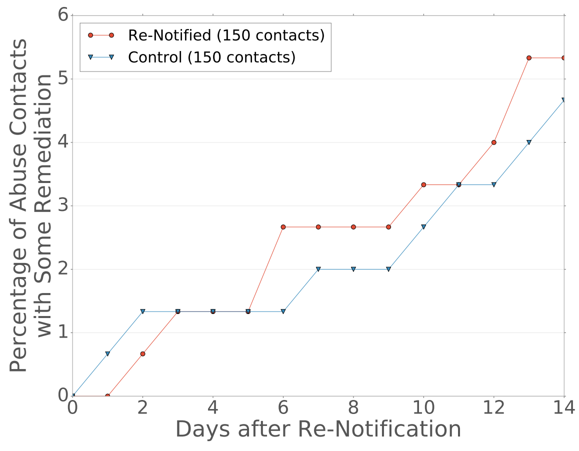
As can be seen in Figure 9.9, the majority of contacts did not patch all of their servers. Less than 60% secured all hosts and we note that 30% of groups with 100% remediation were only responsible for fixing one or two hosts. This highlights one of the challenges in the vulnerability notification process: even if our messages reach a designated contact, that contact may not have the capabilities or permissions to remediate all hosts. The multiple hops in a communication chain can be broken at any link.



(a) Misconfigured IPv6



(b) DDoS Amplifiers



(c) ICS Services

Figure 9.10: **Re-Notifications—**We find that a second round of notifications did not result in increased remediation.

9.3.9 Repeated Notifications

Given that our notifications resulted in improved patching, a natural question is whether repeat notifications promote further remediation. We conducted a second round of notifications for the contacts that were directly sent verbose messages in the first round since these proved to be the most effective. We randomly split contacts who had not remediated one month after our notifications into two groups, one as a control group and one to receive a second round of notifications.

As can be seen in Figure 9.10, the patch rates between the re-notified group and the control group were similar for all three vulnerabilities, indicating that repeat notifications are not effective. This suggests that contacts who did not remediate during the first round of notifications either were not the appropriate points of contact, or chose (either intentionally or due to lack of capabilities) to not remediate. It is unlikely they simply missed or forgot about our original notification.

9.4 Notification Reactions

We included a link to an anonymous survey in all of our notification emails as well as monitored the email address from which we sent messages. In the two weeks following our disclosures, we received 57 survey submissions and 93 human email replies. In this section, we analyze these responses.

9.4.1 Email Responses

Of the 685 email responses we received, 530 (77%) were automated responses (e.g., acknowledgment of receipt), 62 (9%) were bounces, and 93 (14%) were human responses (Table 9.3). For all three vulnerabilities, over 70% of the human responses expressed positive sentiments. We received only four negative emails, all of which concerned IPv6. Two stated that we were incorrectly using the abuse contact; the other two noted that the open IPv6 services were intentional and asked to be excluded from notifications in the future. None of the emails were threatening. We detail the breakdown for each vulnerability type in Table 9.4.

Beyond expressing sentiments, 23 contacts requested additional information—primarily about how we detected the vulnerabilities; two requested remediation instructions. Of those 23 contacts, 15 (65%) received terse notifications without a link to additional information, while 3 contacts (13%) received verbose messages. We note that verbose messages both reduced follow-up communication and resulted in the highest patching rate.

Unexpectedly, all five contacts who requested information about DDoS amplifiers asked for evidence of DDoS attacks via network logs. This may be a result of the extensive attention amplifiers have received in the past, such that operators only respond to active abuse issues regarding amplifiers.

Twelve IPv6 contacts rebutted our claim of vulnerability. Six stated that the inconsistency was intentional; one was a honeypot; and five explained that the IP addresses we sent them no longer pointed to the same dual-stack host, likely due to network churn. Two amplifier contacts claimed we falsely notified, stating that their hosts were honeypots. However, we do note that these IPs were seen as part of an attack and were therefore likely misconfigured honeypots.

Most human responses were in English, with eight (9%) in other languages: 3 Russian, 1 German, 1 Czech, 1 Swedish, 1 French, and 1 Slovak. These non-English replies were in response to English notifications and expressed gratitude; none requested additional information.

We note that the level of feedback we received regarding DDoS notifications was commensurate with our other efforts, yet the patch response was minimal. This could indicate that operators struggle with actually resolving the issue after encountering and responding to our messages, or have become desensitized enough to DDoS issues to not take real action.

9.4.2 Anonymous Survey Responses

All of our notification messages contained a link to an anonymous seven question survey (Appendix B.1), to which we received 57 submissions. We summarize the results in Table 9.5.

Response Types	**ICS**	**IPv6**	**Ampl.**
Automated	143	214	173
Human	22	48	23
Bounces	10	34	18
Total	175	296	214
Contacts w/ No Reply	85.9%	87.2%	92.8%

Table 9.3: **Email Responses**—We received 685 email responses to our notifications, of which 14% were human replies.

Human Responses	**ICS**	**IPv6**	**Ampl.**
Positive Sentiments	17	35	19
Negative Sentiments	0	4	0
Neutral Sentiments	5	9	4
Request for Information	2	16	5
Taking Actions	12	17	15
False Positive Notification	0	12	2
Total	22	48	23

Table 9.4: **Human Email Responses**—We characterize the human email responses we received in reply to our notifications.

Interestingly, 46% of respondents indicated that they were aware of the vulnerability prior to notification, and 16% indicated that they had previously attempted to resolve the problem. This contrasts with the survey results in the Heartbleed study (Chapter 7), where all 17 respondents indicated they were aware of the Heartbleed vulnerability and had previously attempted to resolve the problem. The widespread media attention regarding the Heartbleed bug may account for this discrepancy, highlighting the differences in the nature of various vulnerabilities.

For DDoS amplifiers and ICS vulnerabilities, the majority of respondents expressed that they were now taking corrective action (75% for DDoS amplifiers, 100% for ICS). For IPv6, only 56% of respondents indicated they would fix the problem. Given the nature of the IPv6 notification, it is likely that some of the misaligned policies were intentional.

Over 80% of respondents indicated that we reached out to the correct contact, who found scanning and notifications acceptable and requested future vulnerability notifications. However, this is a population with whom we successfully established communication. The accuracy of the other contacts from whom we did not hear back could be lower.

Our survey also allowed respondents to enter free form comments. We received 17 IPv6 comments, 4 DDoS amplifier comments, and 1 ICS comment. Of the IPv6 respondents, 5 thanked

Survey Responses	**ICS**	**IPv6**	**Ampl.**
Aware of Issue	2/4	20/45	4/8
Taken Prior Actions	1/4	5/43	3/8
Now Taking Action	4/4	24/43	6/8
Acceptable to Detect	3/4	35/45	7/8
Acceptable to Notify	2/4	34/45	7/8
Would want Future Notifications	2/4	30/43	7/8
Correct Contact	1/3	37/43	6/8
Total	4	45	8

Table 9.5: **Survey Responses**—We included a link to a short, anonymous survey in all of our notifications. We find that most respondents (54%) weren't aware of the vulnerabilities, but found our scanning and notifications acceptable (over 75%). Further, 62% of respondents stated they were taking corrective actions and 71% of respondents requested future notifications.

us, 7 discussed how the misalignment could be intentional or that our detection was incorrect, 3 equated our messages to spam, and 2 noted that they initially thought our translated messages were phishing messages because they expected English messages from an institution in the United States. For amplifiers, we received four comments: two thanking us and two informing us not to notify unless there is a real attack. Finally, there was only one ICS commenter, who suggested contacting vendors instead of network operators, but thanked us for our notification.

The feedback we received from these survey answers and the email responses indicates an overall positive reception of our notifications. While it may be that those who provided feedback are more opinionated, these results suggest that further discourse on notifications is needed within our community.

9.5 Discussion

Here we summarize the main results developed during our study, and the primary avenues for further work that these suggest.

Effective Vulnerability Notifications Our results indicate that vulnerability notifications can improve remediation behavior and the feedback we received from network operators was largely positive. We conclude that notifications are most effective when detailed messages are sent directly to WHOIS abuse contacts (at least when one lacks a channel such as Google Search Console for webmasters, per Chapter 8). These notifications were most effective in our experiments and resulted in an additional 11% of contacts addressing a vulnerability in response to our message.

On the one hand, this result provides clear guidance on how to best notify network operators. On the other hand, the majority of organizations did not patch their hosts despite our notifications. Even among those who patched at least one host, most did not fix all of their vulnerable hosts. In the case of networks hosting DDoS amplifiers, *no* form of notification generated benefits statistically significant over the control.

The failures to remediate could signal a number of problems, including:

1. failure to contact the proper parties who could best instigate remediation;
2. a need for better education about the significance of the vulnerability;
3. a need for better education about the remediation process;
4. administrative or logistical hurdles that proved too difficult for those parties to overcome;
5. or a cost-benefit analysis by those parties that concluded remediation was not worth the effort.

In addition, we found the effects of our notification campaigns to be short-lived: if recipients did not act within the first couple days, they were unlikely to ever do so. Repeat notifications did not further improve remediation levels.

Thus, while we have developed initial guidance for conducting effective notifications, there remain many unanswered questions as to how to best encourage operators to patch vulnerable hosts.

Improving Centralized Notification Mechanisms We observed that relying on national and regional CERT organizations for vulnerability notifications had either a modest effect (compared to our direct notifications) or no effect (indistinguishable from our unnotified controls). While certain national CERTs evinced improved levels of remediation, others either did not act upon the information we reported, or if they did so, recipients ignored their messages. Thus, the community should consider more effective mechanisms for facilitating centralized reporting, either within the existing CERT system, or using some separate organizational structure. This need is quite salient because the burden of locating and messaging thousands of individual contacts is high enough that many researchers will find it too burdensome to conduct notifications themselves.

Open Ethical Questions The process of notifying parties regarding security issues raises a number of ethical questions. The community has already discussed some of these in depth, as in the debates concerning "full disclosure." Contacting individual sites suffering from vulnerabilities, likewise, raises questions regarding appropriate notification procedures.

For example, WHOIS abuse emails are a point-of-contact that multiple notification efforts have relied on, including the studies in Chapters 7 and 8 and other concurrent works [50, 110, 180, 195]. However, these contacts are technically designated for reports of abusive, malicious behavior (a point noted in the feedback we received as detailed in Section 9.4). While vulnerability reports have a somewhat similar flavor, they do not serve the same purpose. It behooves the security community to establish a standardized and reliable point-of-contact for communicating security issues.

Another question concerns whether the benefits of repeated notifications for the same vulnerability outweigh the costs imposed on recipients. Some may derive no benefits from the additional messages due to having no means to effectively remediate, yet must spend time ingesting the notifications. From our results, we observed that repeat notifications did not promote further patching, which argues against performing re-notifications.

More provocative, and related to the full-disclosure debate mentioned above, is the notion of *threatening* recipients with publicly revealing their vulnerabilities if unaddressed after a given amount of time. Likely, the research community would find this (quite) unpalatable in general; however, one can imagine specific situations where the community might conclude that spurring vital action justifies such a harsh step, just as some have concluded regarding full disclosure.

Future Abuse of Notifications In a future with widespread notifications, we would hope that security issues could be rectified more extensively and quickly. However, this would provide a new avenue for abuse, as attackers could potentially leverage the open communication channel to target network operators. As a simple example, a malicious actor could notify operators about a real security issue, and inform the operators to install a malicious application to help hosts resolve the security gap. While existing techniques such as phishing detection and binary analysis can help limit these attacks, the problem domain likely will yield new challenges. It is important that the security community remain cognizant of these dangers as the state of security notifications evolves.

Effective Remediation Tools For contacts that do not remediate, our measurements cannot distinguish which of the underlying reasons sketched above came into play. However, while some operators may lack sufficient motivation to take action, it seems quite plausible that others wish to, but lack the technical capabilities, resources, or permissions to do. We identified some of these barriers in Chapter 3's analysis of security patch development and Chapter 4's study of system administrator software updating. Accordingly, we see a need for investigation into the operational problems that operators encounter when considering or attempting remediation, as well as the development of effective and usable remediation tools that simplify the operators' tasks. This need was also raised in Chapter 8. By reducing the effort and resources required to address a vulnerability, such tools could also increase the likelihood that an operator would take the steps to react to vulnerability reports. Ultimately, automated systems would be ideal, but these face significant challenges, such as heterogeneous platforms, potential abusive or malicious behavior, and inadvertent disruption of mission-critical systems.

9.6 Conclusion

In this chapter, we have undertaken an extensive study of notifying thousands of network operators of security issues present within their networks, with the goal of illuminating which fundamental aspects of notifications have the greatest impact on efficacy. Our study investigated vulnerabilities that span a range of protocols and considerations: exposure of industrial control systems;

apparent firewall omissions for IPv6-based services; and exploitation of local systems in DDoS amplification attacks.

Through controlled multivariate experiments, we studied the impact of a number of variables: choice of party to contact (WHOIS abuse contacts versus national CERTs versus US-CERT), message verbosity, hosting a website linked to in the message, and translating the message into the notified party's local language. We monitored the vulnerable systems for several weeks to determine their rate of remediation in response to changes to these variables.

We also assessed the outcome of the emailing process itself and characterized the sentiments and perspectives expressed in both the human replies and an optional anonymous survey that accompanied our notifications. The responses were largely positive, with 96% of human email responses expressing favorable or neutral sentiments.

Our findings indicate that notifications can have a significant positive effect on patching, with the best messaging regimen being directly notifying contacts with detailed information. An additional 11% of contacts addressed the security issue when notified in this fashion, compared to the control. However, we failed to prompt the majority of contacts to respond, and even when they did, remediation was often only partial. Repeat notifications did not further improve remediation. Given these positive yet unsatisfactory outcomes, we call on the security community to more deeply investigate notifications and establish standards and best practices that promote their effectiveness.

Chapter 10

Internet Outreach Discussion and Conclusion

Security remedies are ultimately only effective if they are deployed in practice. In the first part of this dissertation, we looked at how remedies are created and then deployed to impacted systems, focusing on security patches that fix vulnerabilities. In this second part, we explored the gap connecting those processes, asking how we encourage the application of identified remedies. To do so, we investigated using Internet-scale outreach to spur administrators of Internet-facing systems to correct various security concerns. We note that our efforts in notifying administrators is distinct from end user warnings and notices, as we again focused on a subpopulation (done in Chapter 4 as well) that is more technically sophisticated and operates with increased responsibilities for managing the security of systems on the Internet.

Through the series of studies that populate Part II of this dissertation, we have conducted one of the first systematic explorations of Internet-wide administrator notifications, evaluating whether such notifications have positive effect, how best to conduct such outreach campaigns, and what are existing shortcomings or limitations of these efforts. This body of work has developed broad understanding of notifications that span a variety of different widespread security concerns, from patchable vulnerabilities to server misconfigurations to compromised websites. Given the Internet-wide nature of the notification campaigns, each individual study is not exactly replicable. By design, each campaign shifts the landscape of Internet systems affected by the particular security concern. However, by studying a variety of different security contexts and identifying similarities between corresponding findings, we have been able to establish generalizable results.

Consistently across these works, we have found that security notifications are able to successfully spur a significant portion of administrators to better secure their systems. Our best results came from our study on Google's notifications to webmasters of compromised sites, where direct notifications led to over 75% of webmasters remediating. This rate was more than 50% higher than that of webmasters still receiving public signals through browser interstitials and Google search result warnings, but without direct outreach. In addition, notified webmasters re-secured their sites in almost half the time. Without public signals, the site remediation rate would likely be even lower. We note though that this is close to a best-case scenario for notifications: we have a reliable notification channel from a well-recognized organization (Google) about a severe security issue (compromise, where something malicious has *already* happened) with strong incentives (due to browser interstitials and search engine warnings discouraging site visitors). However, recovering from compromise is arguably more technically complex than most other security concerns we considered, and the large positive effect of notifications demonstrated that it can have an important role to play in Internet security even if only for such ideal situations.

However, we argue that notifications can be used more extensively. Even in the other notification experiments, security notifications could drive a non-trivial minority (e.g., at minimum 10-20%) to correct the security problems. While still not a majority, these efforts did push hundreds of thousands of servers to improve their security stance. What is promising is that these remediation rates can likely be raised by tackling some of the notification issues identified in Chapter 9, providing ample opportunities for future research and community initiatives to improve outreach efforts. Specifically, we believe the most relevant improvements are:

1. Establishing reliable communication channels with the appropriate administrator contacts.
2. Identifying methods of establishing notification trustworthiness that is resistant to use in phishing attacks (this may be simultaneously accomplished through establishing reliable communication channels).
3. Exploring incentive mechanisms to encourage administrators to correct security concerns, such as through public ratings or gamification of security stances.
4. Developing more effective and usable systems for applying remedies, whether it be for patch deployment (explored in Chapters 3 and 4) or compromise recovery, which will ultimately reduce the costs and burden of taking remediation actions.

In the meantime, the results from our studies have helped identify best practices for conducting outreach efforts today. In particular, the most salient recommendations are:

1. Outreach messages should be sent directly to the most appropriate technical contact. Currently, if one lacks a previously established communication channel, emails to WHOIS abuse or technical contacts are likely the most effective.
2. The notification messages themselves should be detailed in describing who the notifiers are, what the security concern is, why action should taken, and what the recommended next steps are. Ideally the message would also including a trustworthy link (e.g., to a university-hosted web page) to more extensive information. However, one should consider the target audience and their technical expertise. For example, webmasters may be less technically sophisticated on average compared to network administrators.
3. Messages should only be translated if the context makes it reasonably expected (e.g., if from a global organization such as Google)
4. Messages should only sent once (without followups).

Since these initial studies, the body of work on Internet-wide outreach has expanded further with studies both investigating outreach methods and using it [46–50, 57, 117, 132, 164, 179, 180, 211]. We expect this line of investigation to continue, and believe that outreach efforts will play an important role in securing Internet systems. We also note that outreach is ultimately a social or human process, yet our methods for studying it have not only explored human factors (such as through recipient surveys), but have leveraged Internet scanning and network measurements to monitor Internet-wide behavior. As with Part I, we relied on a diverse set of research techniques to establish empirical grounding. This again highlights the complexity of this problem space, where we will need to consider and tackle various factors at play beyond just traditional computer technical considerations, including human, social, and economic facets.

Chapter 11

Conclusion and Parting Words

As a research community, we are constantly advancing our capabilities for securing computer systems and networks. Yet today, we remain reliant on the remediation of security issues to address emergent concerns. This state will *not* change anytime in the near future, and it is plausible that it will never change. Thus, understanding and improving how we remedy security problems is of utmost importance.

In this dissertation, we have performed an extensive treatment of the remediation of security concerns at an Internet scale, investigating in depth each stage of the remediation process. Specifically, we have studied the development of remedies (Chapter 3), how to inform and encourage those affected by a security issue to apply existing remedies (Chapters 7–9), and the remedy deployment process itself (Chapter 4). By no means does this dissertation comprehensively explore the remediation space or solve the existing issues; such an achievement is far beyond the contributions of a single dissertation. However, through using a variety of data-driven methods, this dissertation has provided new empirical grounding and insights on this critical aspect of real-world security, shedding light on potential avenues for progress.

Moving forward, there are ample opportunities for improvements, from developing systems and tools to automate remediation actions, to establishing reliable, trustworthy methods of communicating information at scale on urgent security issues, to exploring economic and policy mechanisms to incentivize prompt responses. These three examples highlight the diversity and complexity of this problem space, as problems are not simply traditional technical ones. As a consequence, solutions will not be either. The studies forming this dissertation already exhibit this characteristic, as they involve a variety of distinct research methods, from large-scale data mining to user studies to Internet-wide network measurements and experiments. However, there is even more that can and should be done, including exploration of economic, organizational, policy, and legal considerations. Admittedly, the many dimensions of this problem space makes it challenging. However, it also makes it exciting, intellectually stimulating, and fruitful for further efforts.

The nature of the remediation problem space also necessitates an empirical perspective. To tackle such a complex problem, one must truly understand how its many dimensions manifest in practice. Without a comprehensive model of remediation empirically-grounded in the real world, solution attempts will likely fall short as they will fail to account for important aspects. For example, some have considered security patching a solved problem, as we know how to create patches and people just need to get their acts together and apply them promptly. Yet the results from this dissertation have shown that no part of that statement is true. We are not able to reliably create security fixes, and users have rational calculated reasons for delaying or avoiding patches. Ultimately, we cannot hope to improve security in practice through relying on personal experiences, anecdotes, and assumptions. We must take a more scientific approach and rigorously analyze reality, and use data-supported insights to drive developments. This dissertation serves as an example of such an effort.

Bibliography

[1] Alexa Top 1,000,000 Sites. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip.

[2] American Fuzzy Lop. http://lcamtuf.coredump.cx/afl/.

[3] Android Platform Versions. https://developer.android.com/about/dashboards/index.html#Platform.

[4] Ansible. https://www.ansible.com/.

[5] Apache HTTPD Changelog. https://www.apache.org/dist/httpd/CHANGES_2.4.

[6] Bitcoin Core Version History. https://bitcoin.org/en/version-history.

[7] Cassandra Wiki - Internode Encryption. http://wiki.apache.org/cassandra/InternodeEncryption.

[8] cgit. https://git.zx2c4.com/cgit/about/.

[9] Chef. https://www.chef.io/chef/.

[10] Core Infrastructure Initiative. https://www.coreinfrastructure.org.

[11] Exuberant Ctags. http://ctags.sourceforge.net/.

[12] Firefox Release Notes. https://www.mozilla.org/en-US/firefox/releases/.

[13] GitLab. https://about.gitlab.com/.

[14] GitWeb. https://git-scm.com/book/en/v2/Git-on-the-Server-GitWeb.

[15] HP Support Document c04249852. http://goo.gl/AcUG8I.

[16] Install Ejabberd. http://www.ejabberd.im/tuto-install-ejabberd.

[17] Installing OpenDKIM. http://www.opendkim.org/INSTALL.

[18] ISC Software Defect and Security Vulnerability Disclosure Policy. https://kb.isc.org/article/AA-00861/164/ISC-Software-Defect-and-Security- Vulnerability-Disclosure-Policy.html.

[19] Ksplice. https://www.ksplice.com/.

[20] Open Crypto Audit Project. https://opencryptoaudit.org.

[21] Puppet. https://puppet.com/.

[22] Reddit. https://www.reddit.com/.

[23] Spicework. https://www.spiceworks.com/.

[24] SSL Pulse. https://www.trustworthyinternet.org/ssl-pulse/.

[25] Telnet Server with SSL Encryption Support. https://packages.debian.org/stable/net/telnetd-ssl.

[26] Terraform. https://www.terraform.io/.

[27] Undefined Behavior Sanitizer. https://clang.llvm.org/docs/UndefinedBehavior Sanitizer.html.

[28] Is Openfire Affected by Heartbleed? https://community.igniterealtime.org/thread/52272, 2014.

[29] Tomcat Heartbleed. https://wiki.apache.org/tomcat/Security/Heartbleed, 2014.

[30] Adobe. Heartbleed Update. http://blogs.adobe.com/psirt/?p=1085, 2014.

[31] D. Akhawe and A. P. Felt. Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness. In *USENIX Security Symposium*, 2013.

[32] M. Al-Bassam. Top Alexa 10,000 Heartbleed Scan—April 14, 2014. https://github.com/musalbas/heartbleed-masstest/blob/94cd9b6426311f0d20539e696496ed3d7bdd2a94/top1000.txt.

[33] Alienth. We Recommend that You Change Your Reddit Password. http://www.reddit.com/r/announcements/comments/231hl7/we_recommend_that_you_change_your_reddit_password/, 2014.

[34] B. Amann, M. Vallentin, S. Hall, and R. Sommer. Extracting Certificates from Live Traffic: A Near Real-Time SSL Notary Service. Technical Report TR-12-014, ICSI, 2012.

[35] Apache Software Foundation. CouchDB and the Heartbleed SSL/TLS Vulnerability. `https://blogs.apache.org/couchdb/entry/couchdb_and_the_heartbleed_ssl`, 2014.

[36] D. Armstrong, A. Gosling, J. Weinman, and T. Marteau. The Place of Inter-Rater Reliability in Qualitative Research: An Empirical Study. *Sociology*, 31(3):597–606, 1997.

[37] AWeber Communications. Heartbleed: We're Not Affected. Here's What You Can Do To Protect Yourself. `http://blog.aweber.com/articles-tips/heartbleed-how-to-protect-yourself.htm`, 2014.

[38] R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, and M. Prabaker. Field Studies of Computer System Administrators: Analysis of System Management Tools and Practices. In *ACM Conference on Computer Supported Cooperative Work (CSCW)*, 2004.

[39] L. Bilge, Y. Han, and M. Dell'Amico. RiskTeller: Predicting the Risk of Cyber Incidents. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[40] Bitcoin. OpenSSL Heartbleed Vulnerability. `https://bitcoin.org/en/alert/2014-04-11-heartbleed`, 2014.

[41] K. Borgolte, C. Kruegel, and G. Vigna. Meerkat: Detecting Website Defacements through Image-based Object Recognition. In *USENIX Security Symposium*, 2015.

[42] BTCJam. Official BTCJam Update. `http://blog.btcjam.com/post/82158642922/official-btcjam-update`, 2014.

[43] CAIDA. Archipelago (Ark) Measurement Infrastructure. `http://www.caida.org/projects/ark/`.

[44] D. Canali, D. Balzarotti, and A. Francillon. The Role of Web Hosting Providers in Detecting Compromised Websites. In *World Wide Web Conference (WWW)*, 2013.

[45] Centers for Disease Control and Prevention. Patient Notification Toolkit. `http://www.cdc.gov/injectionsafety/pntoolkit/index.html`.

[46] O. Çetin, C. Gañán, , L. Altena, S. Tajalizadehkhoob, and M. van Eeten. Tell Me You Fixed It: Evaluating Vulnerability Notifications via Quarantine Networks. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[47] O. Çetin, C. Gañán, L. Altena, T. Kasama, D. Inoue, K. Tamiya, Y. Tie, K. Yoshioka, and M. van Eeten. Cleaning Up the Internet of Evil Things: Real-World Evidence on ISP and Consumer Efforts to Remove Mirai. In *Network and Distributed System Security Symposium (NDSS)*, 2019.

[48] O. Çetin, C. Gañán, L. Altena, S. Tajalizadehkhoob, and M. van Eeten. Let Me Out! Evaluating the Effectiveness of Quarantining Compromised Users in Walled Gardens. In *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2018.

[49] O. Cetin, C. Ganán, M. Korczynski, and M. van Eeten. Make Notifications Great Again: Learning How to Notify in the Age of Large-Scale Vulnerability Scanning. In *Workshop on the Economy of Information Security (WEIS)*, 2017.

[50] O. Cetin, M. H. Jhaveri, C. Ganan, M. van Eeten, and T. Moore. Understanding the Role of Sender Reputation in Abuse Reporting and Cleanup. In *Workshop on the Economy of Information Security (WEIS)*, 2015.

[51] S. Chiasson, P. van Oorschot, and R. Biddle. Even Experts Deserve Usable Security: Design Guidelines for Security Management Systems. In *USENIX SOUPS Workshop on Usable IT Security Management (USM)*, 2007.

[52] S. Christey and B. Martin. Buying Into the Bias: Why Vulnerability Statistics Suck. In *BlackHat*, 2013.

[53] Cisco. Annual Security Report. `https://www.cisco.com/web/offers/pdfs/cisco-asr-2015.pdf`, 2015.

[54] Conficker Working Group. Lessons Learned. `http://www.confickerworkinggroup.org/wiki/uploads/Conficker_Working_Group_Lessons_Learned_17_June_2010_final.pdf`, 2010.

[55] O. Crameri, N. Knezevic, D. Kostic, R. Bianchini, and W. Zwaenepoel. Staged Deployment in Mirage, an Integrated Software Upgrade Testing and Distribution System. *SIGOPS Oper. Syst. Rev.*, 41(6):221–236, Oct. 2007.

[56] N. Craver. Is Stack Exchange Safe from Heartbleed? `http://meta.stackexchange.com/questions/228758/is-stack-exchange-safe-from-heartbleed`, 2014.

[57] J. Czyz, M. Luckie, M. Allman, and M. Bailey. Don't Forget to Lock the Back Door! A Characterization of IPv6 Network Security Policy. In *Network and Distributed System Security Symposium (NDSS)*, 2016.

[58] C. Dietrich, K. Krombholz, K. Borgolte, and T. Fiebig. Investigating System Operators' Perspective on Security Misconfigurations. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[59] R. Dingledine. Tor OpenSSL Bug CVE-2014-0160. `https://blog.torproject.org/blog/openssl-bug-cve-2014-0160`, 2014.

[60] D. Dittrich, M. Bailey, and S. Dietrich. Towards Community Standards for Ethical Behavior in Computer Security Research. Technical report, Stevens Institute of Technology, 2009.

[61] Dropbox Support. Quick Update on Heartbleed: We've Patched All of Our User-Facing Services & Will Continue to Work to Make Sure Your Stuff is Always Safe. `https://twitter.com/dropbox_support/status/453673783480832000`, 2014.

[62] T. Duebendorfer and S. Frei. Why Silent Updates Boost Security. Technical report, ETH Zurich, 2009.

[63] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A Search Engine Backed by Internet-Wide Scanning. In *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[64] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS Certificate Ecosystem. In *ACM Internet Measurement Conference (IMC)*, 2013.

[65] Z. Durumeric*, F. Li*, J. Kasten, N. Weaver, J. Amann, J. Beekman, M. Payer, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. The Matter of Heartbleed. In *ACM Internet Measurement Conference (IMC)*, 2014.

[66] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-Wide Scanning and its Security Applications. In *USENIX Security Symposium*, 2013.

[67] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna. COMPA: Detecting Compromised Accounts on Social Networks. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[68] A. Ellis. Akamai Heartbleed Update (V3). `https://blogs.akamai.com/2014/04/heartbleed-update-v3.html`, 2014.

[69] S. Farhang, J. Weidman, M. M. Kamani, J. Grossklags, and P. Liu. Take It or Leave It: A Survey Study on Operating System Upgrade Practices. In *Annual Computer Security Applications Conference (ACSAC)*, 2018.

[70] A. P. Felt, A. Ainslie, R. W. Reeder, S. Consolvo, S. Thyagaraja, A. Bettes, H. Harris, and J. Grimes. Improving SSL Warnings: Comprehension and Adherence. In *ACM Conference on Human Factors in Computing Systems (CHI)*, 2015.

[71] A. Forget, S. Pearman, J. Thomas, A. Acquisti, N. Christin, L. F. Cranor, S. Egelman, M. Harbach, and R. Telang. Do or Do Not, There Is No Try: User Engagement May Not Improve Security Outcomes. In *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2016.

[72] Forum of Incident Response and Security Teams. Common Vulnerability Scoring System v3.0: Specification Document. `https://www.first.org/cvss/specification-document`.

[73] S. Frei. End-Point Security Failures: Insights gained from Secunia PSI Scans. In *International Symposium on Engineering Secure Software and Systems (ESSoS)*, 2011.

[74] S. Frei, M. May, U. Fiedler, and B. Plattner. Large-Scale Vulnerability Analysis. In *SIGCOMM Workshop on Large-Scale Attack Defense (LSAD)*, 2006.

[75] O. Garrett. NGINX and the Heartbleed Vulnerability. `http://nginx.com/blog/nginx-and-the-heartbleed-vulnerability/`, 2014.

[76] C. Gkantsidis, T. Karagiannis, and M. Vojnovic. Planet Scale Software Updates. *ACM SIGCOMM CCR*, 36(4):423–434, Aug. 2006.

[77] GoDaddy. OpenSSL Heartbleed: We've Patched Our Servers. `http://support.godaddy.com/godaddy/openssl-and-heartbleed-vulnerabilities/`, 2014.

[78] Google. Sanitizers. `https://github.com/google/sanitizers`.

[79] Google. Fighting Spam. `http://www.google.com/insidesearch/howsearchworks/fighting-spam.html`, 2015.

[80] Google. Googlebot. `https://support.google.com/webmasters/answer/182072?hl=en`, 2015.

[81] Google. Safe Browsing Transparency Report. `https://www.google.com/transparencyreport/safebrowsing/`, 2015.

[82] Google. Search Console. `https://www.google.com/webmasters/tools/home?hl=en`, 2015.

[83] Google. "This site may be hacked" message. `https://support.google.com/websearch/answer/190597?hl=en`, 2015.

[84] Google. "This site may harm your computer" notification. `https://support.google.com/websearch/answer/45449?hl=en`, 2015.

[85] Google Open Source Blog. Announcing OSS-Fuzz: Continuous Fuzzing for Open Source Software. `https://opensource.googleblog.com/2016/12/announcing-oss-fuzz-continuous-fuzzing.html`, 2016.

[86] R. Graham. Masscan: The Entire Internet in 3 Minutes. `http://blog.erratasec.com/2013/09/masscan-entire-internet-in-3-minutes.html`, 2013.

[87] L. Grangeia. Heartbleed, Cupid and Wireless. `http://www.sysvalue.com/en/heartbleed-cupid-wireless/`, 2014.

[88] S. Grant. The Bleeding Hearts Club: Heartbleed Recovery for System Administrators. https://www.eff.org/deeplinks/2014/04/bleeding-hearts-club-heartbleed-recovery-system-administrators, 2014.

[89] S. Grob. JSC Exploits. https://googleprojectzero.blogspot.com/2019/08/jsc-exploits.html, 2019.

[90] G. Grossmeier. Wikimedia's Response to the "Heartbleed" Security Vulnerability. https://blog.wikimedia.org/2014/04/10/wikimedias-response-to-the-heartbleed-security-vulnerability/, 2014.

[91] B. Grubb. Heartbleed Disclosure Timeline: Who Knew What and When. http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knew-what-and-when-20140415-zqurk.html, 2014.

[92] Z. Gu, E. Barr, D. Hamilton, and Z. Su. Has the Bug Really Been Fixed? In *International Conference on Software Engineering (ICSE)*, 2010.

[93] L. Haisley. OpenSSL Crash with STARTTLS in Courier. http://sourceforge.net/p/courier/mailman/message/32298514/, 2014.

[94] M. Hicks and S. Nettles. Dynamic Software Updating. *ACM Transactions on Programming Languages and Systems*, 27(6):1049–1096, Nov. 2005.

[95] A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 1970.

[96] S. Hofmeyr, T. Moore, S. Forrest, B. Edwards, and G. Stelle. Modeling Internet-Scale Policies for Cleaning up Malware. In *Economics of Information Security and Privacy III*, 2013.

[97] Z. Huang, M. D'Angelo, D. Miyani, and D. Lie. Talos: Neutralizing Vulnerabilities with Security Workarounds for Rapid Response. In *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[98] IBM. OpenSSL Heartbleed (CVE-2014-0160). https://www-304.ibm.com/connections/blogs/PSIRT/entry/openssl_heartbleed_cve_2014_0160, 2014.

[99] Infusionsoft. What You Need to Know About Heartbleed. http://blog.infusionsoft.com/company-news/need-know-heartbleed/, 2014.

[100] Internal Revenue Service. IRS Statement on "Heartbleed" and Filing Season. http://www.irs.gov/uac/Newsroom/IRS-Statement-on-Heartbleed-and-Filing-Season, 2014.

[101] I. Ion, R. Reeder, and S. Consolvo. "...No One Can Hack My Mind": Comparing Expert and Non-Expert Security Practices. In *USENIX Symposium On Usable Privacy and Security (SOUPS)*, 2015.

[102] M. Isaac. Apple Says iOS, OSX and "Key Web Services" Not Affected by Heartbleed Security Flaw. `http://recode.net/2014/04/10/apple-says-ios-osx-and-key-web-services-not-affected-by-heartbleed-security-flaw/`, 2014.

[103] Jonathan Corbet. Kernel Vulnerabilities: Old or New? `https://lwn.net/Articles/410606/`, 2010.

[104] M. Jones. Link Shim - Protecting the People who Use Facebook from Malicious URLs. `https://www.facebook.com/notes/facebook-security/link-shim-protecting-the-people-who-use-facebook-from-malicious-urls`, 2012.

[105] W. Kamishlian and R. Norris. Installing OpenSSL for Jabberd 2. `http://www.jabberdoc.org/app_openssl.html`.

[106] E. Kandogan, P. Maglio, E. Haber, and J. Bailey. *Taming Information Technology: Lessons from Studies of System Administrators*. Oxford University Press, 2012.

[107] Kees Cook. Security Bug Lifetime, 2016. `https://outflux.net/blog/archives/2016/10/18/security-bug-lifetime`.

[108] S. Kraemer and P. Carayon. Human Errors and Violations in Computer and Information Security: The Viewpoint of Network Administrators and Security Specialists. *Applied Ergonomics*, 38(2):143 – 154, 2007.

[109] K. Krombholz, W. Mayer, M. Schmiedecker, and E. Weippl. "I Have No Idea What I'm Doing" - On the Usability of Deploying HTTPS. In *USENIX Security Symposium*, 2017.

[110] M. Kührer, T. Hupperich, C. Rossow, and T. Holz. Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In *USENIX Security Symposium*, 2014.

[111] L. L. Kupper and K. B. Hafner. On Assessing Interrater Agreement for Multiple Attribute Responses. *Biometrics*, 45(3):957, Sept. 1989.

[112] F. Li, Z. Durumeric, J. Czyz, M. Karami, M. Bailey, D. McCoy, S. Savage, and V. Paxson. You've Got Vulnerability: Exploring Effective Vulnerability Notifications. In *USENIX Security Symposium*, 2016.

[113] F. Li, G. Ho, E. Kuan, Y. Niu, L. Ballard, K. Thomas, E. Bursztein, and V. Paxson. Remedying Web Hijacking: Notification Effectiveness and Webmaster Comprehension. In *World Wide Web Conference (WWW)*, 2016.

[114] F. Li and V. Paxson. A Large-Scale Empirical Analysis of Security Patches. In *ACM Conference on Computer and Communication Security (CCS)*, 2017.

[115] F. Li, L. Rogers, A. Mathur, N. Malkin, and M. Chetty. Keepers of the Machines: Examining How System Administrators Manage Software Updates. In *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2019.

[116] Litespeed Technologies. LSWS 4.2.9 Patches Heartbleed Bug. `http://www.litespeedtech.com/support/forum/threads/lsws-4-2-9-patches-heartbleed-bug.8504/`, 2014.

[117] D. Liu, S. Hao, and H. Wang. All Your DNS Records Point to Us. In *ACM Conference on Computer and Communication Security (CCS)*, 2016.

[118] M. Luckie. Scamper: A Scalable and Extensible Packet Prober for Active Measurement of the Internet. In *ACM Internet Measurement Conference (IMC)*, 2010.

[119] L. Ma. Webmaster Tools now in 26 languages. `http://googlewebmastercentral.blogspot.com/2008/05/webmaster-tools-now-in-26-languages.html`, 2008.

[120] H. Mann and D. Whitney. On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other. *Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[121] S. Marquess. Of Money, Responsibility, and Pride. `http://veridicalsystems.com/blog/of-money-responsibility-and-pride/`, 2014.

[122] Mashable. The Heartbleed Hit List: The Passwords You Need to Change Right Now. `http://mashable.com/2014/04/09/heartbleed-bug-websites-affected/`, 2014.

[123] A. Mathur and M. Chetty. Impact of User Characteristics on Attitudes Towards Automatic Mobile Application Updates. In *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2017.

[124] A. Mathur, J. Engel, S. Sobti, V. Chang, and M. Chetty. “They Keep Coming Back Like Zombies”: Improving Software Updating Interfaces. In *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2016.

[125] A. Mathur, N. Malkin, M. Harbach, E. Peer, and S. Egelman. Quantifying Users’ Beliefs about Software Updates. In *NDSS Workshop on Usable Security (USEC)*, 2018.

[126] MaxMind, LLC. GeoIP2 Database.

[127] T. J. McCabe. A Complexity Measure. In *IEEE Transaction on Software Engineering*, 1976.

[128] N. Mehta and Codenomicon. The Heartbleed Bug. `http://heartbleed.com`, 2014.

[129] Microsoft. System Center Configuration Manager. `https://www.microsoft.com/en-us/cloud-platform/system-center-configuration-manager`.

[130] Microsoft. Windows Server Update Services. `https://docs.microsoft.com/en-us/windows-server/administration/windows-server-update-services/get-started/windows-server-update-services-wsus`.

[131] Microsoft. Microsoft Services Unaffected by OpenSSL Heartbleed Vulnerability. `http://blogs.technet.com/b/security/archive/2014/04/10/microsoft-devices-and-services-and-the-openssl-heartbleed-vulnerability.aspx`, 2014.

[132] A. Mirian, Z. Ma, D. Adrian, M. Tischer, T. Chuenchujit, J. Mason, T. Yardley, R. Berthier, Z. Durumeric, J. A. Halderman, and M. Bailey. An Internet-Wide View of Publicly Accessible SCADA Devices. In *International Conference on Privacy, Security and Trust (PST)*, 2016.

[133] MITRE Corporation. Common Vulnerabilities and Exposures. `https://cve.mitre.org/`.

[134] MITRE Corporation. CWE: Common Weakness Enumeration. `https://cwe.mitre.org/`.

[135] MongoDB. MongoDB Response on Heartbleed OpenSSL Vulnerability. `http://www.mongodb.com/blog/post/mongodb-response-heartbleed-openssl-vulnerability`, 2014.

[136] T. Moore and R. Clayton. Ethical Dilemmas in Take-down Research. In *Financial Cryptography and Data Security (FC)*, 2011.

[137] Mozilla. Handling Mozilla Security Bugs. `https://www.mozilla.org/en-US/about/governance/policies/security-group/bugs/`, 2019.

[138] N. Munaiah and A. Meneely. Vulnerability Severity Scoring and Bounties: Why the Disconnect? In *International Workshop on Software Analytics (SWAN)*, 2016.

[139] K. Murchison. Heartbleed Warning - Cyrus Admin Passowrd Leak! `http://lists.andrew.cmu.edu/pipermail/info-cyrus/2014-April/037351.html`, 2014.

[140] E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan. The Design of Bug Fixes. In *International Conference on Software Engineering (ICSE)*, 2013.

[141] A. Nappa, R. Johnson, L. Bilge, J. Caballero, and T. Dumitraş. The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[142] A. Nappa, M. Z. Rafique, and J. Caballero. Driving in the Cloud: An Analysis of Drive-By Download Operations and Abuse Reporting. In *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2013.

[143] National Center for Biotechnology Information. Clinical Effectiveness of Partner Notification. `http://www.ncbi.nlm.nih.gov/books/NBK261439/`.

[144] National Institute of Standards and Technology. Special Publication 800-40 Revision 3: Guide to Enterprise Patch Management Technologies. `https://doi.org/10.6028/NIST.SP.800-40r3`, 2013.

[145] Netcraft. June 2014 Web Server Survey. `http://news.netcraft.com/archives/2014/06/06/june-2014-web-server-survey.html`, 2014.

[146] L. H. Newman. Equifax Officially Has No Excuse. `https://www.wired.com/story/equifax-breach-no-excuse/`, 2017.

[147] L. H. Newman. Equifax's Security Overhaul A Year After Its Epic Breach. `https://www.wired.com/story/equifax-security-overhaul-year-after-breach/`, 2018.

[148] E. Ng. Tunnel Fails after OpenSSL Patch. `https://lists.openswan.org/pipermail/users/2014-April/022934.html`, 2014.

[149] M. O'Connor. Google Services Updated to Address OpenSSL CVE-2014-0160 (the Heartbleed Bug). `http://googleonlinesecurity.blogspot.com/2014/04/google-services-updated-to-address.html`, 2014.

[150] M. Oltrogge, Y. Acar, S. Dechand, M. Smith, and S. Fahl. To Pin or Not to Pin—Helping App Developers Bullet Proof Their TLS Connections. In *USENIX Security Symposium*, 2015.

[151] P. Ondruska. Does OpenSSL CVE-2014-0160 Effect Jetty Users? `http://dev.eclipse.org/mhonarc/lists/jetty-users/msg04624.html`, 2014.

[152] OpenSSL Project Team. OpenSSL Security Advisory [07 Apr 2014]. `http://www.mail-archive.com/openssl-users@openssl.org/msg73408.html`, 2014.

[153] OpenSSL Project Team. OpenSSL Version 1.0.1g Released. `http://www.mail-archive.com/openssl-users@openssl.org/msg73407.html`, 2014.

[154] OpenVPN. OpenSSL Vulnerability—Heartbleed. https://community.openvpn.net/openvpn/wiki/heartbleed.

[155] Oracle. OpenSSL Security Bug—Heartbleed / CVE-2014-0160. http://www.oracle.com/technetwork/topics/security/opensslheartbleedcve-2014-0160-2188454.html, 2014.

[156] A. Ozment and S. E. Schechter. Milk or Wine: Does Software Security Improve with Age? In *USENIX Security Symposium*, 2006.

[157] L. Padron. Important Read – Critical Security Advisory And Patch for OpenSSL Heartbleed Vulnerability. http://blog.zimbra.com/blog/archives/2014/04/important-read-critical-security-advisory-patch-openssl-heartbleed-vulnerability.html, 2014.

[158] J. Park, M. Kim, B. Ray, and D.-H. Bae. An Empirical Study on Supplementary Bug Fixes. In *International Conference on Mining Software Repositories (MSR)*, 2012.

[159] PayPal. OpenSSL Heartbleed Bug—PayPal Account Holders are Secure. https://www.paypal-community.com/t5/PayPal-Forward/OpenSSL-Heartbleed-Bug-PayPal-Account-Holders-are-Secure/ba-p/797568, 2014.

[160] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar. VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits. In *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[161] W. Pinckaers. http://lekkertech.net/akamai.txt.

[162] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All Your iFRAMEs Point to Us. In *Usenix Security Symposium*, 2008.

[163] Publishers Clearing House. Stay Smart About The "Heartbleed" Bug With PCH! http://blog.pch.com/blog/2014/04/16/stay-smart-about-the-heartbleed-bug-with-pch/, 2014.

[164] A. Quach, Z. Wang, and Z. Qian. Investigation of the 2016 Linux TCP Stack Vulnerability at Scale. In *ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*, 2017.

[165] Rackspace. Protect Your Systems From "Heartbleed" OpenSSL Vulnerability. http://www.rackspace.com/blog/protect-your-systems-from-heartbleed-openssl-vulnerability/, 2014.

[166] K. Rankin. Sysadmin 101: Patch Management. Linux Journal. https://www.linuxjournal.com/content/sysadmin-101-patch-management, 2017.

[167] Rapid7. DNS Records (ANY) Dataset. `https://scans.io/study/sonar.fdns`, 2015.

[168] Red Hat. How to Recover from the Heartbleed OpenSSL Vulnerability. `https://access.redhat.com/articles/786463`, 2014.

[169] RhodeCode. Version Control Systems Popularity in 2016. `https://rhodecode.com/insights/version-control-systems-2016`, 2016.

[170] C. Rossow. Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[171] T. Saunders. ProFTPD and the OpenSSL "Heartbleed" Bug. `http://comments.gmane.org/gmane.network.proftpd.user/9465`, 2014.

[172] B. Say. Bleedingheart Bug in OpenSSL. `http://www.stunnel.org/pipermail/stunnel-users/2014-April/004578.html`, 2014.

[173] R. Seggelmann, M. Tuexen, and M. Williams. Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension. IETF Request for Comments (RFC) 6520, February 2012.

[174] I. Seidman. *Interviewing As Qualitative Research: A Guide for Researchers in Education and the Social Sciences*. Teachers College Press, 2013.

[175] M. Shahzad, M. Z. Shafiq, and A. X. Liu. A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles. In *International Conference on Software Engineering (ICSE)*, 2012.

[176] J. Sliwerski, T. Zimmermann, and A. Zeller. When Do Changes Induce Fixes. In *International Conference on Mining Software Repositories (MSR)*, 2005.

[177] K. Soska and N. Christin. Automatically Detecting Vulnerable Websites Before They Turn Malicious. In *USENIX Security Symposium*, 2014.

[178] M. Soto, F. Thung, C.-P. Wong, C. Le Goues, and D. Lo. A Deeper Look into Bug Fixes: Patterns, Replacements, Deletions, and Additions. In *International Conference on Mining Software Repositories (MSR)*, 2016.

[179] B. Stock, G. Pellegrino, F. Li, M. Backes, and C. Rossow. Didn't You Hear Me? Towards More Successful Web Vulnerability Notifications. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[180] B. Stock, G. Pellegrino, C. Rossow, M. Johns, and M. Backes. Hey, You Have a Problem: On the Feasibility of Large-Scale Web Vulnerability Notification. In *USENIX Security Symposium*, 2016.

[181] B. Stone-Gross, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[182] StopBadware. Request A Review. `https://www.stopbadware.org/request-review`, 2015.

[183] StopBadware and CommTouch. Compromised Websites: An Owner's Perspective. `https://www.stopbadware.org/files/compromised-websites-an-owners-perspective.pdf`, 2012.

[184] N. Sullivan. The Results of the CloudFlare Challenge. `http://blog.cloudflare.com/the-results-of-the-cloudflare-challenge`, 2014.

[185] K. Thomas, F. Li, C. Grier, and V. Paxson. Consequences of Connectivity: Characterizing Account Hijacking on Twitter. In *ACM Conference on Computer and Communications Security (CCS)*, 2014.

[186] Tumblr. Urgent Security Update. `http://staff.tumblr.com/post/82113034874/urgent-security-update`, 2014.

[187] Twitter. Unsafe Links on Twitter. `https://support.twitter.com/articles/90491`, 2015.

[188] United States Postal Service. Avoiding Heartbleed. `https://ribbs.usps.gov/importantupdates/HeartbleedArticle.pdf`, 2014.

[189] U.S. National Institute of Standards and Technology. CVSS Information. `https://nvd.nist.gov/cvss.cfm`.

[190] U.S. National Institute of Standards and Technology. National Checklist Program Glossary. `https://web.nvd.nist.gov/view/ncp/repository/glossary`.

[191] U.S. National Institute of Standards and Technology. National Vulnerability Database. `https://nvd.nist.gov/`.

[192] U.S. National Institute of Standards and Technology. NVD Data Feed. `https://nvd.nist.gov/download.cfm`.

[193] K. Vaniea, E. Rader, and R. Wash. Betrayed by Updates: How Negative Experiences Affect Future Security. In *ACM Conference on Human Factors in Computing Systems (CHI)*, 2014.

[194] K. Vaniea and Y. Rashidi. Tales of Software Updates: The Process of Updating Software. In *ACM Conference on Human Factors in Computing Systems (CHI)*, 2016.

[195] M. Vasek and T. Moore. Do Malware Reports Expedite Cleanup? An Experimental Study. In *Workshop on Cyber Security Experimentation and Test (CSET)*, 2012.

[196] M. Vasek and T. Moore. Identifying Risk Factors for Webserver Compromise. In *Financial Cryptography and Data Security (FC)*, 2014.

[197] N. F. Velasquez and S. P. Weisband. Work Practices of System Administrators: Implications for Tool Design. In *ACM Symposium on Computer Human Interaction for Management of Information Technology (CHiMiT)*, 2008.

[198] N. F. Velasquez and S. P. Weisband. System Administrators As Broker Technicians. In *ACM Symposium on Computer Human Interaction for the Management of Information Technology (CHiMiT)*, 2009.

[199] D. Y. Wang, S. Savage, and G. M. Voelker. Cloak and Dagger: Dynamics of Web search Cloaking. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[200] R. Wash and E. Rader. Too Much Knowledge? Security Beliefs and Protective Behaviors Among United States Internet Users. In *USENIX Symposium On Usable Privacy and Security (SOUPS)*, 2015.

[201] R. Wash, E. Rader, K. Vaniea, and M. Rizor. Out of the Loop: How Automated Software Updates Cause Unintended Security Consequences. In *USENIX Symposium On Usable Privacy and Security (SOUPS)*, 2014.

[202] C. Weston, T. Gandell, J. Beauchamp, L. McAlpine, C. Wiseman, and C. Beauchamp. Analyzing Interview Data: The Development and Evolution of a Coding System. *Qualitative Sociology*, 24(3):381–400, 2001.

[203] M. Wimmer. Removed Support for OpenSSL. `https://jabberd.org/hg/amessagingd/rev/bcb8eb80cbb9`, 2007.

[204] WordPress. Heartbleed Security Update. `http://en.blog.wordpress.com/2014/04/15/security-update/`, 2014.

[205] Z. Xu, B. Chen, M. Chandramohan, Y. Liu, and F. Song. SPAIN: Security Patch Analysis for Binaries Towards Understanding the Pain and Pills. In *International Conference on Software Engineering (ICSE)*, 2017.

[206] V. Yegneswaran, P. Barford, and V. Paxson. Using Honeynets for Internet Situational Awareness. In *ACM Workshop on Hot Topics in Networks (HotNets)*, 2005.

[207] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. Bairavasundaram. How do Fixes become Bugs? In *ACM European Conference on Foundations of Software Engineering (ESEC/FSE)*, 2011.

[208] S. Zaman, B. Adams, and A. E. Hassan. Security Versus Performance Bugs: A Case Study on Firefox. In *International Conference on Mining Software Repositories (MSR)*, 2011.

[209] ZDNet. Heartbleed Bug Affects Yahoo, OKCupid Sites. `http://www.zdnet.com/heartbleed-bug-affects-yahoo-imgur-okcupid-convo-7000028213/`, 2014.

[210] ZEDOinc. Customers and partners: none of the ZEDO sites or assets are affected by Heartbleed. `https://twitter.com/ZEDOinc/status/456145140503957504`, 2014.

[211] E. Zeng, F. Li, E. Stark, A. P. Felt, and P. Tabriz. Fixing HTTPS Misconfigurations at Scale: An Experiment with Security Notifications. In *Workshop on the Economics of Information Security (WEIS)*, 2019.

[212] H. Zhong and Z. Su. An Empirical Study on Real Bug Fixes. In *International Conference on Software Engineering (ICSE)*, 2015.

[213] Y. Zou, A. H. Mhaidli, A. McCall, and F. Schaub. "I've Got Nothing to Lose": Consumers' Risk Perceptions and Protective Actions after the Equifax Data Breach. In *USENIX Symposium on Usable Privacy and Security (SOUPS)*, 2018.

Appendix A

Survey and Interview Instruments for Chapter 4

Here we list the survey and interview questions used in our study of system administrator software updating, from Chapter 4. As described in Section 4.2 of that chapter, our study contained three phases: preliminary pilot interviews, large-scale surveys, and detailed semi-structure interviews.

A.1 Preliminary Phase - Pilot Interview Questions

Below we list the questions from our semi-structured pilot interviews (the preliminary phase of the study, as described in Section 4.2 of Chapter 4).

Job responsibilities and processes

1. Tell me more about your main job responsibilities (how does he/she keep machines up to date).
2. Tell me about any relationships you have with the vendors that develop the software updates for the programs your organization/employees depend on.
3. Can you walk me through your process of how you find out about an update?
4. Why do you find out about updates in this way?
5. How do you determine which updates to deploy on the machines you manage?
6. Tell me more about how this process differs for the types of machines you manage?
7. Why does your deployment process differ for different machines?
8. How does the process differ depending on who owns the machines, if at all?

9. Tell me more about how you install the software updates (manually, automatic, silent) you apply.
10. Why do you apply the software updates in this way?
11. Can you walk me through the process of testing whether an update will be compatible with the machines?
12. Why do you do this testing for the updates? Do you test all updates and why?

Software Update Information

1. Tell me about the information you currently receive when an update is available.
2. How do you usually receive this information?
3. Do you ever seek additional information about updates? Why or why not?
4. What are the main advantages of the current update information? Why?
5. What are the main disadvantages of the current update information? Why?
6. Which is the least important part of the current update information for you?
7. Which is the most important part of the current update information for you?

Securing the Users

1. Tell me about what you do to protect your users.
2. Tell me about what kinds of online hazards you are protecting them from.
3. Once an update is deployed how do you communicate the information to the end users?
4. What do you expect of the end users once the updates are released?
5. Can you tell me about the process of deciding what updates you can trust?

Software Updates in General

1. What updates are most important to you? Why?
2. What updates are least important? Why?
3. Tell me what cybersecurity means to you.
4. What are the most important things to consider to secure the network?
5. What are the least important things to consider to secure the network?

6. What are the main advantages of the current software updating process? Why?
7. What are the main disadvantages of the current software updating process? Why?
8. What changes would you want to make to software updates? Why?
9. Is there anything else you would like to tell us about how you manage software updates?

A.2 Phase One - Survey Questions

Below we list the questions from our survey (phase one of the study, as described in Chapter 4 Section 4.2).

1. How old are you?
   a) 18-25
   b) 26-35
   c) 36-45
   d) 46-55
   e) 56-65
   f) Over 65
   g) I do not wish to disclose
2. Which state do you live in?
3. What is your gender?
   a) Male
   b) Female
   c) Other
4. What is your annual income?
   a) Less than $25,000
   b) $25,000 to $34,999
   c) $35,000 to $49,999
   d) $50,000 to $74,999
   e) $75,000 to $99,999
   f) $100,000 to $124,999

   g) \$125,000 to \$149,999
   h) \$150,000 or more

5. What is your job title?
6. For how many years have you worked as a System Administrator in your current role?
7. For how many years have you worked as a System administrator before you entered your current role?
8. What is the highest level of education that you have completed?
   a) 12th grade or less
   b) High school degree or equivalent
   c) Some college, no degree
   d) Bachelor's degree
   e) Master's degree
   f) Other graduate degree
9. What was the subject area of your highest level of education (if above high school)?
10. What technical certifications, courses, or degrees have you completed, if any? You may paste entries from your resume or CV if you wish.
11. When did you complete these certifications or education? (Check all that apply)
    a) Before I took up my current role
    b) After I took up my current role
12. How have these technical certifications, courses, or degrees helped you complete your current role?
13. What is the industry of the organization that you work for?
14. How large is the organization that you work for?
    a) $\leq$10 employees
    b) 11 - 50 employees
    c) 51 - 100 employees
    d) 101 - 500 employees
    e) 501-2000 employees
    f) More than 2000 employees

15. What is the main purpose of the organization you work for?
16. How many machines/devices do you manage?
    a) *Sliding scale between 0 and 1000+*
17. What type of machines/devices do you manage? (Check all that apply)
    a) Laptops
    b) Desktops
    c) Servers
    d) Mobile devices
    e) Routers/network appliances such as firewall middleboxes
    f) Embedded devices/ Internet of Things
    g) Other: *free response*
18. What are the operating systems on the machines that you manage? (Check all that apply)
    a) Mac
    b) Windows
    c) Linux
    d) iOS
    e) Android
    f) Blackberry
    g) ChromeOS
    h) None
    i) Other: *free response*
19. What is the predominant operating system, if any?
    a) Mac
    b) Windows
    c) Linux
    d) iOS
    e) Android
    f) Blackberry
    g) ChromeOS
    h) Other: *free response*

20. What are these machines used for? (Check all that apply)
    a) Education or training
    b) Personal
    c) Research
    d) Servers
    e) Work
    f) Testing
    g) Other: *free response*
21. Which of the following applies to the machines you manage? (Check all that apply)
    a) The machines are used internally by the organization you work for
    b) The machines are used externally by customers of the organization you work for
    c) Other: *free response*
22. What updates are most important to you and why?
23. What updates are most important to your organization and why?
24. Are you solely responsible for updating the machines you manage?
    a) Yes
    b) No
    c) Other: *free response*
25. How many updates do you run on the machines that you manage per week?
    a) *Sliding scale between 1 and 500+*
26. How do you manage the updates across the machines/devices you manage? (Check all that apply)
    a) I log into each system to perform updates
    b) I use 3rd party software to manage the updates
    c) I write programs to manage updates
    d) I enable automatic updates
    e) Other: *free response*
27. What type of updates do you install regularly? (Check all that apply)
    a) Security updates

  b) Non-security related updates
  c) Other: *free response*

28. Select all of the security measures you take to protect your machines.
    a) Firewall
    b) Intrusion Detection System
    c) Intrusion Prevention System
    d) Antivirus System
    e) Security updates
    f) Different accounts with varying access (admin, regular, etc.)
    g) Access codes/Passwords
    h) Port scanners
    i) Vulnerability testing
    j) Backup and Disaster Recovery
    k) Other: *free response*
29. How are the security measures you use deployed? (Check all that apply)
    a) On the hosts
    b) On the network
    c) Other: *free response*
30. How do you find out about the updates you apply on the machines you manage? (Check all that apply)
    a) Online forums
    b) Security advisories
    c) Blogs
    d) News
    e) Social media
    f) RSS feeds
    g) Professional mailing lists
    h) Project mailing lists
    i) Direct notification from vendor
    j) Third-party service
    k) When the software pops up a notification

l) Other: *free response*

31. When do you apply security updates? (Check all that apply)
    a) As soon as they are released
    b) After testing
    c) On a regular cadence
    d) After a specific amount of time since its release has elapsed
    e) Applied automatically
    f) Other: *free response*
32. What is the reason for applying updates in the frequency described above?
33. When do you apply non-security related updates? (Check all that apply)
    a) As soon as they are released
    b) After testing
    c) On a regular cadence
    d) After a specific amount of time since its release has elapsed
    e) Applied automatically
    f) Other: *free response*
34. What is the reason for applying non-security related updates in the frequency described above?
35. What kind of testing do you do with updates (if any), before applying them to the machines/devices you manage? Please explain why.
36. How frequently do you find an update to cause problems on the machines you manage?
    a) Never
    b) Rarely
    c) Occasionally (every few update cycles)
    d) Frequently (most update cycles)
37. How do you become aware of any problems caused by updates that you install?
38. What, if any, is your process for rolling back or undoing updates that cause problems on the machines you manage?
39. What aspects or steps in your update management process work well for you?
40. What aspects or steps in your update management process are most challenging to handle?
41. What would help you to better manage software updates for multiple machines?

A.3 Phase Two - Interview Questions

Below we list the questions from our semi-structured interviews (phase two of the study, as described in Chapter 4 Section 4.2).

Job responsibilities and processes

1. Tell me more about the company you work for?
2. Tell me more about your main job responsibilities (how does he/she keep machines up to date)
3. How long have you worked in your job?
4. Have you had any training in IT? If so, tell me more about that.
5. Have you had any training in security? If so, tell me more about that.

Machines/Devices Managed

1. Does your organization have any security related policies for their machines?
2. How many machines/devices do you manage?
3. What kinds of machines/devices do you manage?
4. What are these machines used for?
5. Who are these machines used by?

Managing Software Updates for Multiple Machines

1. Does your company have any policies on software updates for their machines?
2. How do you handle security for these machines?
3. How often do you update these machines? Does the frequency differ for different machines? If so, why?
4. Who do you have to notify about updates that you are applying? Why?
5. In an average week, how many hours do you spend dealing with software updates?
6. Can you walk me through your process of how you find out about an update?
7. What are the advantages of using this process?
8. What are the disadvantages of using this process?

9. How do you determine which updates to deploy on the machines you manage?
10. When do you apply updates for the machines you manage? Why?
11. What is your process for applying updates on the machines you manage?
12. Tell me more about how this process differs for the types of machines you manage.
13. Why does your deployment process differ for different machines?
14. How does the process differ depending on who owns the machines, if at all?
15. Tell me more about how you install the software updates (manual, automatic, silent) you apply.
16. Why do you apply the software updates in this way?
17. Do you use any tools/programs to help you manage updates on multiple devices? What are these tools? Why do you use them?
18. Do you test whether an update will be compatible with the machines you manage in any way? How so?
19. Why do you do this testing for the updates? Do you test all updates and why?
20. How do you track which updates different machines need?
21. Do you prioritize any particular type of updates for any machines? Why/why not?
22. How do you track how well updates have been installed on different machines?
23. If any update requires a restart, what is your process for managing the restart?
24. Do you have to notify anyone about updates that you have applied or are about to apply?

Software Update Information

1. Tell me about the information you currently receive when an update is available.
2. How do you usually receive this information?
3. Do you ever seek additional information about updates? Why or why not?
4. What are the main advantages of the current update information? Why?
5. What are the main disadvantages of the current update information? Why?
6. Which is the least important part of the current update information for you?
7. Which is the most important part of the current update information for you?

8. What improvements would you make to the information that is included with current updates?

Securing the Users

1. Who are the users that you manage machines for?
2. Tell me about what you do to protect your users.
3. Do you use any technical solutions to protect users?
4. Do you use any educational solutions for protecting your users?
5. Are these solutions driven by your own or company policy? Tell me more about that.
6. Tell me about what kinds of online hazards you are protecting them from.
7. Once an update is deployed how do you communicate the information to the end users?
8. What are your responsibilities for handling updates for your users?
9. What are the responsibilities of your users for handling updates?
10. Can you tell me about the process of deciding what updates you can trust?

Software Updates in General

1. What updates are most important to you? Why?
2. What updates are most important to your organization? Why?
3. How does your organizational policy influence how you manage updates if at all?
4. What updates are least important to you? Why?
5. What updates are least important to your organization? Why?
6. Tell me what cybersecurity means to you.
7. What are the most important things to consider to secure your machines?
8. What are the least important things to consider to secure your machines?
9. What are the main advantages of your current software updating process? Why?
10. What are the main disadvantages of your current software updating process? Why?
11. What would your ideal way to handle software updates be? Why? What changes would you want to make to software updates themselves? Why?
12. Is there anything else you would like to tell us about how you manage software updates?

Appendix B

Survey and Notification Messages for Chapter 9

Here we provide the survey and notification messages from our study of effective Internet-scale administrator vulnerability notifications, from Chapter 9. As described in Section 9.2 of that chapter, we sent notifications about three security issues: overly permissive IPv6 firewalls, exposed industrial control systems (ICS), and open DDoS amplifiers. For each security concern, we tested sending terse messages, terse messages with a link to an information page, and a verbose message. Additionally, all of our notifications contained an anonymous and optional survey to obtain feedback from contacted administrators.

B.1 Anonymous and Optional Security Notifications Survey

Help us better understand the factors surrounding security notifications by providing anonymous feedback in this survey. Each question is optional, so answer the ones you feel comfortable answering. Thank you!

1. Was your organization aware of the security issue prior to our notification?
2. Did your organization take prior actions to resolve the security issue before our notification?
3. Is your organization planning on resolving the security issue?
4. Do you feel it was acceptable for us to detect the security issue?
5. Do you feel it was acceptable for us to notify your organization?
6. Would your organization want to receive similar security vulnerability/misconfiguration notifications in the future?
7. Did we notify the correct contact?

B.2 IPv6 Notification: Terse with Link

Subject: [RAND#] Potentially Misconfigured IPv6 Port Security Policies

Body: Computer scientists at the University of Michigan, the University of Illinois Urbana-Champaign, and the University of California Berkeley have been conducting Internet-wide scans to detect IPv4/IPv6 dual-stack hosts that allow access to services via IPv6, but not IPv4. This likely indicates a firewall misconfiguration and could be a security vulnerability if the services should not be publicly accessible. We have attached a list of hosts that are potentially vulnerable on your network.

[LINK: More information is available at https://security-notifications.cs.berkeley.edu/[RAND#]/ipv6.html.]

Thank you,

Berkeley Security Notifications Team

Help us improve notifications with anonymous feedback at: https://www.surveymonkey.com/r/Q2HLJ5D.

B.3 IPv6 Notification: Verbose

Subject: [RAND#] Potentially Misconfigured IPv6 Port Security Policies

Body: During a recent study on the network security policies of IPv4/IPv6 dual-stack hosts, computer scientists at the University of Michigan, the University of Illinois Urbana-Champaign, and the University of California Berkeley have been conducting Internet-wide scans to detect IPv4/IPv6 dual-stack hosts that allow access to services via IPv6, but not IPv4. This likely indicates a firewall misconfiguration and could be a security vulnerability if the services should not be publicly accessible. We have attached a list of hosts that are potentially vulnerable on your network (as determined by WHOIS information).

For each dual-stack host, we test whether popular services (e.g., SSH, Telnet, and NTP) are accessible via IPv4 and/or IPv6 using a standard protocol handshake. For ICMP this is an echo request, for TCP it is a SYN segment, and for UDP this is an application-specific request (e.g., DNS A query for ‘www.google.com’ or an NTP version query). We do not exploit any vulnerabilities, attempt to login, or access any non-public information.

The protocols we scanned are popular targets for attack and/or can be used to launch DDoS attacks when left publicly available to the Internet. We suspect they are misconfigured and are notifying you because hosts rarely offer services on IPv6 that are not offered on IPv4, and we believe these services may have been left exposed accidentally. This is a common occurrence when administrators forget to configure IPv6 firewall policies along with IPv4 policies.

If these IPv6-only accessible services should not be accessible to the public Internet, they can be restricted by updating your firewall or by disabling or removing the services. If none of your systems use IPv6, you can also disable IPv6 on your system. Make sure your changes are persistent and will not be undone by a system reboot.

More information is available at https://security-notifications.cs.berkeley.edu/[RAND#]/ipv6.html.

Thank you,

Berkeley Security Notifications Team

Help us improve notifications with anonymous feedback at: https://www.surveymonkey.com/r/Q2HLJ5D

B.4 ICS Notification: Terse with Link

Subject: [RAND#] Vulnerable SCADA Devices

Body: Computer scientists at the University of Michigan and the University of California Berkeley have been conducting Internet-wide scans to detect publicly accessible industrial control (SCADA) devices. These devices frequently have no built-in security and their public exposure may place physical equipment at risk for attack. We have attached a list of SCADA devices on your network that are publicly accessible.

[LINK: More information is available at https://security-notifications.cs.berkeley.edu/[RAND#]/ics.html.]

Thank you,

Berkeley Security Notifications Team

Help us improve notifications with anonymous feedback at: https://www.surveymonkey.com/r/ZC7BVW5

B.5 ICS Notification: Verbose

Subject: [RAND#] Vulnerable SCADA Devices

Body: During a recent study on the public exposure of industrial control systems, computer scientists at the University of Michigan and the University of California Berkeley have been conducting Internet-wide scans to detect publicly accessible industrial control (SCADA) devices. These devices frequently have no built-in security and their public exposure may place physical equipment at risk for attack. We have attached a list of SCADA devices on your network (as determined by WHOIS information) that are publicly accessible.

We scan for potentially vulnerable SCADA systems by scanning the full IPv4 address space and attempting protocol discovery handshakes (e.g., Modbus device ID query). We do not exploit any vulnerabilities or change any device state.

SCADA protocols including Modbus, S7, Bacnet, Tridium Fox, and DNP3 allow remote control and monitoring of physical infrastructure and equipment over IP. Unfortunately, these protocols lack critical security features, such as basic authentication and encryption, or have known security vulnerabilities. If left publicly accessible on the Internet, these protocols can be the target of attackers looking to monitor or damage physical equipment, such as power control, process automation, and HVAC control systems.

SCADA services are not designed to be publicly accessible on the Internet and should be maintained on an internal, segmented network, or otherwise protected by a firewall that limits who can interact with these hosts. Make sure your changes are persistent and will not be undone by a system reboot.

More information is available at https://security-notifications.cs.berkeley.edu/[RAND#]/ics.html.

Thank you,

Berkeley Security Notifications Team

Help us improve notifications with anonymous feedback at: https://www.surveymonkey.com/r/ZC7BVW5

B.6 DDoS Amplification Notification: Terse with Link

Subject: [RAND#] Vulnerable DDoS Amplifiers

Body: Computer scientists at George Mason University and the University of California Berkeley have been detecting open and misconfigured services that serve as amplifiers for distributed denial-of-service (DDoS) attacks. Attackers abuse these amplifiers to launch powerful DDoS attacks while hiding the true attack source. We have attached a list of hosts that are potentially vulnerable on your network.

[LINK: More information is available at https://security-notifications.cs.berkeley.edu/[RAND#]/amplifiers.html.]

Thank you,

Berkeley Security Notifications Team

Help us improve notifications with anonymous feedback at: https://www.surveymonkey.com/r/Y99J8K8

B.7 DDoS Amplification Notification: Verbose

Subject: [RAND#] Vulnerable DDoS Amplifiers

Body: During a recent study on distributed denial-of-service (DDoS) attacks, computer scientists at George Mason University and the University of California Berkeley have been conducting Internet-wide scans for open and misconfigured services that serve as amplifiers for DDoS attacks. Attackers abuse these amplifiers to launch powerful DDoS attacks while hiding the true attack source. We have attached a list of hosts that are potentially vulnerable on your network (as determined by WHOIS information).

We detect amplifiers by monitoring hosts involved in recent DDoS attacks and checking whether these hosts support the features used for launching an attack (e.g., NTP monlist or recursive DNS resolution). We do not exploit any vulnerabilities or attempt to access any non-public data on these servers.

DDoS attacks are often conducted by directing an overwhelming amount of network traffic towards a target system, making it unresponsive. Amplifiers are services that send large amounts of data in response to small requests. Attackers leverage these in DDoS attacks by spoofing traffic to the amplifier, forging it to look as if it came from the attacker's target. Amplifiers then respond to the target with a large response that overwhelms the target. Publicly accessible amplifiers are constantly abused by attackers to conduct the DDoS attacks for them while hiding the tracks of the real attacker.

These amplifiers can be avoided by disabling the application or updating your firewall to block the application port or restrict the IP addresses that can access it. More specifically, Chargen should be closed as it is rarely useful and is inherently an amplifier. If left open, DNS should be configured to restrict who can make recursive requests, and NTP should be configured to disable the monlist functionality. Make sure your changes are persistent and will not be undone by a system reboot.

More information is available at https://security-notifications.cs.berkeley.edu/[RAND#]/amplifiers.html.

Thank you,

Berkeley Security Notifications Team

Help us improve notifications with anonymous feedback at: https://www.surveymonkey.com/r/Y99J8K8