Title
Detecting Credential Compromise in Enterprise Networks

Author
Javed, Mobin

Publication Date
2016

Peer reviewed|Thesis/dissertation

Detecting Credential Compromise in Enterprise Networks

by

Mobin Javed

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 Adityanand Guntuboyina

Fall 2016

Detecting Credential Compromise in Enterprise Networks

Mobin Javed

Abstract

Detecting Credential Compromise in
Enterprise Networks

by

Mobin Javed

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Vern Paxson, Chair

Secure remote access is integral to the workflow of virtually every enterprise today. It is also an avenue ripe for network infiltration—attackers who can steal network-login credentials can often readily penetrate a site's perimeter security to obtain a persistent foothold within the network. Once inside, they can often further their access by escalating privileges and moving laterally, potentially operating for months, all the while remaining undetected under the guise of a legitimate user. Such threats can prove hugely expensive and damaging to sites, fueling APT campaigns and enormous data breaches. For example, the 2013 theft from Target of 40,000,000 credit card numbers began with attackers compromising remote-access credentials of one of its contractors [39], and the 2015 breach of SSNs and biometric data of millions of government employees likewise stemmed from a stolen credential [13].

This dissertation aims to advance the state of credential compromise detection for enterprise settings. We leverage several years worth of real-world network logs from the Lawrence Berkeley National Laboratory (LBNL) in order to develop systems for detecting: (*i*) stealthy, distributed brute-force attacks that compromise password-based credentials by attempting a number of guesses against the site's servers—these attacks proceed in a stealthy fashion by distributing the brute-force work across an army of machines, such that each individual host only makes a few attempts, and thereby becomes hard to differentiate from failed attempts of legitimate users, and (*ii*) *anomalous* logins indicating that a user's login credentials may have been potentially compromised—either through brute-forcing attacks or broadly through other vectors (phishing attacks and credential-stealing malware).

For the detection of stealthy brute-force attacks, we first develop a general approach for flagging distributed malicious activity in which individual attack sources each operate in a stealthy, low-profile manner. We base our approach on observing statistically significant changes in a parameter that summarizes *aggregate* activity, bracketing a distributed attack

in time, and then determining which sources present during that interval appear to have coordinated their activity. We then apply this approach to the problem of detecting stealthy distributed SSH brute-forcing activity, showing that we can model the process of legitimate users failing to authenticate using a beta-binomial distribution, which enables us to tune a detector that trades off an expected level of false positives versus time-to-detection. Using the detector we study the prevalence of distributed brute-forcing, finding dozens of instances in an extensive eight-year dataset collected at the Lawrence Berkeley National Lab. Many of the attacks—some of which last months—would be quite difficult to detect individually. While a number of the attacks reflect indiscriminant global probing, we also find attacks that targeted only the local site, as well as occasional attacks that succeeded.

For the detection of anomalous logins, we first extensively characterize the volumes and diversity of login activity at LBNL's network, with the goal of engineering features that with good confidence can serve as indicators of compromise. We then develop a practical rule-based detector that leverages the global view of the network as well as historical profile of each user to flag *potentially* compromised credentials. On average, our detector raises 2–3 alarms per week—a reasonable analyst workload for an enterprise with several thousand users. To understand these alarms, we worked with the site operators, who deemed the top ten percent of instances suspicious enough to merit an inquiry to the affected user. Our detector successfully flagged a *known* compromised account and discovered an instance of a (benign) shared credential in use by a remote collaborator.

In summary, this dissertation develops approaches to detect both stealthy brute-force attempts and anomalous successful logins among hundreds of thousands of logins in an enterprise network's activity. Using our detectors, we show that stealthy brute-force attacks that target password-based credentials have been around for years, and that attackers do occasionally succeed in compromising the credentials. Our work makes advances in detecting such stealthy break-ins that, if they remain undetected, can prove hugely expensive for sites.

To my parents and grandparents

Contents

List of Figures

List of Tables

Acknowledgments

I wish to express my deepest thanks to a number of people whose generous help, enormous support, and sincere guidance has made this dissertation possible.

First and foremost, I am indebted to Vern Paxson, my advisor and my mentor through these years. Perhaps words will fall short to describe the gratitude and respect that I have for Vern for his role, but here is my humble effort. Vern's advising has been an amazing mix of rigor, support, and fun. His discipline is inspiring and his dedication towards his students unparalleled. I have immensely enjoyed working with him, and to this day he remains a continued source of inspiration in how he conducts both his teaching and research. Thank you Vern for guiding my research, teaching me the art of sound Internet measurement, and helping me discover a joy for data analysis done well. Thank you for the freedom to explore various topics that interested me during graduate school. Your enthusiasm and keen interest in the research efforts made this road a very exciting one. The rigor in your advising in all aspects of research, be it numbers, methodology, or writing, has greatly shaped my development and growth as a researcher. For all your time and energy, my sincere thanks!

This dissertation, being a heavily empirical undertaking, would not have been possible without the gracious data support from the Lawrence Berkeley National Lab. I would like to thank Aashish Sharma, my operational contact at LBNL, for his extra-ordinary patience in working with the data preparation scripts, for running down various puzzles that arose during the course of this research, and for going out of the way to help in times of critical deadlines. This work has also benefitted from conversations with Aashish and Partha Bannerjee on the operational security practices and challenges at LBNL.

I would also like to thank David Wagner and Adityanand Guntuboyina for serving on my committee, and for reading through drafts of my dissertation. A special thanks to Adam Bloniarz in the statistics department for his help with the *Cumulative Sum* change detection framework, which forms the basis for one of the main parts of this work.

Several people, although not directly involved in the work, contributed in various ways through my graduate school journey, and I would like to express my gratitude to them.

First, to my office-mate Matthias Vallentin, who has been a great source of reference and advice for many matters in graduate school, starting from the day I arrived in Berkeley. Thanks Matthias for being an incredibly helpful office-mate, and for all the times I could simply follow your foot steps instead of spending energy figuring things out, for always welcoming discussions, whether research or beyond, and for introducing me to various new tools and ideas over the years. (Most non-plot figures in this thesis were made using `OmniGraffle`, which Matthias introduced me to, and the thesis uses his modified version of the *ucbthesis* `LaTeX` template for typesetting, which again he was very kind to share).

To my two peer collaborators, Sheharbano Khattak and Sakshi Jain, with whom I have worked very closely. These collaborations, supervised by Vern, played a significant role in

my journey as a budding researcher and the learnings that resulted from them had a very positive influence for my dissertation work.

Sheharbano, working with you has been instrumental in keeping alive the spirit of my Ph.D. Only your craziness and determination could make several collaborations across various time zones possible. It was amazing energy and a lot of fun. Thanks for being my partner in several exercises in Internet measurement, and for an endless supply of energetic music, which of course this dissertation has greatly benefitted from.

To Sakshi Jain, whom I luckily met through the Big-Sister mentoring program at Berkeley, and subsequently worked with on class projects and her Master's thesis. It is co-incidental that we met through this program and soon developed a bond that is very much like real sisters. Thanks Sakshi for all your support, for always being ready for a cup of chai, and for several crucial career and life discussions. It is almost hard to imagine how graduate school would have looked without a truly genuine friend and an awesome collaborator like you.

My thanks also extends to several other colleagues in the security research groups at UC Berkeley and ICSI, with whom I have had the opportunity to discuss various aspects of this as well as other works, and with whom I have grown from a first year student to a graduating one. Thanks Brad Miller, Devdatta Akhawe, Justin Samuel, and Paul Pearce for several helpful discussions and critiques. Thanks post-docs at ICSI, Narseo Vallina-Rodriguez and Srikanth Sundaresan, for extended discussions, fun collaborations, and for enlightening me with your experiences and perspectives on academia.

I am also lucky to have found a great many friends whose support and encouragement has brought me loads of strength and has provided me with a community much needed to produce this work: Anshul Jain, Abhishek Kar, Balu Vellanki, Divya Krishnakumar, Dharashree Panda, Floraine Berthouzoz (late), Garvit Juniwal, Jose Manuel Falerio, Mohit Bansal, Sakshi Jain, Shikha Yadav, Reid Felice, and Wendy Bloom, a heartfelt thank you to all of you for making my grad school journey wonderful—for all the life, laughter, wisdom, comfort, great food, amazing hikes, and fun discussions that your friendship brought in this journey.

Two friends deserve a special mention for their unwavering support despite being remote: Junaid Khalid and my sister Yusra Javed, who were always a phone call away should I need help thinking any ideas through or need any piece of writing to be proof-read. They also very kindly read through early drafts of parts of this dissertation. Thanks Junaid and Yusra!

My dissertation has also benefitted greatly from the vibrant energy that Berkeley provides. Most of my papers were written at these cafes filled with focused and driven people: *Au Coquelet*, *Asha Tea House*, and *Berkeley Espresso*. I wish to acknowledge the contribution of these cafes in providing me the reflection and writing environment. I would also like to thank the staff at ICSI: Maria Quintana, Cindy Ngu, Chris Switzer, and Jaci Considine, who have always smilingly helped me with logistics, and whose bright smiles at the reception desk have been a source of positive energy in my coffee-fetching trips from office to the ICSI kitchen.

Lastly but most importantly, to my family. To my parents who taught me to dream and achieve, who taught me the power of persistence, who encouraged me and whole-heartedly supported me in pursuing graduate school, who have very patiently endured my absence from home all these years, and have been the most understanding about my academic commitments. I know it has not been easy for you. Thank you abbu and ammi! To my sisters, Iqra, Yusra, and Sana for their kind love and support, and to my sunshine, my little brother Talha, whose ever-present positivity and outlook on things has often been the source of regaining perspective whenever the going got tough and the destination seemed out of sight. Thanks for always being there all the way from Pakistan!

Chapter 1

Introduction

Credentials unlock the doors to a variety of resources on the Internet today. We use them to protect access to our emails, online social networks, medical records, bank accounts, work systems, and with the rise of Internet of things, even our smart TVs and light switches at home. Needless to say, these credentials often guard a wealth of sensitive data and protected resources. It is no wonder then that they provide for a very attractive target for criminals [17]—with illicitly acquired credentials in hand, attackers can pursue a range of possibilities: extortion [14, 4], scamming the victim's contacts [41, 40], spamming [11, 26], compromising more accounts [26, 48], spreading mis-information [5, 7], and silently working their way up to massive data breaches [13, 39].

A special class among the array of credentials in use today are *network-login* credentials, which organizations employ to authenticate users for remote-access to the enterprise networks. Theft of these credentials can endanger a site to potential infiltration—attackers who can compromise network-login credentials can often readily penetrate a site's perimeter security to obtain a persistent foothold within the network. Once inside, they can then further their access by escalating privileges and moving laterally across systems in the network, potentially operating for months all the while remaining undetected under the guise of a legitimate user. Such threats can prove hugely expensive and damaging to enterprises, fueling Advanced Persistent Threat (APT) campaigns and enormous data breaches. As recent striking examples, the 2013 theft from Target of 40,000,000 credit card numbers began with attackers compromising remote-access credentials of one of its contractors [39], and the 2015 breach of SSNs and biometric data of millions of government employees likewise stemmed from a stolen credential, this time one used to penetrate the US Office of Personnel Management [13].

One line of defense against the threat of stolen credentials is to use stronger authentication mechanisms which can withstand such compromise. Passwords and key-based credentials, the two common forms of authentication that organizations employ today, are vulnerable to re-use—once in attacker's hands, it is simple enough for her to log in as the legitimate user and freely access the victim's data and resources. In comparison, one-time passwords and multi-

factor authentication provide resiliency against compromise by requiring the authenticating user to provide a proof of possession of something that the user *has* (such as a device or a token generator), in addition to something that the user *knows*, thereby making it hard for the attacker to re-use stolen credentials. However, these advanced authentication solutions suffer significant usability and deployment challenges. Many sites, such as universities and national laboratories, still find themselves struggling to deploy these solutions comprehensively due to collaborations often spanning multiple institutions. Given these difficulties, investing in the second line of defense, i.e., monitoring credential usage for indicators of compromise, can play a crucial role in securing enterprise networks today. This dissertation focuses on advancing this second line of defense for enterprise settings.

1.1 Threat Vectors and Detection Opportunities

In order to monitor sites for credential compromise, it is important to understand the vectors that attackers employ to steal network credentials today: brute-forcing (trying out a number of passwords in the hope of guessing the right one), phishing (luring a victim to enter credentials at a legitimate-looking website under attacker's control), and infecting hosts with credential-stealing malware [3]. The first two vectors, brute-forcing and phishing, reflect a threat primarily for password-based credentials and are widespread due to the relative ease of conducting the attacks. They compromise credentials by exploiting human weaknesses: users either choose weak passwords that are easy to brute-force or are gullible enough to fall for phishing. The third vector, credential-stealing malware, presents a wider threat for both passwords and key-based credentials, but is relatively less common due to the difficulty in exploiting machine weaknesses—successfully exploiting this vector may involve finding zero-day vulnerabilities (a vulnerability that has not been discovered before).

Given these threat vectors, sites monitoring their network activity have two opportunities to detect compromised credentials: (*i*) at the time of compromise, and (*ii*) at the time of subsequent account access by the attacker. With respect to the former angle, sites can always monitor the activity resulting from the brute-force vector (by definition these attacks involve checking a number of passwords against the site's servers). However, attacks involving the other two vectors, phishing and credential-stealing malware, can hit the users both when they are on-site (in which case the site has an opportunity to detect the attack) and when they are off-site (no visibility into the attack for the site). The second line of monitoring, i.e., looking for anomalous logins and suspicious account activity, can detect compromise regardless of *how* and *where* the credentials are compromised.

1.2 The Shift Towards Stealthy Attacks

Over the past decade, the credential stealing attacks have become more sophisticated, and thereby harder to detect. This is in part due to the increased sophistication of adversaries,

both in response to the maturing Internet defenses and due to new powerful adversaries surfacing (such as nation-state actors), and in part due to the rise of an *underground economy* that facilitates access to the tools/resources needed for the attacks. Consider brute-force attacks for example: attackers today can readily rent a botnet (an army of machines) to spread the brute-force attempts across thousands of machines, thereby making the attacks *distributed* and potentially *stealthy* in nature. If each of the attack hosts makes only a few attempts, the low-rate activity becomes very difficult to differentiate from the occasional failed attempts of legitimate users. Similarly, with defenses against phishing attacks improving, attackers are shifting to *spear-phishing* attacks—highly targeted emails crafted such that they appear to be from one of the victim's contacts.

Therefore, while improvements in defenses have made credential stealing harder for attackers, the adversaries we face today are powerful and sophisticated. Viewed from a defender's perspective, this arms race has pushed the attacks to a sophisticated space which is harder to monitor and detect.

1.3 Compromise Detection: An Overview and Open Problems

This dissertation aims to advance the state of credential compromise detection by developing detectors that analyze *login attempts* to enterprise servers and classify each of the attempts as resulting from either the legitimate user, a brute-force attacker attempting to break-in, or an attacker who has successfully compromised the account. Our work focuses on the sub-space of credential compromise that is detectable from login attempts: the brute-force attack vector and *anomalous* logins reflecting successful compromise. Within this space, our main contribution is to advance the detection capability to stealthy compromises. Other work in this field tackles detection of other attack vectors, and generally analyzes site activity beyond login attempts: detecting phishing attacks [24, 27, 29, 19, 21, 49], a large body of work on detecting malware [25, 38, 20, 33, 34, 46], and detecting account compromise by analyzing post-login account activity [26, 48, 56].

Along the lines of brute-force detection in enterprise settings, a range of threshold-based approaches exist to identify individual brute-force hosts [1, 31, 2, 37, 12]. These approaches monitor each host's activity and raise an alarm when the host crosses a certain number of failed attempts in a short period of time. Although effective for detecting and blocking high-rate brute-forcers, these approaches fail to detect brute-force attempts that are part of low-rate distributed attacks. Some works also propose techniques that use only network (flow) data to detect when a site is under a brute-force attack [36, 42, 54], but fall short of evaluation on real-world data, and questions about accuracy and practical deployment remain unaddressed in these works. None of the existing approaches address the detection of distributed brute-force attacks, and more importantly are completely blind to compromise(s) when such attacks are stealthy in nature.

Even if the sites can detect individual attack hosts in a distributed attack point-wise (due to high-rate activity), they lack the capability to piece together that the hosts are working in a *coordinated* fashion on behalf of an individual attacker. The capability to tell when a site is under a coordinated attack is important for *situational awareness*— to understand the nature and power of the adversary and the scale of resources they bring to the operation. Given the lack of a practical detection technique, we as a community also lack insight into broader questions, such as how prevalent are such coordinated brute-force attacks, and how often are these attacks stealthy in nature? Similarly, how often are the attacks indiscriminate, i.e., they hit the Internet wholesale in the hope of compromising credentials on *any* site, versus how often do they target particular sites?

Existing approaches for identifying *anomalous* network logins in enterprise settings suffer poor performance [48, 32], generating dozens of false alarms per day. Further, techniques for web-login/VPN credentials compromise detection [28, 58] are not directly applicable to enterprise settings, because of the difference in how users use remote-access accounts versus how they use web and VPN accounts. For example, it is not uncommon for enterprise users to *concurrently* log in from geographically distant servers in a normal workflow—making it problematic to leverage features such as "suspicious geo-location sequence" that can provide a powerful signal in other settings. The existing literature lacks a comprehensive characterization of the diversity of login activity in enterprise settings. Such a study can serve as a foundation for developing features and transformations thereof that have the potential of detecting anomalous login events, while maintaining a manageable workload for analysts.

1.4 Thesis Contributions

In this dissertation, we leverage several years worth of real-world network logs from the Lawrence Berkeley National Laboratory in order to develop systems for detecting: (*i*) stealthy, distributed brute-force attacks that compromise password-based credentials by attempting a number of guesses against the site's servers—each host proceeding in a low-rate manner and thereby hard to differentiate from failed attempts of legitimate users, and (*ii*) *anomalous* logins indicating that a user's login credentials may have been potentially compromised—either through brute-forcing attacks or broadly through other vectors (phishing attacks and credential-stealing malware).

(i) Measurement and Detection of Stealthy, Distributed Brute-force Attacks:

Moving away from the idea of point-wise threshold based detection, we propose a general approach for detecting distributed malicious activity in which individual attack sources each operate in a stealthy, low-profile manner. We base our approach on observing statistically significant changes in a parameter that summarizes *aggregate* activity, bracketing a distributed attack in time, and then determining which sources present during that interval appear to have coordinated their activity. We apply this approach to the problem of detecting stealthy distributed SSH brute-forcing activity, showing that we can model the process of legitimate

users failing to authenticate using a beta-binomial distribution, which enables us to tune a detector that trades off an expected level of false positives versus time-to-detection. Using the detector we study the prevalence of distributed brute-forcing, finding dozens of instances in an extensive 8-year dataset collected from a site with several thousand SSH users. Many of the attacks—some of which last months—would be quite difficult to detect individually. While a number of the attacks reflect indiscriminant global probing, we also find attacks that targeted only the local site, as well as occasional attacks that succeeded.

(ii) Detection of Compromised Credentials:

In the second part, we look at the detection of anomalous successful logins by analyzing the properties of login attempts, regardless of the threat vector and whether the credential is compromised on-site or off-site. We first extensively characterize the volumes and diversity of login activity at our network, with the goal of engineering features that with good confidence can serve as indicators of compromise. We then develop a practical rule-based detector that leverages the global view of the network as well as a historical profile of each user to flag *potentially* compromised credentials. On average, our detector raises 2–3 alarms per week—a reasonable analyst workload for an enterprise with several thousand users. To understand these alarms, we worked with the site operators, who deemed the top ten percent of instances suspicious enough to merit an inquiry to the affected user. Our detector successfully flagged a *known* compromised account and discovered an instance of a (benign) shared credential in use by a remote collaborator.

In summary, this dissertation develops approaches to detect both stealthy brute-force attempts and anomalous successful logins among hundreds of thousands of logins in an enterprise network's activity. Using our detectors, we show that stealthy brute-force attacks that target password-based credentials have been around for years, and that attackers do occasionally succeed in compromising network-login credentials. Our work makes advances in detecting such stealthy break-ins that, if they remain undetected, can prove hugely expensive for sites.

1.5 Thesis Outline

The rest of the thesis is organized as follows:

- §2 reviews the existing literature. We discuss detection approaches and measurement studies on compromised credentials in the context of enterprise networks as well as in the context of online social networks.
- §3 describes the datasets we use in the development and evaluation of our detectors. The chapter also sketches the data slicing and filtering we performed in order to prepare the datasets for this work.
- §4 discusses the design and evaluation of our distributed brute-force attacks detector. We also characterize in detail the attacks we find by running the detector on an extensive

eight-year dataset from LBNL. We correlate these attacks with datasets from different vantage points to establish whether the attacks are indiscriminate or targeted.

- §5 discusses the design and evaluation of our *anomalous* logins detector. The chapter starts with a characterization of the enterprise login activity using several months of logs, and sketches techniques to identify the context of logins. We then illustrate the engineering of building blocks (*features*) for our detector, and how we use them to design our final detection framework.

- §6 concludes the dissertation and frames future work.

Chapter 2

Related Work

The literature relevant to this thesis lies both along the dimensions of detection and of measurement. First, to understand the state of the art in the detection of credential compromise, three areas of work are of relevance: (*i*) detection of SSH brute-force attacks, (*ii*) detection of coordinated attacks, and (*iii*) detection of compromised credentials by identifying suspicious account activity (both anomalous logins and post-login activity). Second, studies on the measurement and analysis of such attacks and incidents are relevant for understanding the prevalence of the threat and how it has evolved over time.

We review the existing work in detail below, organizing it according to the two subproblems of this thesis.

2.1 Detection and Measurement of SSH Brute-force / Coordinated Attacks

With regard to SSH brute-forcing, host-based detection techniques such as DenyHosts [2], BlockHosts [1], BruteForceBlocker [31], fail2ban [37], and sshguard [12] block hosts that cross a user-configurable threshold for failed attempts in a specified amount of time. Since this threshold must be set high enough to to avoid inconvenience to legitimate users who forget or mistype their usernames/passwords, these point-wise approaches fail to detect low-rate brute-forcing activity.

Other work has developed network-based approaches. Kumagai et al. propose an increase in the number of DNS PTR record queries to detect SSH dictionary attacks [42]. This increase results from the SSH server logging the fully qualified domain names of the SSH clients attempting access. This work does not discuss how to establish detection thresholds, nor does it present an evaluation of the system's accuracy. Vykopal et al. develop flow signatures for SSH dictionary attacks [54]. They show that a large number of short flows having a few bytes transferred in both directions and appearing together in a short duration of time are indicative of failed login attempts, providing the means to then detect brute-force attacks

from flow data. Hellemons also studied the possibility of using only flow data to detect SSH brute-force attacks, modeling the brute-force attacks as consisting of three phases: scanning, brute-force and die-off (in case of successful compromise) [36]. They monitor the ranges of three parameters—flows-per-second, packets-per-flow and bytes-per-packet—to identify these phases. Both of these works test their detectors only on simulated dictionary attacks, and do not address how to distinguish instances of forgotten usernames/passwords from brute-forcers. More generally, none of these brute-force detection approaches have the ability to detect stealthy coordinated attacks.

The detection of coordinated attacks has received little treatment in the literature. The earliest work of which we are aware is that of Staniford et al., who cluster anomalous events using simulated annealing for the detection of stealthy port scans [52]. A main limitation of this work is that it first requires identification of individual anomalous events. Gate's work on coordinated scan detection is the most prominent subsequent effort in this domain [30]. Given an input set of scan sources, Gate's algorithm extracts the subset of hosts that appear coordinated by using a set-covering approach; the premise is that the attacker divides the work among the coordinating scanning hosts in a manner that maximizes information gain while minimizing work overlap. This work has two limitations: (i) the individual attack hosts require pointwise identification, and thus the approach will not find stealthy attacks, and (ii) the algorithm lacks a procedure for determining when a site is under attack. Other work has addressed the somewhat similar problem of detecting DDoS attacks, but these detection approaches face a difficult problem of how to differentiate attack participants from legitimate users [55, 51].

Malecot et al. use information visualization techniques to detect distributed SSH brute-force attacks [43]. For each local host, the remote IP addresses that attempt to log in are displayed using a quadtree—a tree data structure formed by recursively subdividing two dimensional space into four quadrants. The procedure performs 16 iterations to map 32-bit IP addresses onto a quadtree, each time deciding the next sub-quadrant by looking at the next two bits of the IP address. The analyst then visually compares quadtrees for different local hosts to identify as coordinated attackers remote IP address(es) that appear in quadtrees of multiple local hosts.

Regarding the prevalence of SSH brute-force attacks, Bezut et al. studied four months of SSH brute-force data collected using three honeypot machines [22]. They find recurring brute-forcing activity, sometimes with several weeks in between, indicating that the attacks target a wide range of IP address space. Owens et al. performed a measurement study of SSH brute-force attacks by analyzing data from honeypots on three networks—a small business network, a residential network, and a university network—for eleven weeks during 2007–2008 [47]. They find that the number of login attempts during different attacks varied from 1 or 2 to thousands. More than a third of the attacks consisted of ten or fewer login attempts. They find instances of both slow and distributed attacks designed to evade detection. They also find that precompiled lists of usernames and passwords are shared across

different attackers, identifying five such dictionaries. Their study reveals that only 11% of the attempted passwords are dictionary words.

2.2 Detection and Measurement of Compromised Credentials

Credential compromise detection is also known as *masquerade detection* in the literature—an attacker possessing the credentials in essence masquerades as the user to whom the account belongs. We can divide the detection approaches in this area into two categories: those that model the attributes associated with the login itself (for example, IP address and the time of login), and those that model post-login account activity (for example, command line activity in case of network-based credentials, and messaging behavior in case of web-based credentials). The former category aims for early detection (at the time of login), while the latter leverages account activity for potentially better detection performance at the cost of delayed detection. Our work falls in the first category and complements detection approaches in the second category. Further, we can categorize the existing work depending on whether they detect compromise of network-login credentials or web-login/VPN credentials.

Network-Login Credentials. The earliest line of work on masquerade detection models user command-line activity using various supervised learning approaches [50, 45, 44, 56]. These efforts primarily uses the *Schonlau* dataset, collected by authors in [50], which contains the command-line activity of 70 users, with each user having a set of 15,000 commands (collected over varying durations of time ranging from several days to several months). In order to craft an evaluation dataset, they select 50 users as *test users* and 20 users as *masquerade users*, and inject command sequences from the masquerade users in the data of each test user. In [50], authors compare six classifiers, and show that the best method achieves a 70% detection rate at a 5% false alarm rate, but the detection rate drops to 40% for a 1% false alarm rate. In [45], the authors evaluate a Naive Bayes classifier on the *Schonlau* dataset, improving the detection rate to 62% for a 1% false alarm rate, and introduce a variation of the evaluation dataset called *1v49* that does not involve injecting masquerade data, but rather treats the data for the other 49 users as positive examples of masquerade. Wang et al. extend this line of work by showing that one-class classifiers, such as one-class Naive Bayes and one-class Support Vector Machines (SVM), can perform on par with the multi-class approaches [56]. A major advantage of these one-class approaches is that they require training data only from the user themselves, and do not require examples of masquerade. In [44], the authors use a richer dataset, collected from set of 168 volunteers at the University of Calgary, that includes command line arguments and flags, and show that this additional information can improve performance (82% detection rate at 6% false positive rate). For these works, it is hard to gauge what the false positive rates translate to in terms of analyst workload in a practical deployment.

Gold et al. also formulate the problem as masquerade detection, but model the properties

of login attempts instead of the command line activity [32]. They develop a probabilistic model to determine if a sequence of logins comes from a two-user model as opposed to that from a single-user model (the original user to whom the account belongs). The single-user likelihood, which is modeled using Naive Bayes, depends on the probabilities that: (i) the user logs in from the given ASN, (ii) the user logs in from a particular IP given that ASN, (iii) the user logs in at the given time of the day and day of the week, (iv) users participate in overlapping sessions from two different locations, and (v) the user has a session with response bytes as extreme as that in the given session. The Expectation Maximization (EM) algorithm estimates the likelihood of a two-user model, and the system flags an account as compromised if the likelihood of the two-user model is higher than that of the single-user model. The data in this work comes from a medium-sized enterprise network with 2,000 users and spans roughly a year. For evaluation, the authors craft a one-week dataset injecting sessions from a set of 100 selected users into the sessions of the remaining users that appear within this test window. At a 1% false positive rate (10 users flagged as compromised per week for a user-base of 1,000 active users), the system achieves a true detection rate of 59%. A higher detection rate ($> 90\%$) comes at a cost of a much higher false alarm rate (on the order of a hundred false alarms per week).

Pecchia et al. studied three years of data collected at the National Center for Supercomputing Applications at the University of Illinois, which runs an instrumented version of SSH and records commands typed by the users in addition to recording the properties of the login attempts [48]. The dataset reflects the activity of ≈5,000 users and contains 20 known compromise incidents over the three years. They train a Bayesian network to detect compromised credentials, and use three kinds of features in the development of their detector: (i) binary indicator features (e.g., unknown IP address and unknown authentication mechanism), which determine whether the user has logged in using those values in the past, (ii) command anomalies identified using a blacklist of suspicious commands, and (iii) intrusion alerts (e.g., suspicious file downloads and contact with blacklisted IP addresses) generated by the site Intrusion Detection System (IDS). Using a conservative threshold on the probability of compromise as 0.7%, the system is able to detect all the 20 known incidents, but the number of potentially compromised users is on the order of a hundred per day. Given the high number of false positives, the authors propose using the framework mainly for aiding an analyst, who can use it to prioritize investigations based on the probability of compromise.

Web-Login/VPN Credentials. Zhang et al. propose UCAAS (University Credential Abuse Auditing System) to detect compromised accounts in university settings [58]. They collected two weeks of web-login/VPN authentication logs, each from the University of Michigan and the University of Illinois at Urbana-Champaign. The accounts in this setting are used to access the university webmail, the online course portal, scholarly articles via library, and to connect to the university VPN. The authors first establish ground truth using incident tickets and a combination of heuristics (temporal-spatial violations, suspicious IP addresses accessing multiple accounts on a given day, accounts exclusively having VPN / library access activity). Using this labeled dataset, they then train a logistic regression

classifier with three different kinds of features: (i) suspicious behavior (for example, logins from multiple countries in a short amount of time), (ii) resource usage features (for example, percentage of VPN versus university website accesses), and (iii) how much does the profile differ from that computed over the past one week—the profile includes timing features, values of properties of IP address (such as ASN, geographic location, and top-level domain), and resource based features. They evaluate this classifier on two weeks of validation data at both the sites, finding a set of 124 compromised accounts and 2 shared credentials at the first site, and 11 compromised accounts at the second site. Surprisingly, validating their system with the site security operators, they find that the system does not result in *any* false positives or missed detections. This could be due to over-fitting, since authors first use the same heuristics to find compromised accounts that they then use as features in their detector. One of the findings of this work is that the university compromised accounts are used to bypass censorship and are also sold on underground markets to access scholarly articles.

In a recent work, Freeman et al. developed a Bayesian framework to score suspicious web login attempts in real time [28]. They use two key ideas to deal with the limitations of data available for training a supervised model: (i) massaging the likelihood decision function to use independently computed reputation scores from external sources instead of prior probabilities from the training data, and (ii) using smoothing techniques to account for (expected) unseen feature values due to the sparsity of data. The system uses only two features (IP address and User-Agent), and assumes them to be independent. They evaluate their system on both simulated attack data and real-world labeled data from LinkedIn, the popular online professional networking platform. They report detection rates at a 10% false positive rate (which they frame as a business decision reflecting the amount of inconvenience to the users that the company is willing to tolerate). At this FPR, they are able to achieve a true positive rate of 77% for the real-world compromised accounts, and a 74% true positive rate for the hardest-to-detect simulated attack reflecting a phishing attacker who is able to spoof the User-Agent and also knows the country that the user logs in from.

Egele et al. proposed COMPA, a system to detect groups of compromised accounts on online social networks [26]. COMPA first identifies groups of messages that are similar in a given observation interval, by looking at both *content similarity* (messages are similar if they share at least one four-gram) and *URL similarity*. Then for each group, COMPA determines if it is potentially suspicious by looking at the fraction of users who deviate from their normal posting behavior. In order to detect a change from a user's normal posting behavior, COMPA uses the following set of features: the language of the post, the topics a user posts about, the domain of the links the user shares, the time of the post, message source, and personal interaction history. The authors evaluate the detector on large-scale datasets from Twitter and Facebook, and build ground truth through manual inspection of URLs in the posts, topic changes, and description pages of the message posting applications. They find hundreds of thousands of compromised Twitter accounts by running COMPA on three months worth of data, and tens of thousands of accounts on two years worth of data from selected groups in Facebook. The authors find that these compromised accounts were being used to launch

large-scale phishing and scam campaigns, and also to spread worms.

Similarly, Thomas et al. discover groups of compromised Twitter accounts by clustering user tweets based on content and URL similarity [53]. However, as opposed to COMPA, which employs deviation of account activity from past behavior to identify clusters representing compromised accounts, the authors of this work use two key observations to label the clusters in retrospect: (*i*) compromised users delete spam tweets upon realizing that their account has been hijacked, and (*ii*) Twitter suspends fraudulent acccounts that send spam or have excessive connections. After manually labeling a sample of clusters using tweet and account statuses, they train a multi-class logistic regression classifier to categorize the complete set of clusters into memes, compromised accounts, and fraudulent accounts. Using this approach, they find on the order of 2,600 clusters representing 13.8M compromised accounts. The authors then study the impact of compromise finding that 21% of the users never return to their accounts post-compromise, and 57% lose connections due to spam originating from their accounts. One of the key findings of this work is that compromise on social networks spreads through phishing and malware campaigns along the social graph, and the more connected a user is the higher the likelihood of their getting compromised.

In summary, when viewed in relation to existing literature, this thesis makes the following contributions. First, we build the first system to detect stealthy and distributed brute-force attacks, and in addition run the detector on eight years worth of data to construct a temporal picture of how these attacks have evolved over time from the viewpoint of an enterprise network with several thousand employees. None of the existing works address the detection of such stealthy, distributed attacks. Second, we build on existing work on detecting compromised credentials, specifically in the context of enterprise settings. We present a practical and deployable system that produces a high-quality feed for analysts, with an easy-to-understand explanation of why a given login is anomalous. In this regard, we improve on the existing work on this detection problem, which suffers the limitation of achieving a reasonable analyst workload while maintaining a good detection rate.

Chapter 3

Datasets

The network monitoring systems in place at the Lawrence Berkeley National Lab (LBNL) provide the primary source of data for this work. LBNL is a large US national research laboratory with several thousand employees, located in Berkeley, California. The site's systems primarily reside in two /16 address blocks (from two separate /8's). Only a small fraction of the address space runs externally accessible SSH servers, providing access to both individual user machines and compute clusters. Due to privacy concerns, LBNL's data sharing agreement allows us access to data only in *anonymized* form. However, the operational security team at the lab greatly facilitated questions that we were unable to resolve using the anonymized data.

We carried out the development and evaluation of the detectors presented in this thesis on this primary dataset. We also used datasets from three other geographical locations / address blocks to enrich and supplement our findings on the primary dataset. Figure 3.1 gives an overview of all the datasets and their temporal-spatial characteristics. We discuss them in detail next.

3.1 Main Dataset (LBNL)

The data from this site spans several years (dating back to 2005). We used an eight-year cut (2005–2012) for the brute-force detection work and a later one-year cut (2015) for the credential theft detector. The difference in temporal cuts is due to two reasons: (*i*) the time at which the work was carried out: the distributed brute-force work was completed in 2013 and the credential theft work in 2016, and (*ii*) the difference in the nature of evaluation for the two problems: for the credential theft work, our assessment of anomalous logins flagged by the detector involves a feedback loop with the users/site operators, which required working with a recent snapshot. In comparison, the evaluation of brute-force attack detection work is largely independent of user feedback, allowing us to fully exploit the temporal breadth of the available data and study the distributed attacks across years.

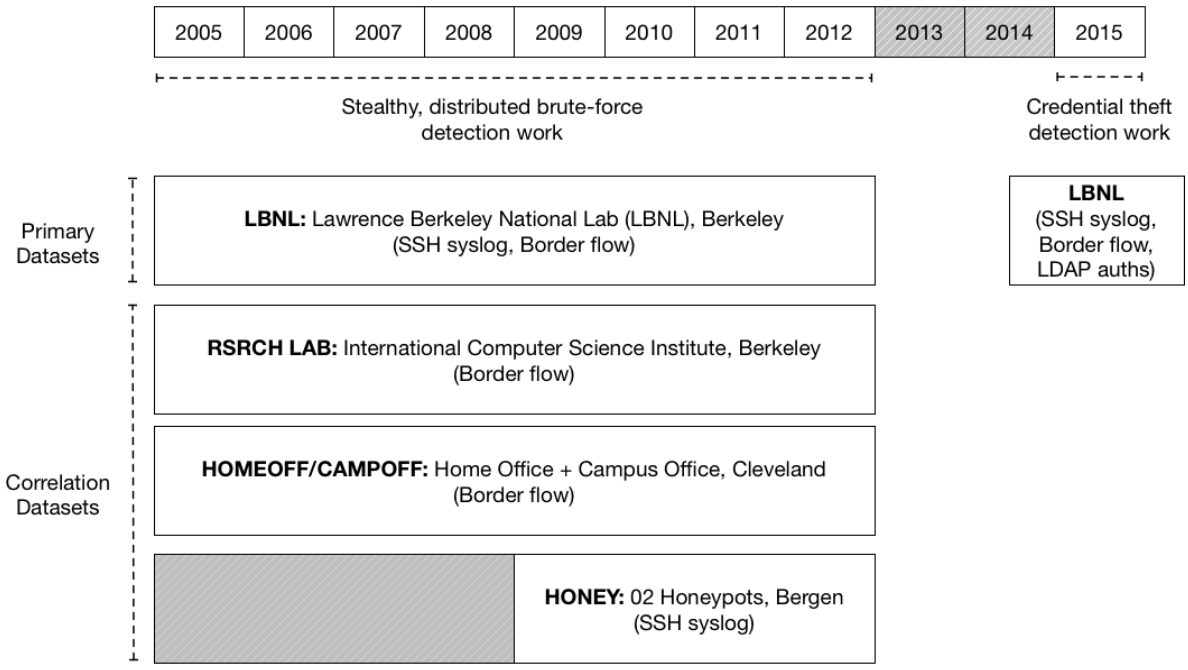


Figure 3.1: Temporal-spatial aspects of the datasets used in this work.

We use three different types of network monitoring logs from the site. In the discussion below, we sketch what data each log source provides. We also discuss how we anonymize the data (in order for the site to share it with us), as well as how we combine data for a given login/user across the different datasets.

3.1.1 SSH Syslog Data

A central syslog server at LBNL records information about login attempts reported by (most of) the SSH servers at the site.[1] This data contains both local login activity between systems at LBNL as well as inbound activity from systems external to the site. We term the former *local* logins and the latter *external* logins. For each login attempt, the logs provide the following information: the time of login, client and server[2] IP addresses, username on the server, whether the login succeeded, and the authentication type used.

A caveat regarding the timestamps in these logs: the central syslogging server records the time of login according to the clock of the SSH server to which the user logs in (as opposed to the time at the central syslogging server). This means that the logs do not reflect correct timestamps for logins to servers that have incorrect clocks. In order to correct the bad timestamps, we observe that the SSH syslog data is dense: for a sample day that we looked

[1] Some machines are not set up to forward syslog to the central syslogging server.

[2] Some of the syslog records log the server's hostname rather than its IP address. For these we correlated the records against the site's DNS and DHCP logs to resolve to IP addresses.

at, 88% of the entries had the same timestamp as that of the preceding entry, and 91% had the same or one second greater. At a ten-second granularity, no 10-second interval had less than a couple dozen entries. This is dense enough for us to correct clocks to a 10-second accuracy by using the surrounding log entries.

We use the following algorithm to correct the bad timestamps: for a given day, we initialize the `CORRECT_TIME` to midnight (the central syslogging server rotates logs every midnight). We then iterate over the log lines, advancing the `CORRECT_TIME` according to that in the log line, if it is within the accuracy window of 10 seconds from the current `CORRECT_TIME`. However, if the timestamp recorded in the log line is behind or greater than the accuracy window, we correct it to the current `CORRECT_TIME`.

```
1  ACCURACY_WINDOW = 10 seconds
2  CORRECT_TIME = midnight timestamp for the log date
3  for line in file do
4      current_timestamp = get_timestamp(line)
5      if current_timestamp < CORRECT_TIME then
6          adjusted_current_timestamp = CORRECT_TIME
7      else if current_timestamp > CORRECT_TIME + ACCURACY_WINDOW then
8          adjusted_current_timestamp = CORRECT_TIME
9      else
10         adjusted_current_timestamp = current_timestamp
11     CORRECT_TIME = adjusted_current_timestamp
```

Algorithm 1: Correcting bad timestamps due to incorrect clocks in syslog data.

3.1.2 Border Flow Data

LBNL monitors all network traffic crossing its border using the `Bro` network security monitor [15]. `Bro` produces summary connection logs for all incoming and outgoing connections (`conn.log`). Among other details, the `conn.log` provides flow-level information on bytes transferred in each direction as well as how long each connection lasted. In addition, `Bro`'s SSH protocol analyzer produces `ssh.log`, which records the client and server version strings extracted from the `SSH Protocol Version Exchange` messages during connection establishment [10]. Both these logs provide additional session characteristics that the syslog data does not record: session duration, bytes transferred, and the SSH client/server versions for connections corresponding to both inbound and outbound SSH logins. (Note that the border monitor lacks visibility for local activity within the site).

In addition to providing session characteristics, the border flow data supplements the partial view of syslog data by providing contact information (but no details on username and success/failure) for external attempts to machines that do not run an SSH server, or that

run an SSH server which does not log via the central syslog server. The logs also provide a similar flow-level visibility into outbound SSH activity, which also is not captured by the syslog data.

Combining syslog and flow data: For *inbound* logins, we have both syslog and flow data, allowing us to combine the two data sources for this set of logins. However, since the two datasets are collected at different vantage points, they log slightly different timestamps for the same login. To find the flow record corresponding to each login attempt, we search the flow data for a connection that has the same `<client IP address, server IP address, source port>` value and occurs within a small time-window of the login time recorded in the syslog data.[3] A small caveat regarding the time-window search for combining the two data sources: LBNL rotates `Bro` logs every day, which means that any ongoing sessions at the time of rotation get recorded in the logs for the day on which they end. This is because `Bro` only logs a connection once it ends, since the attributes it records (e.g., duration, connection state, and total bytes transferred) depend on the complete duration of the connection. We first sorted the logs to move the connections spanning multiple days to logs for their respective start dates, allowing us to perform a narrower time-window search for each login attempt instead of searching for them across multiple days.

Note that we do not have syslog data for outbound SSH activity, nor for servers that do not syslog. For these, we use the unmerged flow records in our analysis.

3.1.3 LDAP Authentication Data

The LDAP authentication server at LBNL logs authentication requests for various organization-specific web services, such as webmail, HR self-service sign-on, and the Lab Employee Time System (LETS). The LDAP logs provide the time-of-login, username, client IP address, and the web service for each *successful* LDAP authentication. (Note that we have this data only for 2015).

Unlike syslog and flow data, which reflect data collected at two different vantage points for `ssh` activity, the LDAP data reflects login activity of users for services other than `ssh`. This data stream can potentially provide a rich view into additional site-wide activity of a user around the time they log in to an `ssh` server on site. This view provides an angle to identify anomalous `ssh` logins by checking for incongruities between `ssh` authentications and other authentications of a user at the site. However, successfully leveraging the LDAP data requires the ability to combine it with the syslog/flow datasets.

Combining syslog and LDAP data: To combine these two datasets, we need to perform a join by users (the same underlying human users). Unfortunately, the identities for majority

[3] We use a search window of 180 seconds in this work. The syslog data does not record the port on which the SSH server is running, though with high likelihood it is port 22. Regardless, the `<client IP address, server IP address, source port>` triplet on a given day is unique enough that it suffices to match the syslog and flow data.

of the users in the two datasets differ, and LBNL does not maintain a mapping between the identities. For the users whose identity is the same in the two datasets we can directly combine their data. For the remaining `ssh` users, we first attempt to identify their corresponding LDAP identity. Towards this end, we extracted the list of account pairs that log in from the same IP address in the two datasets within a given 24-hour time window. We gave this list to the site operators, who made a decision on whether each LDAP/`ssh` account pair belongs to the same user or not by looking for syntactic clues in the account names. For example, a user with an identity *mjaved* in the syslog data might have an LDAP identity of *mobinjaved* — the human eye can tell that these two accounts potentially belong to the same user. We further discuss this matching strategy and the results in §3.3.2.

3.1.4 Data Anonymization

Our agreement with LBNL allows us to access the above-mentioned logs only in anonymized form. To this end, we provided a data preparation and anonymization script to the operational security team at LBNL. The script first parses the syslog data (correcting any incorrect timestamps along the way), then merges the syslog and flow data, and then translates the usernames and IP addresses in the logs to an integer space, keeping the other fields intact. The unmerged flow data as well as the LDAP data is then anonymized using the same anonymization dictionary. To prevent anonymization from limiting our work, LBNL also provided us with the following meta-information for each anonymized IP address: Autonomous System Number (ASN), geo-location (at the granularities of lat/long, city, and country), and whether the IP address belongs to LBNL or not (this helps us in establishing whether the login activity is local or inbound/outbound). The ASN information was obtained through `whois` lookups, and the geo-location information through the `Maxmind` database. Further, for the local IP addresses, LBNL also provided us with labels on whether the IP address is private or public, wired or wireless, and static or DHCP.

3.1.5 Data Logs Summary

After anonymization and merging, the merged syslog-flow data has the following fields:

```
<
      timestamp          : timestamp of login as recorded in syslog
                           (adjusted if server clock is bad)
      anon-client-ip     : anonymized source IP address
      anon-client-port   : anonymized source port
      is-client-ip-local: true if the source IP address belongs to the site,
                           else false,
      anon-server-ip     : anonymized server IP address
      anon-user          : anonymized username
      auth-type          : type of authentication (e.g., password, public-key)
```

```
      is-auth-successful: true if authentication successful, else false
      SSH-client-version: SSH client version string
                          (empty if is-client-ip-local is true)
      SSH-server-version: SSH server version string
                          (empty if is-client-ip-local is true)
      orig-bytes        : bytes transferred in originator direction
                          (empty if is-client-ip-local is true)
      resp-bytes        : bytes tranferred in responder direction
                          (empty if is-client-ip-local is true)
      session-duration  : session duration
                          (empty if is-client-ip-local is true)
>
```

We also retain the unmerged flow data as it gives us a view into connections to site servers for which syslog does not have coverage, as well as a view into outbound activity. The unmerged flow data has the following fields that are of relevance for this work:

```
<
      timestamp         : timestamp for connection start as seen in Bro log
      anon-client-ip    : anonymized source IP address
      anon-client-port  : anonymized source port
      anon-server-ip    : anonymized server IP address
      anon-server-port  : anonymized server port
      orig-bytes        : bytes transferred in originator direction
      resp-bytes        : bytes tranferred in responder direction
      session-duration  : session duration
      direction         : inbound/outbound
>
```

The LDAP data is fairly simple with only the following four fields:

```
<
      timestamp         : timestamp for LDAP authentication
      service           : service for LDAP authentication
      anon-client-ip    : anonymized source IP address
      anon-ldap-user    : anonymized LDAP username
>
```

Note that we use the same anonymization dictionary across all three datasets in order to ensure consistent anonymization of IP addresses/usernames that appear across multiple datasets. For all the IP addresses that appear across the datasets, we also have the following meta-data:

```
<
      client-ip-ASN        : Autonomous System Number/Name of source IP address,
      client-ip-city       : city of source IP address
      client-ip-country    : country of source IP address
      client-ip-lat-long   : lat/long of source IP address
      is-client-ip-local   : true if the source IP address belongs to site,
                             else false
      is-client-ip-wireless: true if client-ip belongs to the site and is
                             allocated to wireless network,
                             false if client-ip belongs to the site and is
                             allocated to wired network,
                             empty if client-ip is external to the site
      is-client-ip-airport : true if client-ip belongs to an airport/in-flight
                             address block, else false
      is-client-ip-private : true if client-ip is in one of the private
                             address blocks, else false
      is-client-ip-static  : true if client-ip is static, false if dhcp
>
```

3.2 Correlation Datasets

We also draw upon SSH datasets from four other sites spread across IP address space and geographic locations. Due to limitations of these datasets in terms of the breadth of information they provide (either only flow data or only syslog data from a single server instead of a network), we are unable to leverage them for the development and evaluation of our detectors. However, we can confirm whether the attacks we find in the main LBNL dataset occur in these datasets too, thereby allowing us to leverage them as *"correlation datasets"*.

We refer to the four correlation datasets as Honey, RsrchLab, HomeOff, and CampOff, and discuss them below. In the description below, the term *syslog data* reflects the same data fields as we sketched in the last section for LBNL, but *flow data* refers to a reduced version containing only the following four fields: `<timestamp, client-ip, client-port, server-ip>`. These fields suffice for our correlation analysis for the distributed brute-force detection work.

The Honey dataset provides SSH syslog data captured by two honeypot servers in Bergen, Norway during 2009–2012 [16]. The RsrchLab dataset reflects flow data (2005–2012) from the International Computer Science Institute, a research organization in Berkeley, California, with a /23 address block. HomeOff and CampOff capture inbound SSH flow data for a home office (HomeOff) and a campus office (CampOff), both in Cleveland, OH (but in separate netblocks). The data likewise spans 2005–2012.

Time span	Jan 2005–Dec 2012
SSH servers	2,243
Valid users	4,364
Distinct valid user/server pairs	10,809
Login attempts (externally-inititated)	12,917,223
Login successes	8,935,298
Remote clients	154,318
Attempts using passwords	5,354,833
successes	1,416,590
remote clients	119,826
SSH border flows	215,244,481
remote clients seen in flows	140,164
High-rate brute-forcers	7,476
Mean attempts per high-rate brute-forcer	382.84
Mean daily password login attempts	486.13 ($\sigma = 182.95$)
Mean daily users	116.44 ($\sigma = 32.41$)

Table 3.1: Summary of LBNL syslog and flow data (2005-2012).

Note that we leverage these correlation datasets only for the distributed brute-force detection work.

3.3 Data Characterization and Filtering

The last two sections familiarized the reader with the nature of the datasets and their macro aspects, i.e., the temporal and spatial breadth. In this section, we dig deeper into the datasets, characterizing various additional aspects (such as, the number of servers, valid users, and login attempts in the dataset), as well as finding opportunities to filter the raw datasets down to a form on which we start building the detectors. Because the two problems we tackle in this work differ in nature, they require different filtering criteria. In particular, for the credential theft work, we focus on finding *successful* anomalous logins, so we can filter out the failed login attempts. Similarly, for the distributed brute-force detection work, it makes sense to focus only on *inbound* login attempts corresponding to authentication types that are vulnerable to brute-forcing.

We organize this section by the two subproblems of this dissertation, and discuss each in turn below.

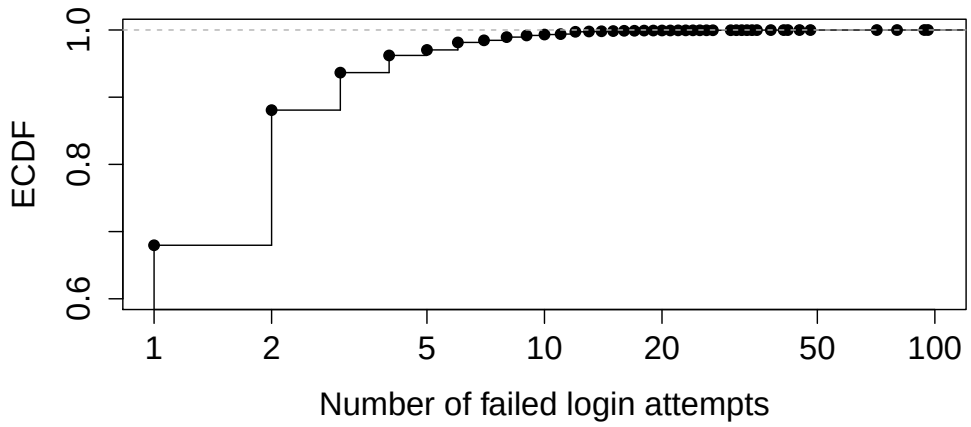


Figure 3.2: Empirical CDF of the number of failed login attempts per hour until a success for legitimate user login efforts with forgotten or mistyped usernames/passwords.

3.3.1 Dataset Slicing & Characterization for Brute-force Attacks Detection

Table 3.1 provides summary statistics for the 2005–2012 cut of the main LBNL dataset. Over the eight years, the central syslog server recorded ≈12M externally initiated login attempts for ≈2.2K servers and ≈4.3K valid users. The benign SSH activity in this data consists of interactive as well as scripted logins.

Filtering local activity and non-password authentication attempts. For this data, we work with externally-initiated logins, and further the subset of SSH authentication types vulnerable to brute-forcing (i.e., we omit those using public key authentication), about half of the attempts. We perform all of the characterizations and analyses in the remainder of this work in terms of this subset.

Filtering high-rate brute-forcers. In addition, we filter this dataset to remove individual brute-forcers that we can readily detect using a per-host threshold for the number of failed login attempts per remote host within a window of time. Given the ease of detecting these brute-forcers, they do not reflect an interesting problem for our detector, though they would heavily dominate the data by sheer volume if we kept them as part of our analysis.

To identify and remove such brute-forcers, we need to empirically establish reasonable thresholds for such a per-host detector. We do so by analyzing the process by which legitimate users make password authentication failures, as follows. We assume that any user who makes repeated failed login attempts followed by a successful attempt reflects a legitimate user. (This assumption may allow consideration of a few successful SSH brute-forcers as "legitimate", but these are very rare and thus will not skew the results.)

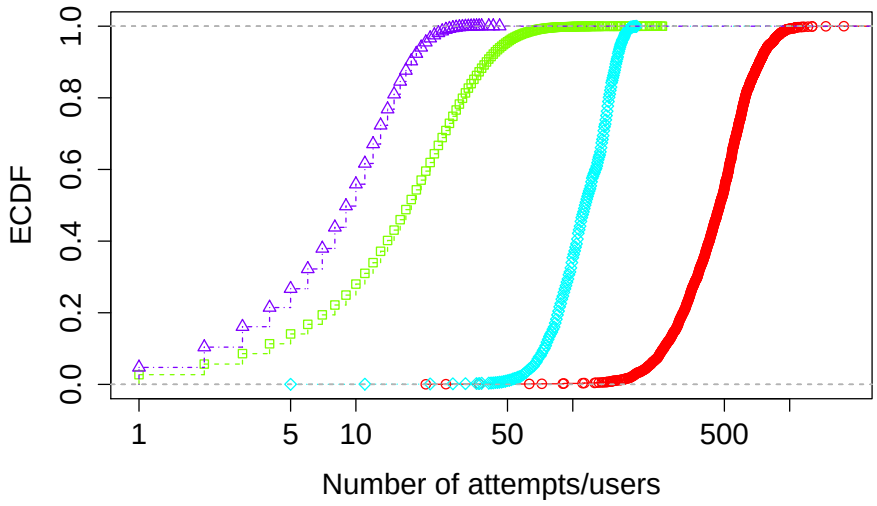


Figure 3.3: Empirical CDFs for benign password-based SSH usage in LBNL data. Left to right: (i) valid users per hour, (ii) successful logins per hour, (iii) valid users per day, (iv) successful attempts per day.

Figure 3.2 plots the number of failed attempts such users make prior to finally succeeding. We see that instances exceeding 10 failed attempts are quite rare, but do happen occasionally. Accordingly, we consider 20 failed attempts as constituting a conservative threshold. We manually analyzed the instances of legitimate users falling after this cutoff (the upper right tail in the figure) and found they all reflect apparent misconfigurations where the user evidently set up automation but misconfigured the associated password. Thus, we deem any client exhibiting 20 or more failures logging into a single server (with no success) over a one-hour period as a *high-rate brute-forcer*, and remove the client's entire activity from our dataset. Table 3.1 summarizes the brute-forcers removed using this definition.

Finally, to give a sense of the volume of activity that remains after these filtering steps, Figure 3.3 shows the empirical CDF of the hourly and daily numbers of successful logins. A typical day sees 500 successful logins (maximum seen: 1,200) involving 117 distinct users (maximum: 197).

We next briefly characterize the correlation datasets:

- The HONEY dataset reflects five manually identified SSH brute-forcing "campaigns" (our term for ongoing instances of attack, as we discuss later) as captured by two SSH honeypot servers in Norway [16]. Table 3.2 summarizes these campaigns, which for the most part we characterize as large-scale, persistent, and stealthy. For all but the last campaign, many of the remote clients would evade detection by the simple per-host detector that we discussed earlier in this section.

Attack Episode	Days	Remote clients	Login attempts	Avg. attempts per remote client
Oct 2009–Jan 2010	78	4,158	44,513	10 (σ=24)
Jun 2010–Aug 2010	56	5,568	23,009	4 (σ=7)
Oct 2011	6	338	4,773	14 (σ=16)
Nov 2011	13	252	4,903	20 (σ=24)
Apr 2012	6	23	4,757	206 (σ=760)

Table 3.2: Summary of attacks in the HONEY data.

Time span	Jan 2015–Dec 2015
Login attempts	46,059,215 (15% remote)
Login successes	37,105,930 (4% remote)
Valid users	2,491
SSH servers	2,192
Distinct valid user/server pairs	11,504
Total clients (local + external)	27,670
External	16,242
Local	11,378
Users with at least one external login	1,822
Users with all external logins	489

Table 3.3: Summary of LBNL syslog data (2015).

- The RSRCHLAB dataset reflects flow data from the International Computer Science Institute, a research organization in Berkeley, CA, with a /23 address block. The dataset spans the same time as that of LBNL, though due to the limitations of flow data, we cannot establish how many coordinated attacks exist in it. We discuss our correlation strategy for this dataset in §4.3.
- HOMEOFF and CAMPOFF capture inbound SSH flow data for a home office (HOMEOFF) and a campus office (CAMPOFF), both in Cleveland, OH (but in separate netblocks). The data likewise spans Jan 2005–Dec 2012, and again due to its limitation to flow data, we cannot identify coordinated attacks in these datasets.

3.3.2 Dataset Slicing & Characterization for Credential Theft Detection

Table 3.3 summarizes the 2015 temporal-cut of the LBNL *syslog* data that we use for the credential theft detection work. Note that as opposed to the syslog data in the brute-force detection work, which we pre-filtered to retain only externally initiated attempts, this dataset

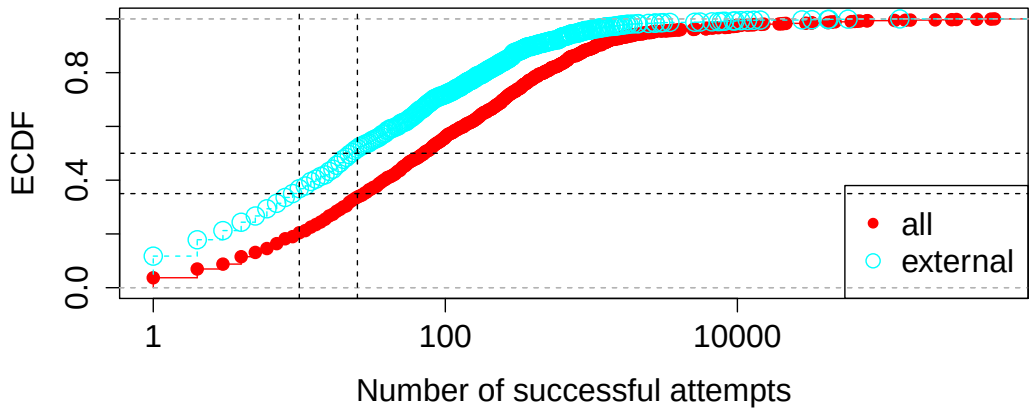


Figure 3.4: Empirical CDF of the number of logins per user during the first six months, filtered on the users who have at least one external login during the complete year.

also contains local activity (lab-to-lab logins).[4] This is because our approach for detecting anomalous *successful* logins utilizes a complete view of the login activity of a given user. The raw dataset contains 46M login attempts over the span of one year. Local activity dominates, with external logins accounting only 15% of the total attempts.

Filtering unsuccessful login attempts. Since we primarily model successful logins for credential theft detection, we filtered out all the unsuccessful attempts (regardless of whether they are from brute-forcers or from the user themselves). This filtering reduces the syslog data to 37M successful logins, reflecting the login activity of ≈2.5K users across ≈2.2K servers. The proportion of external logins in the filtered dataset is 4%, a significant reduction from the 15% in the raw dataset.

Characterizing remote-login accounts. Next we characterize the user accounts along the lines of: (*i*) their external activity in the dataset, and (*ii*) whether the account is generic (such as *root*) or reflects an account belonging to an individual user.

External activity of accounts: Our threat model in this work is that of an attacker who logs in from a machine external to the site. This implies that we only score externally-initiated attempts for suspiciousness and assume that the logins that originate from the site itself are benign (we do leverage local logins in making a decision on the external attempts). This means that for users who only have local activity in the entire dataset, we do not have an opportunity to detect a compromise. Further, the local activity of an account is not sufficient to train our detector; some of the history-based features that we develop in this work can only

[4]This explains the difference in the volume of login attempts in the two datasets: 46M over one year versus 12M over seven years.

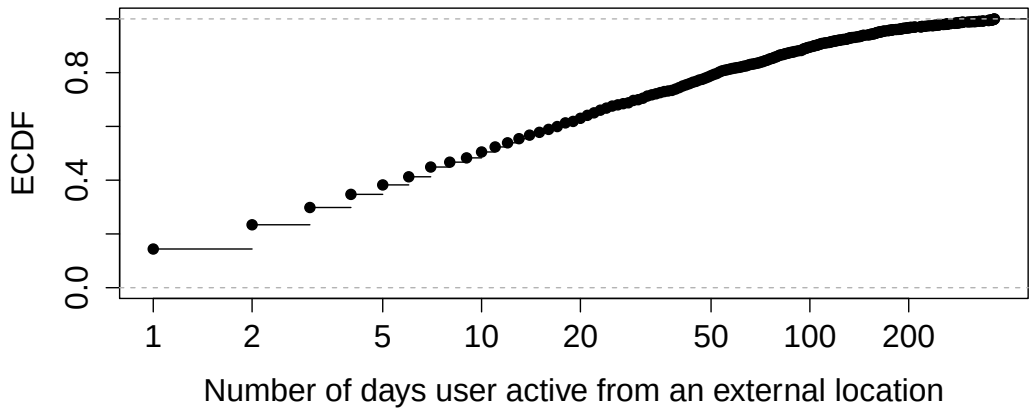


Figure 3.5: Empirical CDF of the number of days a user is active from a location external to the lab.

be bootstrapped using external logins. For example, we have SSH client version information only for external logins (this is due to limitations of our dataset; as we sketched in §3.1.2, the SSH client version is extracted from border monitor logs). Similarly, leveraging IP address history/reputation for detection requires bootstrapping using external logins.

Figure 3.4 shows the distribution of total and external logins during first six months for the users who have at least one external login during the complete dataset. 50% of the users have 25 or fewer external logins during the first six months. (This proportion is lower (38%) if we consider users who have less than 25 total logins instead). Since a user might perform multiple logins in the same sitting in a short amount of time, instead of using the *number of external logins* as a measure of external activity, we also look at the *number of days active from an external location*. We observe at least one successful external login from 1,822 users (73% of the user-base). Of these, ≈50% of the users are active on ten or fewer days during the year (Figure 3.5). The sparsity of external activity for a large fraction of users argues that the detection approach we develop should be able to perform well with only a few days of per-user training. We circle back to this discussion in §5.2.

We note that a subset of users (489) only log in from external locations, and have no local activity throughout the year. This is presumably because they are based in collaborator institutions. Note that LBNL has a number of users at the University of California, Berkeley and the National Energy Research Scientific Computing Center (NERSC), both located in Berkeley but having a separate network from LBNL. Similarly, LBNL also collaborates with foreign institutions (for example, CERN in Switzerland).

Generic/Service accounts: In this work we develop *per-user* models, which are only meaningful

for accounts for whom the login activity corresponds to an individual human user. Generic account names such as *admin, backup, git, guest, and root* are examples of accounts that either might be operated by multiple users or may correspond to multiple underlying accounts. Therefore, we need to identify these accounts and remove them from our analysis. This would be relatively straight-forward if LBNL maintained such classification labels for its user-base. However, from our conversations with the site operators we learnt that they are familiar with only a subset of the `ssh` user accounts (and it is often not clear just by looking at the unanonymized account name whether the account might be shared or not). The main reason is that the account creation at LBNL is not managed in a centralized fashion by the site operators. Instead, several research groups manage their own servers and admins in these groups may create new accounts to grant access to new users/collaborators.

However, it is possible to identify some of these accounts by looking for outliers in per-account aggregate statistics. In particular, we looked for outliers in the number of logins and the number of clients and servers the user logs to/from in the entire dataset, and confirmed these with the site operators. As an example, the account *root* shows up as the most busy user (the user with the highest number of login attempts) and has the highest number of clients/servers in our dataset. We also found maintenance accounts and shared accounts used by specific research groups for cluster job submissions/data fetching. In general, we found that accounts that log in from a number of remote clients but only to a few destinations were either maintenance accounts (such as backup) or multi-user accounts used by research groups for job submissions/data fetching, whereas accounts that log in from a number of remote clients and to a number of servers are generally generic accounts (such as root, guest, admin). However, we found that a number of sysadmin and security group accounts have similar characteristics. As a result of this analysis, we removed a total of nine generic accounts from our analysis.

Tying LDAP and SSH identities. Only 734 out of the 2,491 `ssh` users (29%) have the same identity in the LDAP dataset. For the remaining, we infer *potential* equivalence of identities if we observe both `ssh` and LDAP activity from the same IP address within a 24-hour time window. We identified 578 such LDAP users that had a corresponding `ssh` login within the 24-hour window. The site operators manually investigated this set, and confirmed mappings for 100 out of the 578 cases based on the similarity in usernames. After this addition, we have the corresponding LDAP identities for a total of 834 (34%) of `ssh` users.

Chapter 4

Detection of Stealthy, Distributed Brute-force Attacks

The threat of SSH brute-forcing is well-known: indeed, any SSH server open to general Internet access receives incessant probing by hostile remote systems that energetically attempt to locate instances of weak authentication [18]. The degree to which such attempts also occur in a stealthy slow-but-steady fashion, however, has attracted little study. The difference between single energetic probes and stealthy distributed ones is significant: defenders can easily detect the former, and therefore either block the activity or investigate it (to ensure none of the attempts succeeded). The latter, however, poses a much more difficult detection problem. If each host in a distributed brute-forcing attack itself only attempts username/password logins at a low rate, then distinguishing hostile activity from the inevitable login failures made by legitimate user errors becomes much more difficult. Yet the distinction is vital: a pattern of *attempt*/*attempt*/*attempt*/*success* made by a legitimate user simply reflects a set of typos, or a password that took a few stabs to remember; but by a distributed SSH brute-forcer, it provides the only slender indication of success amongst a mass of probing that in aggregate predominantly failed.

In this chapter we present a general strategy for potentially detecting such stealthy activity, which consists of two basic steps. First, we employ the statistical technique of *change-point detection* to identify times during which a global property has shifted—indicating that, *in aggregate*, a site's activity reflects the presence of problematic activity. We then determine the range of time over which this activity occurred and, within that interval, identify which sources appear to have contributed to the activity.

This chapter proceeds as follows: §4.1 frames our detection approach. In §4.2 we develop a model of the process by which legitimate users make authentication errors when attempting to log in, which serves as the basis for parameterizing our detector. §4.3 presents our evaluation results and findings.

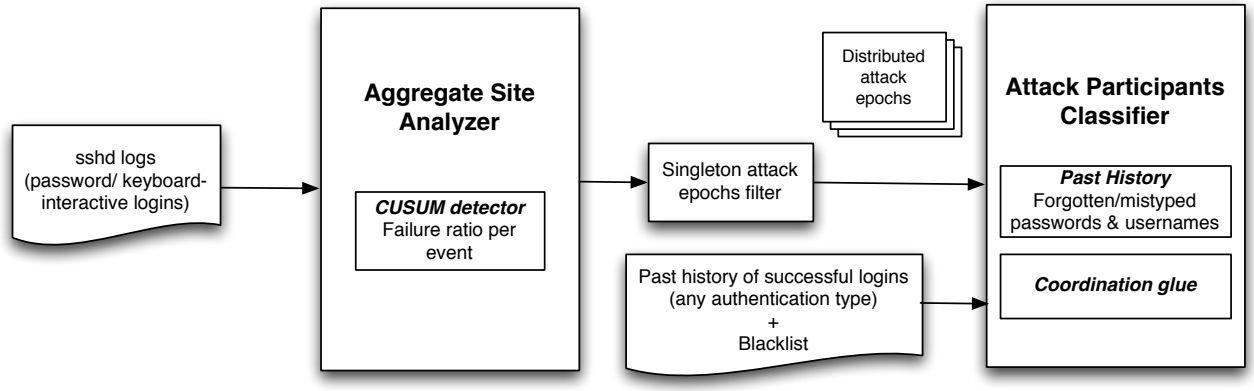


Figure 4.1: System diagram of our distributed SSH brute-forcing detector

4.1 Detector Design

We structure our detection approach as the sequential application of two components. In general terms, the *Aggregate Site Analyzer* first monitors the site's activity to detect *when* an attack of some sort occurs. Upon detection, the *Attack Participants Classifier* analyzes the activity to identify *who* participated in the attack (which remote systems). In this section we develop both the general detection framework and our particular implementation of it for detecting and analyzing distributed SSH brute-forcing attacks. Figure 4.1 presents a system diagram for this latter specific context.

4.1.1 Aggregate Site Analyzer

This component detects the general presence of a coordinated, distributed attack based on complete view of a site's activity. We based the approach on devising a *site-wide* parameter that aggregates information from individual users/machines/events at the site. The *Aggregate Site Analyzer* monitors the probability distribution of this parameter for excessive change and flags the deviations as attacks. Often a single party can by itself induce significant change in the parameter. In this case (which arises for detecting SSH brute-forcing) we need to employ a filtering step to omit *singleton* attacks from the alarms, if our overall goal is focused on detecting distributed activity.

One important design question concerns the *accumulation granularity* of the site-wide parameter, i.e., over what sort of collection of activity do we compute it. For example, if the parameter relates to arrival rates, then windows of time will often be natural choices. In other scenarios, we might achieve better results by defining a normalized parameter amenable to longitudinal comparison. To illustrate, consider the scenario where the two consecutive time windows have 50 and 4 attempts, respectively. If these also have 25 and 2 problematic

attempts, then we would equate them as having equivalent failure rates; but if we expect failures to appear somewhat rarely, we would like to treat the first instance as much more striking than the second, as the latter only needs some modest degree of stochastic fluctuation to occur. Thus, comparing *ratios* across time can prove misleading.

Given that, for detectors that monitor failure ratios we can benefit from a different accumulation granularity: if we compute the site-wide parameter in terms of *events*—defined as the occurrence of n attempts—then we needn't worry about the effects of stochastic fluctuation that can manifest when using windows of time. In addition, such time-variant events have the property that they allow for faster detection of high-rate attacks, as the detector does not need to wait a fixed amount of time before detecting the attack.

In the case of detecting distributed SSH brute-force attacks, we can define an apt *site-wide* parameter, Global Failure Indicator (GFI) as:

$$\text{GFI} = \text{Number of failed login attempts per event}$$

where an event occurs every n login attempts to the site. These n attempts represent a collection of users authenticating to the site, where the number of users will generally vary across events. (Note, while we could normalize GFI by dividing by n, we leave it as a count rather than a ratio, since our subsequent modeling benefits from using discrete variables.) We show in §4.2 that in the absence of an attack (i.e., when failures only occur due to mistakes by legitimate users), this distribution is well-modeled as beta-binomial.

Brute-force activity perturbs the GFI distribution, shifting its mean to a higher value. We use sequential change-point detection to detect significant increases in the mean GFI. In comparison to threshold-based detection, sequential change-point schemes can detect small incremental effects that cumulatively lead to an eventual change of mean. This property enables the detector to detect stealthy attacks. Specifically, we use a Cumulative Sum (CUSUM) change-detection algorithm, as prior work has shown its sensitivity to small shifts in the mean [35].

CUSUM Algorithm. CUSUM models significant changes as shifts in the mean of a random variable from negative to positive. To use it requires transforming an original random variable Y to an associated value Z that has a negative mean under normal operation. One achieves this by subtracting the empirical mean μ of Y plus a small reference value k, i.e., $Z_n = Y_n - \mu - k$. To do so we need to compute μ based on data that describes normal operation (no attack underway); see §4.2 for how we identify such activity. Finally, note that with little modification to the general framework we can for convenience select k so that Z_n is integer-valued rather than real-valued. We do so for our subsequent development of the detector.

We then accumulate Z_n over time using the following (again discrete) test statistic: $S_n = \max(0, S_{n-1} + Z_n)$, where $S_0 = 0$. In the case of no change, the value of S_n hovers around zero, but in the face of a change (increase), S_n starts to accumulate in the positive direction.

By convention, one terms the situation of the mean corresponding to normality as *in-control*. When the mean shifts by an amount $\Delta\mu$, one terms the situation *out-of-control*, which corresponds to an attack in our problem domain. Note that the choice of $\Delta\mu$ is specific to the problem we design the detector to detect. In some situations, we might desire a small $\Delta\mu$, while in others we might only have interest in detecting changes corresponding to a large value of $\Delta\mu$. In practical terms, we achieve a given target $\Delta\mu$ by setting two detector parameters, k and H, as discussed below.

The algorithm flags an *out-of-control* situation when S_n crosses an operator-set threshold, H. The subsequent appearance of an event with normal mean marks the return of the situation to *in-control*, and we reset the test statistic S_n to zero at this point. Thus, the CUSUM detector decides whether the mean has shifted or not according to the decision function D_n:

$$D_n = \begin{cases} 1, & \text{if } S_n > S_{n-1} \text{ and } S_n > H \\ 0, & \text{otherwise.} \end{cases}$$

Determining CUSUM parameters and span of change. One tunes the parameters k and H of CUSUM based on: the amount of change $\Delta\mu$ to detect, the desired false alarm rate, and the desired time to detection. First, a general rule of thumb when designing a CUSUM detector to detect a mean shift of $\Delta\mu$ is to set k equal to half the change in shift [35]. The other parameter, H, controls both the false alarm rate and the detection speed. A lower H means faster detection but a higher false alarm rate.

To assess the balance between these two, we consider the effects of H on the average number of steps the CUSUM detector takes to raise an alarm under in-control and out-of-control distributions. (Note that the first of these corresponds to alarms reflecting false positives, while the latter corresponds to true positives.) We refer to these as *in-control ARL* (Average Run Length) and *out-of-control ARL*, respectively, and choose the value of H that results in the closest match with the desired ARLs.

To determine these ARLs, we can model the CUSUM process as a Markov chain with finite states $X_0, X_1, \ldots, X_H$, corresponding to the test statistic values $S_n \leq 0, S_n = 1, S_n = 2, \ldots, S_n \geq H$ respectively. (Recall that we constrain Z and thus S to discrete integer values.) Note that X_H is the absorbing state. The transition probabilities of this Markov chain depend only on the underlying distribution of the random variable Z:

$$\begin{aligned} P[X_i \to X_0] &= P[Z \leq -i] \\ P[X_i \to X_j] &= P[Z = j - i] \\ P[X_i \to X_H] &= P[Z \geq H - i] \end{aligned}$$

For the intuition behind this formulation, consider the first equation. If the cumulative sum has reached i (i.e., $S_n = i$, corresponding to the state X_i) then the possible ways for progressing from it to the state X_0 (i.e., $S_n \leq 0$) are to add a value of Z less than or equal to

$-i$. A similar perspective holds for the other two equations. Given the complete transition probability matrix $\mathbf{R}$ of the Markov chain, we can compute the above probabilities and the *in-control ARL* as:

$$\textit{in-control ARL} = (\mathbf{I} - \mathbf{R})^{-1}\mathbf{1}$$

where $\mathbf{R}$ is the transition probability matrix, $\mathbf{I}$ is the $(H+1) \times (H+1)$ identity matrix, and $\mathbf{1}$ the $(H+1) \times 1$ matrix of ones [23].

We can likewise compute the *out-of-control ARL* of the detector using the same formulation but substituting $k' = k - \Delta\mu$ [35]. We can then estimate the point of the true start of a change by subtracting the value of *out-of-control ARL* (detection delay) from the time of detection.

Finally, the *Aggregate Site Analyzer* reports the information from CUSUM in the form of *attack epochs*. An attack epoch constitutes of: (i) the set of consecutive *out-of-control* events (i.e., $i = 1 \ldots n$ where $D_i = 1$), and (ii) the set of previous events also incorporated into the epoch based on stepping back through the number of events given by the *out-of-control ARL*.

Each attack epoch can reflect instances of either *singleton* or *coordinated* attacks. The first of these corresponds to a global perturbation of the site-wide variable Y induced by a single source. The second refers to the perturbation arising due to the combined action of multiple sources. Since in this work we focus on distributed attack epochs, we need at this point to exclude singleton attacks.[1] We do so by checking whether CUSUM still flags any events in the epoch as reflecting an attack even if we remove the remote host with the highest number of failed login attempts. If so, we mark the attack epoch as a coordinated attack epoch, and proceed to the second component of our analysis. Otherwise, we discard the epoch as uninteresting (which occurred about 3/4s of the time).

4.1.2 Attack Participants Classifier

The second component of our general detection approach addresses how to go from the global site-wide view to that of individual entities. Here we employ a set of heuristics to analyze activity in the attack epochs flagged by the *Aggregate Site Analyzer* to identify *who* participated in the attack. (The need for heuristics rather than more principled identification arises almost fundamentally from the problem domain: if we could directly identify participants with confidence, we could very likely use the same approach to develop an effective pointwise detector and not have to employ a separate approach for detecting stealthy distributed activity in the first place.)

For our particular problem of detecting distributed SSH brute-force attacks, the individual entities we wish to identify are remote hosts (clients). In addition to the problem of

[1] Note that such single sources can arise even though we previously filtered out high-rate brute-forcers (per §3.3.1) because these singletons might spread their activity across multiple servers, or probe at a rate lower than the 20 failures/hour threshold.

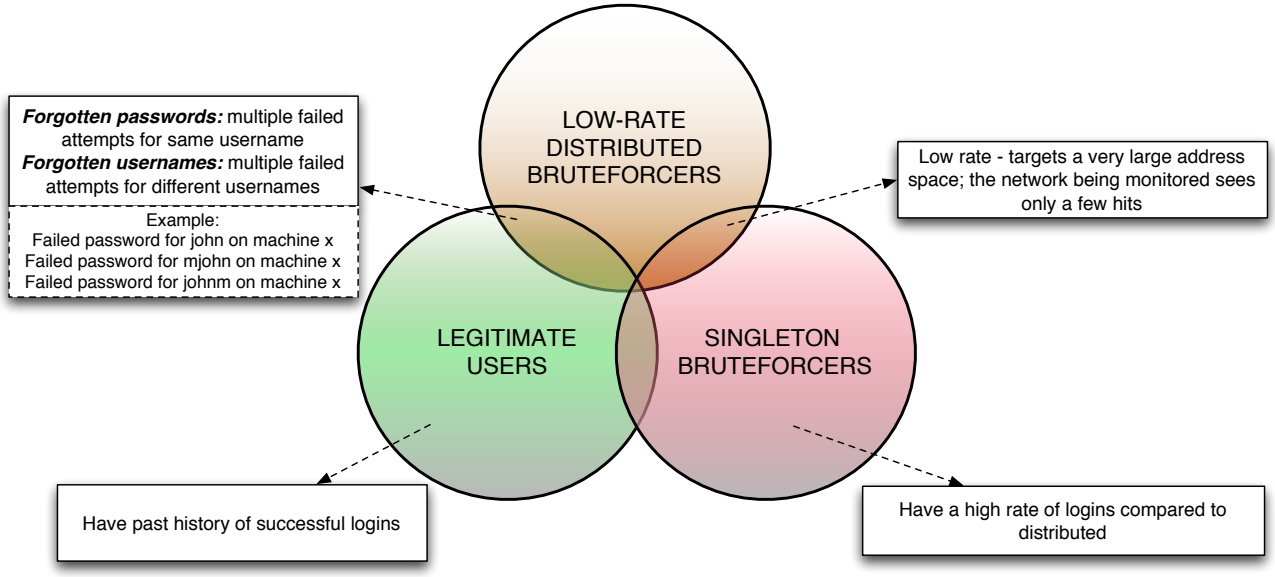


Figure 4.2: Possible characteristics of remote hosts that fail.

including remote hosts corresponding to legitimate users within it, a distributed attack epoch—particularly if spanning a long period of time—can capture multiple brute-forcers, some of whom might operate in a coordinated fashion, while others might function independently. For example, an attack epoch we detect that includes activity from five remote hosts might in fact be composed of four coordinating remote hosts and one singleton brute-forcer that happens to probe the site at the same time.

For each remote host that appears during the attack epoch, we make a decision about whether to classify it as a legitimate remote host, a singleton brute-forcer (operating alone), or a brute-forcer working with other hosts as part of a coordinated attack. This decision might require manual analysis, as sometimes the categories have significant overlap. To illustrate, Figure 4.2 diagrams the characteristics that remote hosts in each of these categories can manifest. Legitimate users that fail due to forgotten or mistyped usernames/passwords generally exhibit only a modest number of attempts, similar to low-rate distributed brute-forcers. A remote client with no past history of successful logins (see below) provides us with little indication as to whether it reflects a legitimate user or a distributed brute-forcer. Likewise, while we can readily identify singleton brute-forcers that probe at rates higher than distributed brute-forcers, ones that probe at low rates fall into a grey area that we find difficult to automatically classify.

Our classification procedure has two parts. First, we make a set of decisions based on past history. Second, for the remaining hosts during an attack epoch we assess the degree to which they evince the same *coordination glue*: that is, commonality in the set of servers and/or usernames that the hosts probe. The premise behind this second step comes from assuming that attack hosts in a distributed attack aim to work together to achieve a particular task:

attackers achieve little utility if their multiple attack hosts do not effectively focus their work. We might also expect that coordinated attack hosts probe at similar levels (number of attempts) due to use of common automation software.

Classifying activity based on past history. To facilitate our discussion of this step, we use the following notation: L refers to a *Local host*; R to a *Remote host*; and U to a *Username*. Given that, we classify activity by analyzing past history as follows:

(1) *Forgotten/mistyped passwords:* we identify ⟨R, U⟩ pairs that have authenticated successfully in the past to any local machine at the site, and consider such activity benign, removing it from the attack epoch. (We filter ⟨R, U⟩ pairs instead of only remote hosts because multiple users might be behind a NAT.) We can safely filter out this set because the remote client has already established its ability to successfully authenticate as the given user, and thus has no need to try to guess that user's password. While the current activity is not necessarily benign (due to the possibility of a malicious login using stolen/compromised credentials), that scenario lies outside the scope of what we try to detect here.

(2) *Forgotten/mistyped usernames:* the previous step of filtering ⟨R, U⟩ pairs will miss instances of benign failures when the user mistypes or fails to recall their username. To identify such failures, for each remote host we determine whether it produced a successful login in the past to *any* internal host using a username *closely related* (edit distance = 1) to a username present in the event. If so, we again mark the ⟨R, U⟩ pair as benign. (Note that we skip this step if we have previously identified R as a singleton brute-forcer.)

Identifying coordination glue. After filtering out some activity on the basis of past history, we then turn to making decisions about attack participation amongst the remaining activity on the basis of evidence of "coordination glue", as discussed above. We expect to find either a common set-of-local-servers or set-of-usernames as the coordination glue in most of attacks.

We identify such glue using an approach based on bi-clustering. We construct a bipartite graph with the remote clients as the first node set A and either usernames or local servers as the second node set B. We create an edge between remote client r in node set A and a username u (local server l) in node set B if r made a login attempt as u (to l). For each graph we then look for partitions of the graph that maximize the edges within the partitions and exhibit very few edges across partitions. We mark nodes belonging to very small partitions as either legitimate users or coincidental singleton brute-forcers, and remove them.

Presently we proceed with the above as a manual process, which we find tractable since the number of attack epochs that emerge from our various filtering steps is quite manageable. In addition, we observe that a distributed attack that targets many hosts on the Internet might lead to activity at the site that exhibits little in the way of apparent coordination glue. In these scenarios, we also take into account timing patterns, number of attempts per remote host, and the presence of an alphabetical progression of usernames as alternate forms of coordination glue.

4.2 Modeling User Authentication Failures

To apply the approach developed in the previous section, we need to model the process by which legitimate users make authentication errors when attempting to log in. We want to achieve this modeling not simply in a highly aggregated fashion (e.g., by determining a global benign-user failure rate), but distributionally, as the latter will allow us to form well-grounded estimates of the expected false positives and time-to-detection behavior of our detection procedure. In particular, capturing the distribution will allow us to model GFI (number of failures per event of n login attempts), and thus to determine the Markov chain probabilities required to compute average-run-lengths.

In order to extract a characterization of legitimate login failures from the LBNL syslog data, we need to first identify clients within it that do *not* reflect legitimate users. Figure 4.3 shows a heat map of the number of login attempts vs. failure ratio per remote host computed over the complete data for password-based authentication (except with high-rate brute-forcers already filtered out, per §3.3.1). The major density of remote clients resides in the lower left and middle areas of the plot; these mainly depict benign activity. The top region of the plot is highly dominated by brute-forcers (those that went slowly or broadly enough to survive the high-rate filtering), with a few legitimate users in the top-left corner, and some possible misconfigurations throughout. The mid-right region, with a high number of login attempts and a failure ratio in the range 0.4–0.6, very likely consists of misconfigured automations. Finally, the lower-right region captures well-configured automations; these use scripts that log in repeatedly with the correct password, and thus no chance of failure, except for intervening interactive access.

We now discuss our techniques to clean the data to remove both brute-forcers and automations (both well-configured and misconfigured) in the data. After doing so, we illustrate a distribution that fits the cleaned data quite well, providing us with the means to then model GFI.

4.2.1 Removing Brute-forcers

Accurately identifying low-rate brute-forcers poses a circular problem, as that is exactly what we ultimately set out to detect in this work. Instead we develop an *approximate* procedure to remove the brute-forcers, as follows. We remove from the dataset all remote hosts that never achieve a successful login attempt. The chance that this removes legitimate hosts is low, because it should be rare that a legitimate user repeatedly attempts to log in and never succeeds.

This heuristic removes all of the brute-forcers except the successful ones. In order to cull them, we remove remote hosts with failure ratio $\geq$ 0.95 and $\geq$ 20 login attempts. We choose these thresholds based on our earlier finding that legitimate users extremely rarely have 20 failed login attempts before success (Figure 3.2). We find 13 remote hosts whose activity falls within these thresholds. Our manual investigation of these determined that

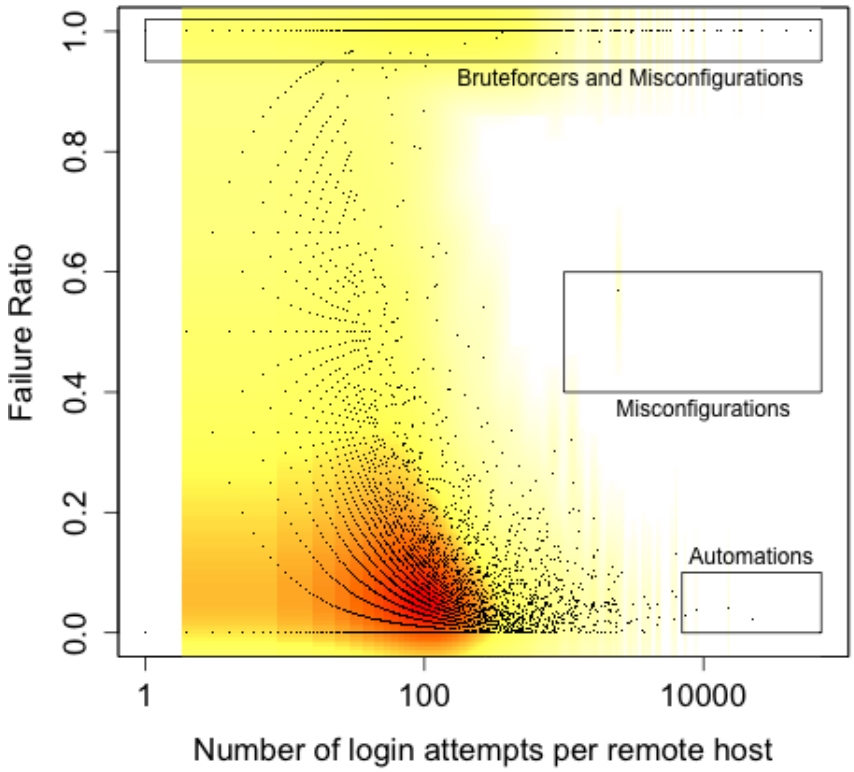


Figure 4.3: Number of logins and failure ratio of remote hosts over the complete dataset. Note that the dataset has been filtered to remove high-rate brute-forcers that can be detected pointwise using per host detection.

six of them reflect misconfigurations (established due to periodicity in the attempts), six eluded classification (activity low enough that they could reflect either legitimate users or participants in coordinated attacks), and one brute-forcer that succeeded in breaking in (clearly an attacker due to employment of a dictionary of generic usernames).

4.2.2 Removing Automations and Misconfigurations

To find candidates for automated activity, we used Zhang's χ^2-based detection [57], which tests for uniformity of when activity occurs in terms of seconds-of-the-minute and minutes-of-the-hour. The premise of this approach is that human-initiated activity should be well-modeled as uniform in terms of these fine-grained timing elements, but automation will generally sharply deviate due to the use of periodic execution.

We applied the test on 3-day windows of activity, requiring each remote client to have at

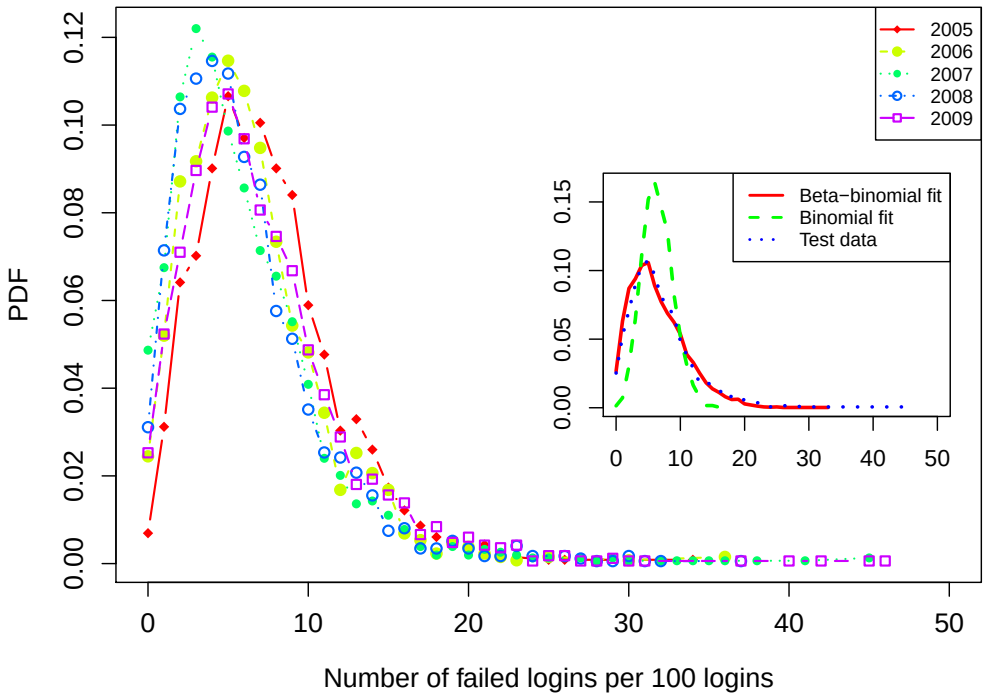


Figure 4.4: Probability distribution of GFI with n=100 logins.

least 50 logins during the window. (We chose the parameters as reflecting clear automation even if spread out over a good amount of time). We used a significance level of 0.001 and a two-sided test in order to detect both non-uniform and extremely-uniform distributions, as both of these likely correspond to automated activity.

The test flagged 363 instances of remote hosts. We manually assessed the 79 of these that triggered detection in multiple windows, since these present instances of long-term automations that can skew our modeling. Doing so found 9 remote hosts that engaged in well-configured long-term automation, and 5 instances of misconfigured automations that frequently failed.[2] As examples, the well-configured automations included jobs that ran: (i) every six minutes for a year, (ii) every ten minutes for two months, (iii) every half hour for two months.

[2] We found it interesting that in some of these cases, when contacted the local administrators were unaware or had forgotten about the existence of this automation.

4.2.3 Deriving the Model

Given the cleaned data, Figure 4.4 then presents the distribution of GFI (using $n = 100$ logins) for five different years. We see a possible trend towards overall less failure from 2005–2008, but 2009 reverses this drift, so we do not attempt to model the prevalence of failure as a function of time.

The figure inset shows the empirical density for 2010 along with two synthetic datasets. First, we fitted a binomial distribution to the 2010 data and randomly generated a new dataset of equal size from that distribution. Second, we applied the same process but instead used a beta-binomial distribution. We see from the inset that the actual data exhibits more variance than we can capture using a binomial model. The beta-binomial model provides a significantly better fit as it allows for an extra variance factor termed *over-dispersion*. Beta-binomial is the predictive distribution of a binomial random variable with a beta distribution prior on the success probability, i.e., $k \sim \text{Binomial}(p, n)$ where $p \sim \text{Beta}(\alpha, \beta)$. Then for a given n, α and β, we have:

$$k \sim \binom{n}{k} \frac{\text{Beta}(k + \alpha, n - k + \beta)}{\text{Beta}(\alpha, \beta)}$$

We can interpret the success of this fitting in terms of lack of independence. If all login attempts were IID, then we would expect to capture GFI effectively using a binomial distribution. The need to resort to a beta-binomial distribution indicates that the random variables lack independence or come from different distributions, such that the probability of success has a beta-prior instead of being constant. This makes sense intuitively because (i) different users will have different probabilities of success, and (ii) the login attempts from a single user are not independent: one failed login attempt affects the probability of success of the next login attempt (negatively if the user has forgotten their password, positively if they simply mistyped it).

4.3 Evaluation

In this section we apply our detection procedure to the extensive LBNL dataset (2005–2012). We discuss parameterizing the detector, assess its accuracy, and characterize the attacks it finds, including whether the attacks appear targeted or indiscriminant.

4.3.1 Parameterization

Our procedure first requires selecting a mean shift $\Delta\mu$ that we wish to detect. We set this to 10 failed logins per event of 100 logins, basing our choice on the stealthiest attack we wish to detect without overburdening the analyst. On average this site sees 500 logins per day, so a threshold of $\Delta\mu = 10$ bounds the number of attempts a brute-forcer can on average make

H	In-control ARL	Out-of-control ARL
1	9	1
10	144	3
20	3,720	5
30	99,548	7
40	2,643,440	9

Table 4.1: In-control and out-of-control ARLs for $k = 5$ and varying values of H.

without detection to 45 (9 attempts × 5 events) spread over a day. Fitting our beta-binomial distribution (§4.2) to the 2005–2008 data yields the parameters $\mu = 7$ and $\sigma = 4.24$, and so our chosen value corresponds to a shift in mean of approximately 2σ. (Note that this is *different* from stating that we detect a "two sigma" event, because due to the cumulative nature of the detection process, it takes significantly more perturbation to the site's activity than simple stochastic fluctuations to reach this level.)

We choose the other parameter, the decision threshold H, based on computing ARLs using the Markov chain analysis sketched in §4.1.1. Table 4.1 shows the *in-control* and *out-of-control* ARLs for $k = 5$ and varying values of H. (We use $k = 5$ based on the rule-of-thumb of setting $k = \frac{\Delta\mu}{2}$ [35].) Given these results, we choose $H = 20$, as this gives a quite manageable expected false alarm rate of one-per-3,720 events, which, given that the site produces about 5 events per day, translates to an average of two alarms per year, and thus an expected 16 false alarms for the complete dataset. This level detects actual stealthy attacks after 5 events (50 brute-forcer login attempts, since the computation is for a shift in the mean of $\Delta\mu = 10$). In a practical setting, $H = 10$ (one false alarm per month) could work effectively.

To validate the assumptions underlying our detection model, we ran the CUSUM detector on the "cleaned" data (per §4.2) to compare the expected false alarms with empirical false alarms. The detector flagged a total of 12 false alarms, reflecting cases where the failure of benign users lead to the alarm.

4.3.2 Assessment of Detection

The two components of our detector can each exhibit false alarms: *false coordinated attack epochs* and *false attack participants*. We can readily identify the former by inspection, as incorrect attack epochs can manifest in one of three ways: (i) the epoch consists of a singleton brute-forcer and a collection of legitimate users who had failures, (ii) the epoch consists of non-coordinating brute-forcers having no apparent coordination glue, and (iii) bad luck: the epoch consists of just legitimate users who failed. The latter kind of false alarms (false attack participants) pose a harder challenge to classify, given we lack ground truth. Since LBNL didn't itself detect and assess the majority of the attacks we detect, we use the following heuristic to gauge whether our procedure correctly classified a remote host as a brute-forcer.

We inspect the the host's login activity in the attack epoch along with its future activity. If none of this succeeded, we can with high confidence deem that host as a brute-forcer. For hosts that ultimately succeed, we confirm whether the success reflected a break-in by checking whether LBNL's incident database eventually noted the activity.

Running the procedure over 8 years of data, the *Aggregate Site Analyzer* detected a total of 99 attack epochs. After then processing these with the *Attack Participants Classifier*, we find nine[3] represent false alarms. We detect a total of 9,306 unique attack hosts participating in the distributed attack epochs, two of which succeed in breaking-in. The procedure classified only 37 benign hosts as attack hosts, reflecting a very low false alarm rate. On days that include an attack epoch, we find on average about 100 benign $\langle R, U \rangle$ pairs filtered out using our past-history assessment, but on average only about 1.7 "forgotten username" instances detected and removed.

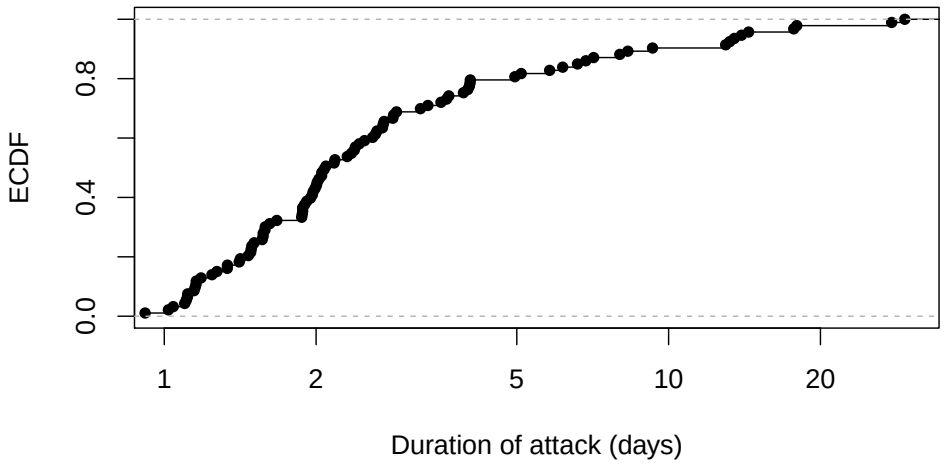


Figure 4.5: Empirical CDF of the duration of attacks (number of days)

Figure 4.5 shows the empirical CDF of the span of detected attack epochs. These coordinated attacks often span multiple days, and sometimes multiple weeks. The majority of the attacks exhibit strong coordination glue in terms of either the set of local machines probed or the usernames targeted for brute-forcing. Of the 90 true attack epochs, 62 have common-set-of-local-machines glue and 25 have username-"root" glue. Only 3 epochs did not manifest any glue we could identify; these epochs probed machines across a wide range of addresses and using a dictionary of generic usernames, such as *mysql* and *admin*.

[3]Note that this number differs from that found earlier on "cleaned" data because some of those false alarms coincided with actual attack epochs, and the *Attack Participants Classifier* then removed them due to a mismatch of coordination glue.

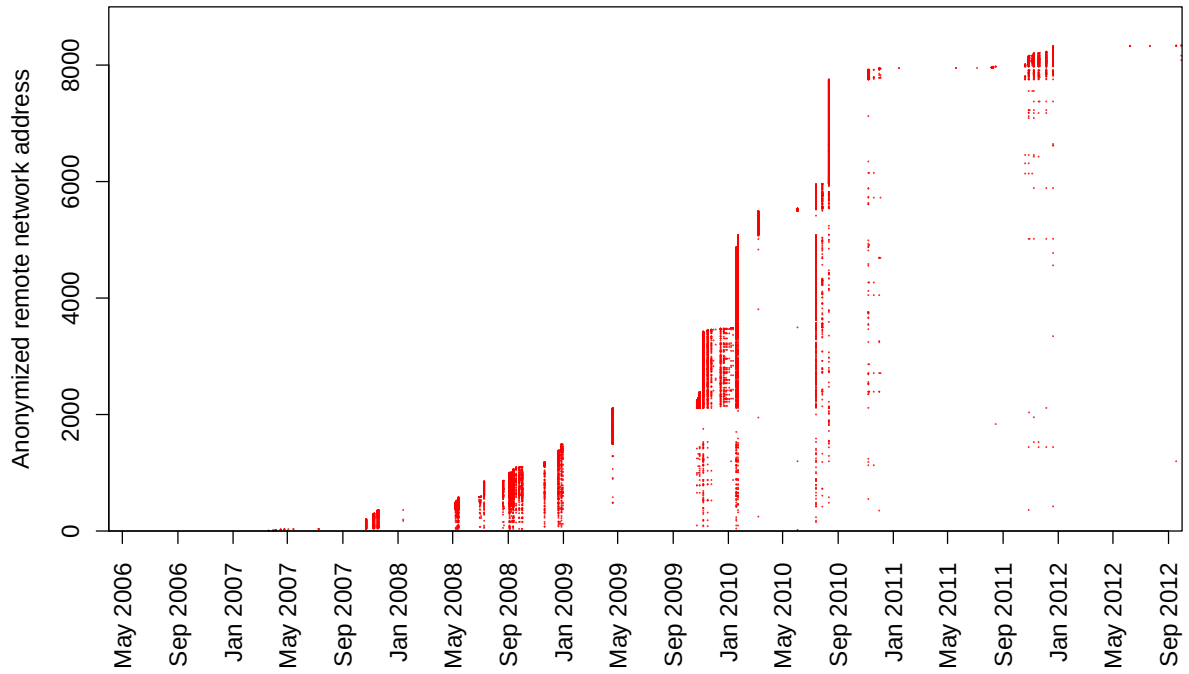


Figure 4.6: Participating attack hosts in the distributed attacks detected from 2005 to 2012 at LBNL.

Figure 4.6 shows the attack hosts participating in the various distributed attack epochs over time, where we number distinct hosts consecutively starting at 1 for the first one observed. The significant overlap of attack hosts across attack episodes shows that many of these attacks employ the same botnet.

We then analyzed the coordination glue in these attack epochs to consolidate the set of epochs into *attack campaigns*. We use the following rules to group epochs into campaigns based on observing evidence that the same attacker conducting different attack epochs that work towards the same goal: (i) epochs with the same common-set-of-local-machines coordination glue, and (ii) epochs appearing on the same day with username-*root* coordination glue. Our detector considers these as multiple attack epochs rather than a single attack because this is indeed how the campaign proceeds, stopping for a few hours/days and then reappearing. Using these rules, we group the 62 attacks with common-set-of-local-machines glue into 12 distinct attack campaigns. Only a few of the 25 epochs group using heuristic (ii), condensing the root-set to 20 campaigns. This leaves us with a total of 35 attack campaigns.

Table 4.2 summarizes the magnitude, scope and stealthiness of the attacks we detect. All of these attacks were stealthy when observed from the viewpoint of individual hosts; on average the attack hosts made ≈ 2 attempts per local machine per hour. We can however detect

ID	Appearances	Attrs.	Aggregate statistics		Per remote avg. hourly characteristics		
			Attack machines	Local machines	Attempts	Locals contacted	Per-Local attempts
1	2007: [Jul 7-9], [Oct 20-23], [Nov 5-9](2), [Nov 13-18](2)	**L,!!**	431	133	74.68	56.10	1.33
2	2008: [Apr 30 - May 7],[May 8-14](3)	**L**	286	140	98.50	54.80	1.79
3	2008: [Jun 28-29], [Jun 30 - Jul 1] [Jul 7-9], [Aug 17-21], [Sep 1-8] (5)	**L**	969	113	293.30	41.70	7.00
4	2008: [Sep 8-13](3)	**L**	378	257	52.50	40.70	1.28
5	2008: [Sep 16-18]	**L,S,T**	88	12	9.00	2.53	3.57
6	2008: [Sep 23-26](2), [Sep 29 - Oct 2](2)	**L**	185	109	48.50	38.38	1.26
7	2008: [Nov 18-19], [Nov 20 - Dec 29](5) 2009: [Apr 7-9]	**L,S**	1,097	22	16.01	8.04	1.99
8	2009: [Oct 22-23], [Oct 27 - Nov 24](5)	**L,S**	1,734	5	5.60	3.70	1.50
9	2010: [Dec 6 - Jan 10](6), [Jan 11-18], [Jan 20-22], [Mar 4-8]	**L**	3,496	44	38.80	21.50	1.80
10	2010: [Jun 16 - Jul 27](2), [Jul 29 - Aug 11]	**L**	7,445	1,494	90.80	34.50	2.70
11	2010: [Nov 1-6] (2), [Nov 7-8], [Nov 27 - Dec 1], [Dec 15-17]	**L,!**	581	98	140.60	45.47	3.09
12	2011: [Oct 11-19], [Oct 25-29](2), [Nov 4-7], [Nov 17-20]	**L**	377	158	33.93	25.25	1.34
13	2010: [Mar 30 - Apr 1]	**R,t**	78	18,815	999.70	118.91	1.33
14	2010: [Apr 23-26]	**R,t**	130	29,924	2325.57	117.97	1.22
15	2010: [May 7-10]	**R,t**	72	9,300	713.05	67.47	1.36
16	2010: [Sep 20-22]	**R,t**	33	5,380	69.05	60.72	1.14
17	2010: [Dec 27-30]	**R,t**	32	3,881	260.59	43.11	1.34
18	2011: [Feb 10-14](2)	**R,t**	108	7,520	40.45	27.21	1.48
19	2011: [May 16-18]	**R,t**	30	1,621	153.23	19.70	2.02
20	2011: [Jul 21-22]	**R,t**	20	2,556	388.25	38.13	1.18
21	2011: [Aug 2-6]	**R,t**	45	9,465	315.12	21.66	2.41
22	2011: [Aug 7-9]	**R,t**	48	6,516	444.16	17.60	2.18
23	2011: [Aug 17-21](2)	**R,t**	22	3,279	33.07	16.40	2.02
24	2011: [Nov 2-4]	**R**	31	3,446	273.80	20.08	1.02
25	2011: [Nov 30 - Dec 5]	**R**	181	10,467	829.68	18.31	1.03
26	2011: [Dec 18-20]	**R**	258	961	1099.85	14.00	1.02
27	2012: [Jul 20-21]	**R,t**	2	53,219	20,844	11,749	1.06
28	2012: [Aug 27 - Sep 2]	**R,t**	10	1,912	20.84	14.38	1.23
29	2012: [Sep 26-29]	**R**	6	1,971	72.30	13.05	1.59
30	2012: [Oct 8 - Nov 1](4)	**R,S**	190	19,639	5.27	4.97	1.06
31	2012: [Nov 16-18]	**R,t**	3	493	38.36	12.22	2.99
32	2012: [Nov 30 - Dec 2]	**R,t**	3	344	133.00	68.80	1.93
33	2008: [Jan 9-12]	**X,t**	17	63,015	2,846.44	1,761.69	1.61
34	2011: [Apr 8-26]	**X,t**	67	19,158	591.34	87.41	6.76
35	2012: [Dec 14-17]	**X,t**	13	45,738	1,490.26	1,430.67	1.04

Table 4.2: Characteristics of the detected coordinated *attack campaigns*. In *Appearances*, numbers in parentheses reflect how many attack epochs occurred during the given interval. *Attrs.* summarizes different attributes of the activity: **L** = coordination glue was set of local machines, **R** = coordination glue was username "root", **X** = no discernible coordination glue, **S** = stealthy, **T** = targeted, **t** = possibly targeted but no corroborating evidence, **!** = successful, **!!** = successful and apparently undetected by the site.

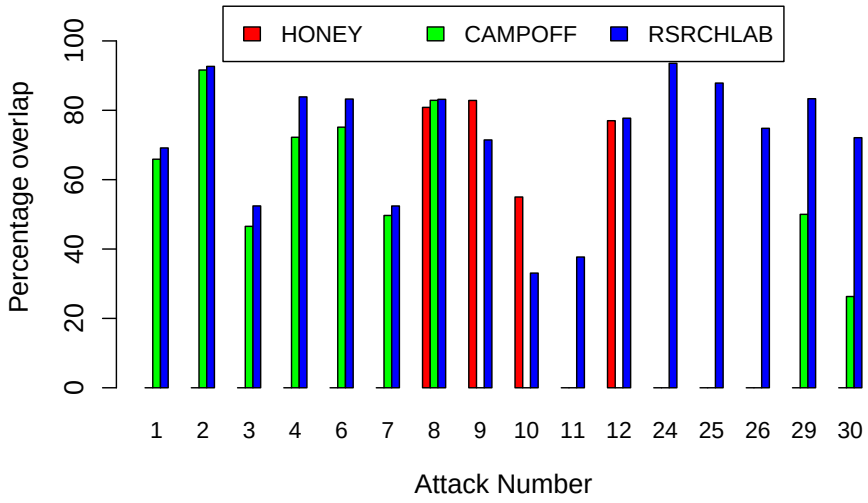


Figure 4.7: Percentage overlap of attack hosts seen at LBNL with that at sites HONEY, CAMPOFF and RSRCHLAB. The figure displays only the subset of attacks that appear in at least one of the three sites. (Note that none of the attacks appear in HOMEOFF).

a large fraction of these attack campaigns using a point-wise network-based detector that looks for high-rate hourly activity in terms of either the total number of failed attempts or the number of local hosts contacted. Note that we also detect attacks that a site cannot detect using either host-based or network-based point-wise detection (campaigns 5, 7 and 8 in Table 4.2). Finally, two of the campaigns succeeded, the first of which (campaign 1) as best as we can tell went undetected by the site.

We also find a difference in the characteristics between attacks that have set-of-local-machines coordination glue versus the ones that only have username-*root* glue. The latter tend to target a wide range of the site's address space and often involve just a few attack hosts brute-forcing at a high rate. Attacks having set-of-local-machines coordination glue often exhibit the pattern of the attackers stopping and coming back. We did not find any sequential pattern in any of these campaigns; rather, the attackers targeted servers spread across the address space, often including addresses in both of LBNL's distinct address blocks. We also did not find any pattern among the local servers in terms of belonging to the same research group or compute cluster.

4.3.3 Establishing the Scope of Attacks

Next, we attempt to establish which attacks specifically targeted the LBNL site versus global attacks that indiscriminantly probed the site. To do so we look for whether the attack hosts of a given campaign appeared in any of our four correlation datasets, HONEY, RSRCHLAB, HOMEOFF, and CAMPOFF.

We find that 16 campaigns appear in at least one of these four datasets. These include five username-root coordination glue attacks and all but one of the attacks with set-of-local-machines coordination. Figure 4.7 plots the percentage overlap of the attack hosts detected in the global attacks at LBNL with that at other sites, showing a high overlap in most cases. We investigated campaign 5, which does not appear at any of the other sites, and found that it indeed targeted LBNL, as the attack hosts all probed a set of six usernames each valid at the site. As shown by the hourly rates in Table 4.2, this targeted attack also proceeded in a stealthy fashion, with each remote host on average making only 9 attempts and contacting 3 local servers per hour. It's possible that some of the other campaigns also specifically targeted LBNL, though for them we lack a "smoking gun" that betrays clear knowledge of the site.

Finally, to give a sense of the nature of global attacks, Figure 4.8 shows the timing patterns of login attempts at the LBNL and HONEY sites during part of campaign 8. From the clear correlation (though with a lag in time), we see that the activity at both reflects the same rate (which varies) and, for the most part, the same active and inactive periods.

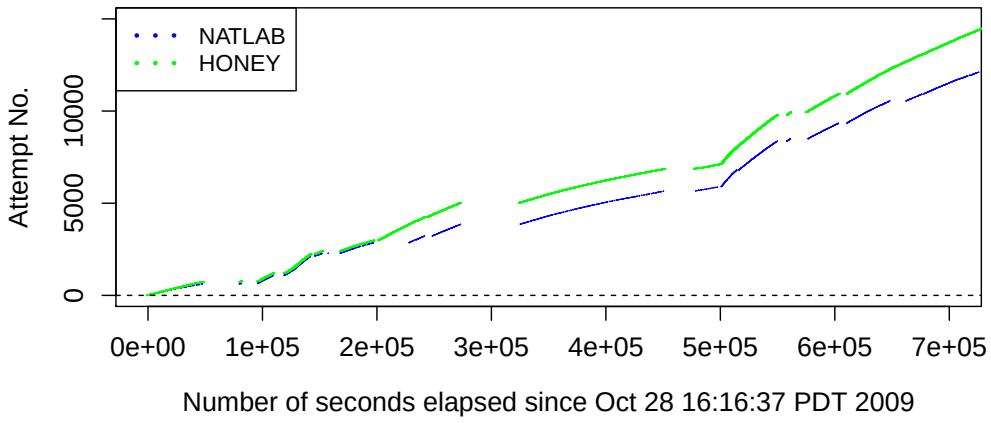


Figure 4.8: Timing of login attempts at HONEY machine and LBNL sites during part of attack number 8 (Oct 2009 - Nov 2009). The plot is based on data for only one of the machines targeted during the attack at LBNL.

4.4 Summary

In this chapter, we proposed a general approach for detecting distributed, potentially stealthy activity at a site. The foundation of the method lies in detecting change in a *site-wide* parameter that summarizes *aggregate* activity at the site. We explored this approach in concrete terms in the context of detecting stealthy distributed SSH brute-forcing activity, showing that the process of legitimate users failing to authenticate is well-described using a

beta-binomial distribution. This model enables us tune the detector to trade off an expected level of false positives versus time-to-detection.

Using the detector we studied the prevalence of distributed brute-forcing, which we find occurs fairly often: for eight years of data collected at a US National Lab, we identify 35 attack *campaigns* in which the participating attack hosts would have evaded detection by a pointwise host detector. Many of these campaigns targeted a wide range of machines and could possibly have been detected using a detector with a site-wide view, but we also find instances of stealthy attacks that would have proven very difficult to detect other than in aggregate. We correlated attacks found at the site with data from other sites and found many of them appear at multiple sites simultaneously, indicating indiscriminant global probing. However, we also find a number of attacks that lack such global corroboration, at least one of which clearly targeted only the local site. Some campaigns in addition have extensive persistence, lasting multiple months. Finally, we also find that such detection can have significant positive benefits: users indeed sometimes choose weak passwords, enabling brute-forcers to occasionally succeed.

Chapter 5

Detection of Compromised Credentials

In this chapter, we draw upon a year's worth of remote-access logs from LBNL to detect *anomalous* logins that indicate the user credentials may have been potentially compromised. This detection problem poses major challenges: incidents of successful compromise might strike a given enterprise less than once a year, which means that finding them is analogous to looking for needles in haystacks. To tackle this challenge, we first extensively characterize the "haystack", with the goal of understanding the volumes and diversity of network login activity at our enterprise. We then draw upon what we learn to guide the development of features that with good confidence can serve as indicators of compromise. This feature engineering exercise provides the basis of our detector, which continually refines a profile of each enterprise user as well as the properties of the machines from/to which they log in, and gauges the legitimacy of a login using both history-based features and the current context of the login.

This chapter proceeds as follows: in §5.1, we develop techniques to identify the context of a given login, i.e., whether it occurs as part of an automated cron job or as part of an existing login session. §5.2 discusses our feature engineering for different properties of the login (IP address, SSH client version, geo-location transition), as well as the state of user (traveling/on-site/off-site-but-in-town). In §5.3, we develop a rule-based system to identify anomalous logins by combining different features. In §5.4, we discuss the results of running our detector on several months of LBNL data.

5.1 Identifying the Context of Logins

Establishing the context in which a login occurs forms the basis of our exploration — without the context, features that assess sequences of logins can be misleading. For example, consider the scenario of a user accessing the enterprise network from their home: they might first log in from their (external) home machine H to the enterprise machine A, and a few minutes later use the same session to step from machine A to another machine B at the enterprise.

Without the context that the latter login $A \rightarrow B$ is nested within the external login, $H \rightarrow A$, it is unclear whether both logins are from the actual user. Similarly, a scripted *local* login that occurs every day at noon, might lead one to believe that a close-in-time *external* login is suspicious, based on the incorrect inference that the user was present at the enterprise network at noon.

In this section, we develop techniques to identify: (*i*) automated logins that do not require any user action at the time, i.e., are part of a scheduled cron job, and (*ii*) nested logins that are part of a broader session — these logins involve the user stepping through another server, and hence do not require the user to be physically present at the origin of login.

5.1.1 Automated Activity

To find candidates for automated activity, we used the Zhang's χ^2-based detection (sketched earlier in Chapter §4). For each ⟨`user, client, server`⟩ triplet that appears in the syslog dataset, we applied the test to 60-day windows of activity, if the triplet had at least 50 logins during the window. (We chose the parameters as it allows us to detect activity that repeats even daily over long periods of time). We used a significance level of 0.001 and a one-sided test to detect non-uniform distributions.

Next, we labeled each login appearing from these triplets as either *manual* or *automated*. To do so, we identified the period of the automation by testing the login activity for the common automation periods {`5m,10m,15m,30m,1h,24h`}, and labeled the logins that occur at the period as *automated*. Restricting the period-finding algorithm to a limited set of periods allows us to keep the brute-force computation manageable, while finding the period for majority of the triplets. We manually checked the triplets for which the algorithm did not flag any period and found that most of these were false detections (i.e., although non-uniform at the minute-of-the-hour, not periodic). The manual analysis also found a few automations at intervals of two and six minutes.

The above approach flagged 27% of the logins in our dataset as automated (these correspond to activity of 45 different users). For the rest of this work, we filter out the automated logins, since periodic activity is highly likely legitimate.

5.1.2 Nested Activity

The source of a remote login does not necessarily reflect the true location of the user at that time. The ability to step through machines means that we may see local logins for a user even if they are physically external to the site, and vice versa. Therefore, inferring the location from individual logins is incorrect in such scenarios where the user steps through machines. In order to infer the true physical location of the user, we need an approach for identifying logins that are *nested* within another session.

Before discussing how we develop this approach, we first sketch the possible remote-login scenarios and to what extent our site has visibility (and the corresponding data) for the complete activity in the scenario. Some of the visibility limitations are due to how our data was collected (such as we cannot tell how long sessions corresponding to *local* logins last because of lack of flow data), while others hold generally for any site (such as the inability to observe the user logging in from one external machine to another).

5.1.2.1 Remote-Login Scenarios and Site Visibility

Direct logins. Figure 5.1 sketches the direct remote-access possibilities for the two cases of when the user is on-site and when they are off-site. 5.1(a) illustrates a user who is physically at the site and is using machine `A`. She can directly access other machines at the site, for which we observe *local* logins in the syslog data,[1] but no corresponding entry in flow data (since the activity does not cross the border). She may also directly connect to machines outside the site, for which we observe *outbound* activity in the flow data, but no corresponding record in the syslog data because the site cannot monitor the external machines.

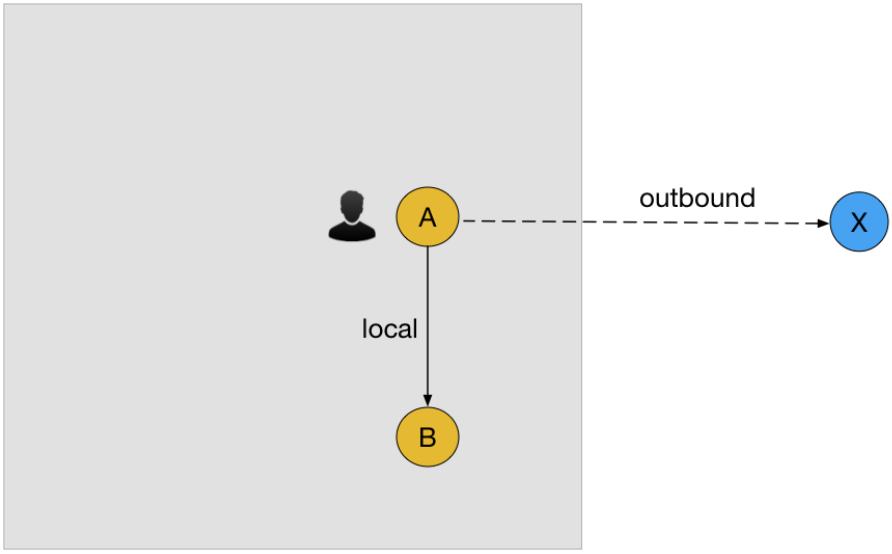
Similarly, 5.1(b) illustrates the scenario where the user is external to the site. She can log in to machines at the site, for which we will observe both *inbound* login records in the syslog data and corresponding activity in the flow data. The user can also connect to other external machines for which we do not have any visibility. (In general, this external-to-external activity is not relevant to the site, and may involve different credentials than what the user has on the site. However, the lack of this visibility becomes relevant in the nesting scenarios that we discuss next).

We introduce the following notation for the three types of direct logins: local (l), inbound (i), and outbound (o).

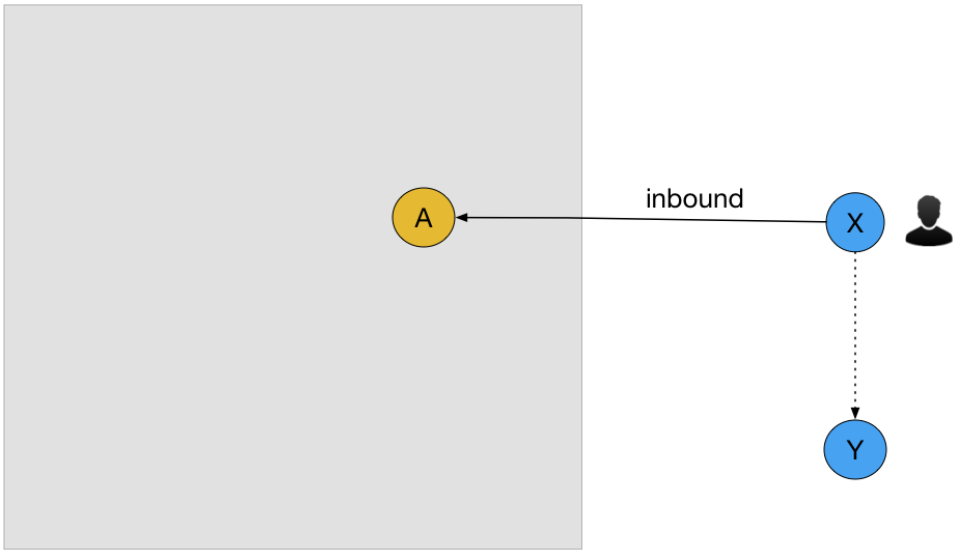
Nested logins. Nesting means that after a direct login, the user uses the same session to step through machines, thereby resulting in a chain of logins. Theoretically speaking, these chains can be of any length. However, each possible chain is a combination of a few base cases, which reflect the nesting scenarios where the destination of the parent login is the source of the child login. Table 5.1 enumerates the base cases of nesting by checking whether each login type (l, i, o) can be directly nested in any of the three types of logins. There are five possible nesting base cases: (l in l), (l in i), (i in o), (o in l), and (o in i). The others cannot occur because it is impossible for the user to perform a login of the second type from the destination of the first type of login for those cases. For example, it is not possible for (l in o) to occur because the destination of a login type o is a machine external to the site, and the user cannot directly have a local login from an external machine.

We now discuss the base nesting cases and sketch site visibility limitations that affect our grouping of logins into sessions and finding the origin of nested logins.

[1] Assuming that the machine is set up to syslog.



(a) Direct access possibilities for a user physically at the site.



(b) Direct access possibilities for a user physically external to the site.

Figure 5.1: Direct remote-access scenarios. Yellow circles represent machines at the site while blue circles represent machines external to the site. Solid lines indicate syslog visibility, dashed lines indicate flow visibility, dotted lines indicate the site cannot observe that the remote access happened.

	Parent Login		
	Local (l)	Inbound (i)	Outbound (o)
Local (l)	Yes	Yes	No
Inbound (i)	No	No	Yes
Outbound (o)	Yes	Yes	No

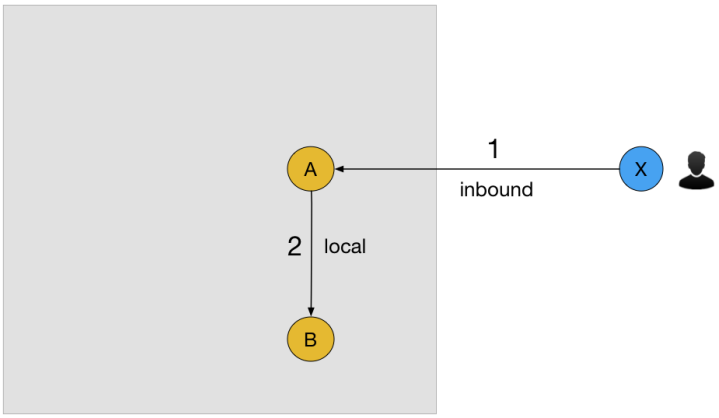
Table 5.1: Exploration of the possible base nesting cases. Yes means it is possible for a login type indicated by the row to be directly nested in a login of the type indicated by the column. No means direct nesting is not possible.

First, we note that session durations are the key to establishing the nesting of logins; if a login occurs within the duration of another login for a base case, we can infer that the second login was caused by the first. However, we lack the durations of local logins, and consequently are unable to establish nesting for the (l in l) and (o in l) base cases. We discuss nesting for the other three cases (l in i), (i in o) and (o in i) next. Figure 5.2 sketches example scenarios for these cases.

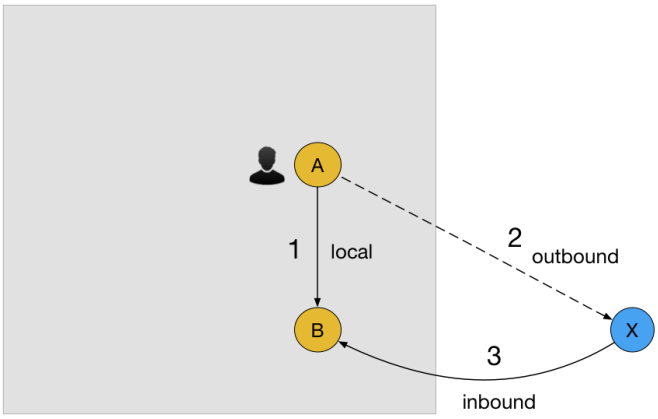
5.2(a) is the simplest and presumably the most common one (l in i), where the user at an external machine `X` first logs into machine `A` on site and then steps through it to access another machine `B` on site. This nested activity can be grouped together using the session duration information, since the login `A` $\rightarrow$ `B` occurs within the duration of the external session `X` $\rightarrow$ `A`. To group, it suffices to check that the starting time of the nested activity (`A` $\rightarrow$ `B`) is within the duration of the external session `X` $\rightarrow$ `A`. The end time of the nested session may not necessarily be within the duration of the external session, because the login may correspond to a data transfer that can continue even after the external session has ended.

5.2(b) sketches an (i in o) scenario where the user is physically at the site and has a workflow that involves accessing off-site machines. We first see a *local* login for this user from machine `A` to `B`, and shortly after she connects from `A` to the external machine `X`. While logged in to `X`, she pulls over some data from machine `B` to this external machine. This data transfer shows up in the logs as an inbound session `X` $\rightarrow$ `B`. For this scenario, we can group the inbound login `X` $\rightarrow$ `B` with the outbound session `A` $\rightarrow$ `X` using session durations.

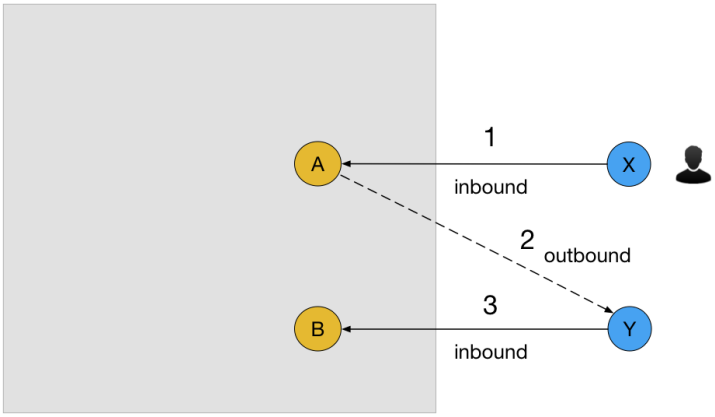
5.2(c) sketches an (o in i) case where the user at an external machine `X` first logs into machine `A` on-site and then steps through it to access an external machine `Y`. Since we only observe outbound activity in the flow data, we cannot associate the user with the outbound login `A` $\rightarrow$ `Y` at the time of login, and consequently cannot identify that it is nested within `X` $\rightarrow$ `A`. However, in this example scenario the user later logs in from an external machine Y to machine B. Since the source of this inbound login is the same as the destination of the outbound flow, we can associate the user with the earlier outbound activity (`A` $\rightarrow$ `Y`) and at that point identify the (o in i) nesting. We note here that identifying a singular (o in i) nesting that does not result in further nested logins is not necessary for our work, since it does not affect our assessment of sequences of logins.



(a) User at an external machine X logs into machine A at the site and steps through it to access machine B.



(b) User at a local machine A logs into machine B at the site, then connects to an external machine X (outbound connection) and pulls data from machine B to machine X (inbound connection).



(c) User at an external machine X logs into machine A at the site, then connects from A to the external machine Y. She then pulls data from machine B to machine Y.

Figure 5.2: Three example nesting scenarios involving users stepping through machines.

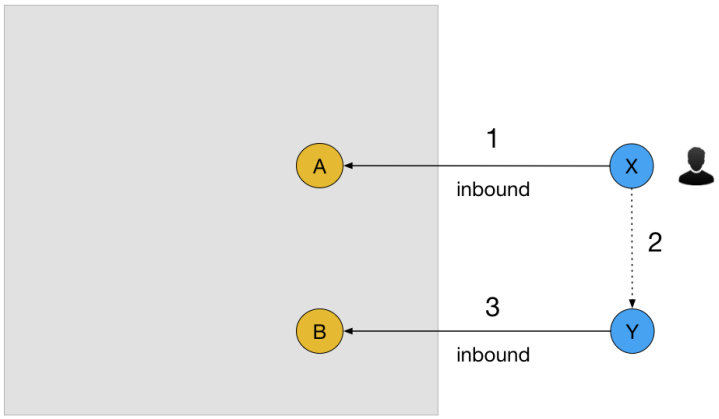


Figure 5.3: Example scenario showing external nesting. User at an external machine X logs into machine `A` at the site, and then steps from machine X to another external machine `Y` (which may be geographically distant). She later connects from `Y` to machine `B` on site.

Nesting Outside the Site's Monitoring Boundary. Figure 5.3 sketches a scenario of nesting that does not fit within the framework of the nesting scenarios that we try to identify in this work, but we discuss it here for completeness. The user is physically external to the site (using machine `X`) and has a workflow that involves connecting to another external machine. We first observe an *inbound* login `X` → `A` to the site at t_1 for this user. Some time later at t_2, the user connects from `X` to another external machine `Y`, for which the site lacks visibility.[2] From this external machine `Y`, she then logs into the site machine `B` at t_3. This scenario involves a series of three logins: `X` → `A`, `X` → `Y`, and `Y` → `B`.

Note that the second inbound login `Y` → `B` may or may not occur within the duration of the first inbound login `X` → `A`. This is because the user is free to log in from `X` to `Y` at any time irrespective of their `X` → `A` login. For the case where the first inbound session is active when the second one starts, we can potentially group them together using session durations. However, we note that a login from an attacker might manifest in the same fashion: overlapping in time with the actual user. Therefore, grouping an inbound login within another inbound login can risk missed detections. (In the next section, we characterize the base rate of how often inbound logins are nested within other inbound logins for legitimate users).

Missing syslog data. Consider the more complicated nesting scenario shown in Figure 5.4. The user is physically external to the site and is using machine `X`. She first logs into the login node of the cluster `C` on-site, and from there connects to the cluster node `D`, which is not publicly accessible and is not set up to syslog. After that, she copies over some data from the cluster node D to her machine `A` on the site. This scenario involves a series of three logins: `X` → `C`, `C` → `D`, and `D` → `A`.

If all the local machines in this scenario syslogged, both *local* logins would satisfy two

[2]Note that the machine `Y` may be located geographically far away from machine `X`.

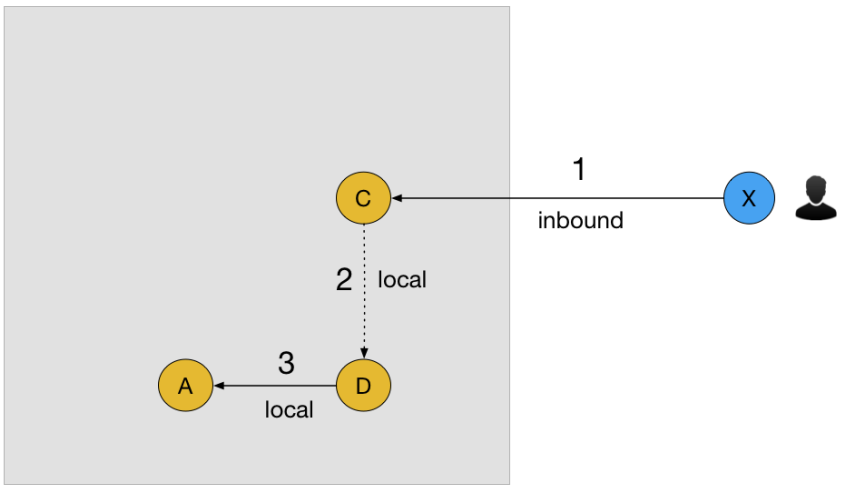


Figure 5.4: Example scenario involving the user stepping through a machine that does not syslog: a user at the external machine `X` logs into the cluster `C` at the site and steps through it to access the cluster node `D`, which does not syslog. She then copies data from `D` to her machine `A`.

properties: (*i*) they occur within the duration of the inbound session `X` → `C`, and (*ii*) the destinations `D` and `A` are *transitively reachable* from `X`. However, since `D` does not syslog, we only observe the *inbound* login `X` → `C` and the *local* login `D` → `A` in the logs. Given just these two logins, it appears that `A` is not transitively reachable from `X`. However, `D` → `A` still satisfies the duration property for grouping with `X` → `C`.

If a considerable proportion of servers in the network do not syslog, and this scenario occurs often in the regular activity of the users, then requiring that the logins satisfy transitive reachability in order to be grouped will result in a lot of missed groupings. As a consequence, we will incorrectly infer that the user is *local* to the site, when in fact they are not. We assess how often this happens for our dataset in the next section, and base our decision of relaxing the transitive reachability requirement on this assessment. (Note that relaxing this requirement raises the possibility of incorrectly grouping the user's local activity with an attacker's logins).

5.1.2.2 Characterizing Session Activity

Given the nesting scenarios that we discussed in the previous section and the ambiguity they lead to in inferring a user's true physical location, we develop an approach for determining whether a new login that arrives is part of an existing *inbound* or *outbound* session. Since a site may lack complete visibility into the true activity, both due to missing syslog data from its own servers as well from servers outside its monitoring boundary, in this section we first characterize session activity in order to get a sense of to what extent: (*i*) our site lacks syslog visibility for its own servers, and (*ii*) users participate in external-external sessions (the

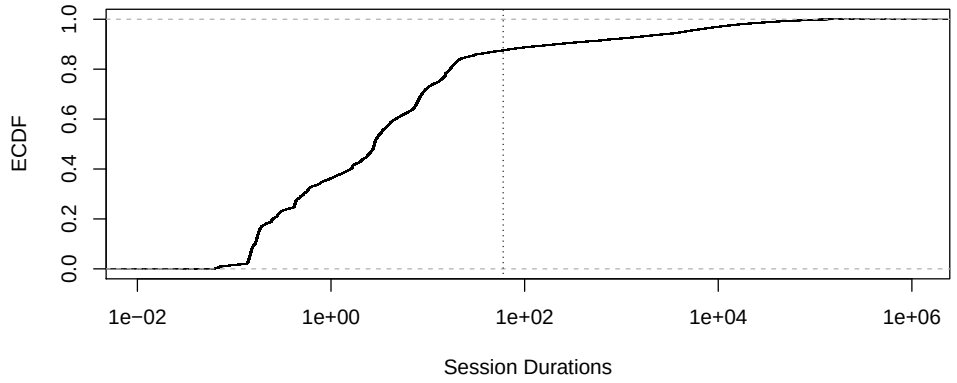


Figure 5.5: Empirical CDF of inbound session durations across all users (in seconds). The vertical line represents 1 minute.

scenario in Figure 5.3) for which the site lacks monitoring coverage. This characterization informs both our approach for associating new logins with existing sessions as well as our feature engineering later. For this characterization, we use the first five months of the data (Jan – May) and filter out any automated logins we identified in the previous section.

Inbound Sessions: We use the following grouping algorithm to characterize nested/concurrent activity for inbound sessions. For each user, we maintain a list of their active inbound connections. When a new local or inbound login arrives, we check if the arrival time is within the duration of any of the active inbound sessions, and use the following rules to associate the login with existing active sessions:

- If there are multiple active connections for the user, we associate the login with the session that satisfies transitive reachability. For example, if a user has two active connections, $X \rightarrow A$ and $X \rightarrow B$ at time t when we see a new login $B \rightarrow C$ for the user, we associate this new login with the inbound session $X \rightarrow B$, since it satisfies transitive reachability.
- If multiple on-going sessions satisfy transitive reachability, we associate the login with the most recently initiated session.
- If none of the sessions satisfy transitive reachability, we still associate the login with the most recent active inbound session, since some servers in the enterprise might not syslog (per Figure 5.4), and we lack syslog visibility for concurrent external logins (per Figure 5.3).

Total	194,670	
Nested/Concurrent	14,060	(7%)
- External-external		
Total	8,429	
Self-external (same remote IP)	5,153	
Other-external	3,276	
Same ASN	1,696	
- External-local		
Total	5,631	
Missing transitive reachability paths		
Total	1,962	
Due to missing syslogs	1,223	
Number of servers missing syslog	532	(24%)

Table 5.2: Inbound session characteristics over five months of data.

Figure 5.5 shows the ECDF of inbound remote-login session durations. 88% of the inbound sessions last less than a minute, but there is a long tail, with some sessions spanning multiple days. Table 5.2 shows a breakdown of the characteristics of session activity of these inbound connections after grouping logins into sessions. 93% of the sessions are stand-alone, i.e., no nested or concurrent activity occurs during the duration of the session. This agrees with the observation that a large fraction of the sessions are short lived. For 7% of the inbound sessions we observe concurrent/nested activity. We characterize the activity of these sessions as follows:

External-external: 58% of the sessions involving concurrent/nested activity fall into this category, which involves logins only from other external locations. These can be further categorized as: (*i*) self-external (same IP address), and (*ii*) other-external (external but a different IP address), with a 60-40% breakdown. Approximately half of the cases from the second category involve IP addresses from the same ASN — these are users at collaborator institutions logging in from multiple servers at their institution concurrently. For the sessions involving different ASNs, 80% have one end-point in one of the three collaborator institutions,[3] with some users having such session-patterns repeating across multiple days, indicating this is a workflow they use routinely.

External-local: Of the ≈5.6K inbound sessions that contain a local login, 35% have a local login that does not satisfy transitive reachability. This occurs for two reasons: (*i*) missing syslog data, and (*ii*) long-lived connections. The former is the dominant case (62%), and happens primarily because the user accessed a server that has a DHCP-assigned IP address and is not set up to syslog. A subset of the users (81) use long-lived connections that cause

[3]University of California at Berkeley, National Energy Research Scientific Computing Center, and Howard Hughes Medical Institute.

the transitive reachability path to be incomplete — the user logs into a machine at the site which already has an active session to another local machine. (Recall that we do not have session durations for local logins to establish on-going sessions between local machines). The top five users account for most (80%) of such cases.

Outbound Sessions: We infer that an inbound login is part of an on-going outbound session if there exists an active outbound session whose destination is the same as the source of the inbound login. Over the five months, we observed ≈5.9K such occurrences of inbound logins nested within an outbound session.

5.1.2.3 Identifying the Origin of Nested Logins

Based on the characterization in the previous section, Algorithm 2 sketches our approach for grouping logins into sessions and identifying the *origins* of logins, where *origin* is defined as the source IP address of the root of the session in which the given login is nested. We maintain a list of active inbound connections for each user as well as a global list of outbound connections for each server. To associate a new *local* login with an existing inbound session, we only check if it occurs within the duration of the inbound session (relaxing the transitive reachability condition due to missing syslog data in our dataset). To associate a new *inbound* login with an existing outbound session, we check whether it occurs within the duration of an active outbound session whose destination is the same as the source of the inbound login. We do not associate a new *outbound* login with an existing inbound session at the time it occurs because of the ambiguity in the user to which the outbound login corresponds. However, if we observe a subsequent inbound login nested within the outbound session, we retrospectively identify if the outbound login occurs within the duration of an inbound login. To identify multiple levels of nesting, we keep a record of the origins of all the logins that have occured in the past. For each new nested login then, we set its origin to the origin of the login in which it is nested.

```
1  X : {x_i} where x_i is a login event with the following attributes:
       x_i.type ∈ {l, i, o}
       x_i.duration: duration of the login
       x_i.start_time: start time of the login
       x_i.dest: destination IP address of the login
       x_i.source: source IP address of the login
       x_i.origin: source IP address of the root of the session in which x_i is nested.
       This field is initialized with the source IP address of the login itself.
2  S: [x_1, x_2, x_3, ..., x_n] where x_i ∈ X // sequence of logins for a given user. (For brevity, we
   show how the algorithm works for one user; we assume that S contains local and inbound
   logins for a single user and outbound connections from all servers).
3  I: list of inbound logins seen until a given point in time, i.e., x_i ∈ X where x_i.type = i
4  O: list of outbound logins seen until a given point in time, i.e., x_i ∈ X where x_i.type = o
5  I ← [] // initialize to an empty array
6  O ← [] // initialize to an empty array

7  function findAllOrigins(S)
8      for t: 1 to n do
9          findLoginOrigin(x_t)

10 function findLoginOrigin(x_t)
11     if x_t.type == l then:
12         root = findRootIfNestedInInbound(x_t)
13         if root is not NULL then:
14             x_t.origin = root
15     if x_t.type == i then:
16         root = findRootIfNestedInOutbound(x_t)
17         if root is not NULL then:
18             x_t.origin = root
19         append x_t to I // update the list of inbound logins
20     if x_t.type == o then:
21         append x_t to O // update the list of outbound logins

22 function findRootIfNestedInInbound(x_n)
23     for y_i ∈ I do
24         if x_n.start_time within duration of y_i then
25             return y_i.origin
26     return NULL
```

Algorithm 2: Finding the origins of logins.

```
27 function findRootIfNestedInOutbound(x_n)
28     for y_i ∈ O do
29         if x_n.start_time within duration of y_i and x_n.source == y_n.dest then
30             root = findOutboundLoginRoot(y_i) // before this call we do not know
   the origin of the outbound login because we are unable to associate the user with the
   login until this point
31             if root is not NULL then:
32                 y_i.origin = root
33             return y_i.origin
34     return NULL

35 function findOutboundLoginRoot(y_n)
36     return findRootIfNestedInInbound(y_n)
```

Algorithm 2: Finding the origins of logins (cont'd).

5.2 Feature Engineering

In this section, we discuss the development of features that feed into our rule-based detection system. We explore two categories of features: (*i*) history-based, and (*ii*) context-based. The history-based features assess whether the current activity of the user is *new* compared to a historically learnt profile. The context-based features infer whether the activity looks anomalous given the recent sequence of logins. We use the first five months of the dataset to guide the development of the features.

Bootstrapping window for history-based features. For history-based features, the longer the duration of training data available to bootstrap the history, the better the accuracy of the features will be. For example, consider the *"new SSH client version"* feature, which determines whether the client version in the current login has been observed for the user in the past. The more history available for a given user, the higher opportunity the detector will have to comprehensively capture the set of SSH client versions that the user legitimately uses. However, a detector that requires a lot of training data in order to achieve good performance will not be able to detect compromise for accounts that have not accumulated enough history, either because they are relatively new, or because the user only lightly uses the account. Therefore, a shorter bootstrapping window trades-off capturing diversity in the history in individual accounts in favor of detecting compromise for a wider user-base (users with both sparse and dense activity).

As we sketched in the data characterization in §3.3.2, most users in our dataset have sparse external login activity (Figure 3.5). Keeping this in view, we use a short bootstrapping window in this work: activity from *five* distinct days on which the user logged in to the site

from an external location. Note that it may take several weeks to collect this five-day history for users who are rarely active from external locations.

We next discuss the exploration of the following set of features: (*i*) location transitions, (*ii*) login IP address reputation, (*iii*) SSH client version history, and (*iv*) whether the user is traveling.

5.2.1 Location Transitions

For each new login that is not part of an existing session, we determine if it results in a geo-location transition for the user, and whether the location transition is *consistent* or *inconsistent*. We call a geo-location transition *consistent* if it is physically possible for the user to travel from the location of last login to the location of current login during the time elapsed between the two logins.

We query the `Maxmind` database to get the `lat/long` coordinates of the IP addresses involved in the last and current login [9]. We then compute the approximate travel-time between the two geo-locations as follows: determine the `great circle` distance between the two geo-locations. If the distance is less than 200 miles, compute the driving time (at a speed of 50 mph), else compute the flight time (at a conservative cruise speed of 500 mph) plus take-off/landing time (one hour). We query the Google Distance Matrix (GDM) API to get more accurate driving times where available. The API does not provide data for other modes of transportation [6].

In addition to determining whether a location transition is consistent or not, we assign the following tags to the location transitions:

- *local movement:* if the transition involves a distance less than 200 miles.
- *city transition:* if the transition involves two cities in the same country, more than 200 miles apart.
- *foreign transition:* if the transition involves two different countries (one of which can be US).

Inconsistent transitions: Next, we explore the power of the *inconsistent* location transition feature. The idea behind this feature is that *inconsistent* location transitions are anomalous to some degree — an attacker in a geographical location far from the user who accesses the compromised account close-in-time to the user, will result in an *inconsistent* location transition for the user. However, users themselves may also concurrently log in from servers in different geographical locations, resulting in inconsistent transitions that occur legitimately.

Table 5.3 shows the number of location transition events we observed over the five months of data. Without bootstrapping and memory to learn inconsistent transitions already seen in

Transition Type	Consistent		Inconsistent	
	All	New	All	New
Events				
Local movement	12,153	3,641	2,338	325
City transition	3,742	1,678	2,952	475
Foreign transition	412	213	678	155
Users across events	(820)	(614)	(370)	(231)

Table 5.3: Break-down of location transition events occuring over five months. *All* reflects transition-event counts without any bootstrapping or memory; *New* refers to unique transitions observed after a bootstrapping of five days.

the past, we observe on the order of thousands of inconsistent location transitions for the five months of data. Investigating these, we find that these are primarily due to users accessing collaborator machines concurrently (as our earlier characterization of concurrent sessions found in Section 5.1). Based on this observation, we refine the feature to only flag *new* inconsistent location transitions. The idea behind this refinement is that a location transition that we have seen for a user in the past, even if inconsistent, is worth remembering, as this pattern is likely to repeat in future if it is a part of user's workflow. With bootstrapping using five days of user activity and counting only *new* inconsistent transitions, we still see on the order of hundreds of users that generate a total of 955 inconsistent location transition events across five months with the following breakdown: local movement (325), city transition (475), and foreign transition (155).

With such a high base rate of legitimate occurrence, this feature is hard to leverage in individuality. However, we observe that one can infer that a *new* inconsistent location transition is benign if it involves IP addresses / collaborator institutions that the enterprise trusts. In the next subsection, we explore various notions of trust for a login IP address.

5.2.2 IP Address Reputation

The premise of this feature is that logins from IP addresses for which the site has prior history of benign activity are with high likelihood trustworthy. However, there is also a small probability that an attacker secures access to a compromised machine corresponding to one of the trusted IP addresses. In this work, we do not have enough visibility through our datasets for detecting compromised hosts in external networks. Therefore, we assume that logins from trusted IP addresses always correspond to legitimate users. Below we explore two notions of IP address trust: (*i*) user-level (the set of IP addresses the site can trust for a given user), and (*ii*) network-level (the set of IP addresses the site can trust for all users).

Trust level	New events	Users
User-level inbound IP address trust	5,181	(757)
+ User-level inbound ASN trust	1,132	(518)
+ Network-level outbound IP / ASN trust	489	(235)

New network-level ASNs trusted per month	Jan	Feb	Mar	Apr	May
	78	30	21	14	11

Table 5.4: Breakdown of new untrusted IP address events with different variations of trust.

5.2.2.1 User-level Trust

For each user, we maintain a history of the IP addresses from which they have logged in the past. We also leverage LDAP authentications for users (where available) to build their IP address history. Using this history to assess trust for new logins, we observe ≈5K new untrusted IP address events over five months. Extending the trust from past IP addresses to past ASNs seen for the user, the number of new untrusted IP address events reduces from 5K to 1.1K.

5.2.2.2 Network-Level Trust

In addition to user-level trust for IP addresses/ASNs seen for a user in the past, we can leverage a notion of *network-level* trust. This is the set of IP addresses/ASNs that we can trust for the global user-base of the network.

To develop this set, we first note that the destinations of outbound `ssh` connections are trustworthy, since they are initiated by trusted users at the enterprise. Hence, if we have seen outbound `ssh` connection from the enterprise to an IP address, we can trust inbound connections from that IP address.

We can also discover collaborator institutions that can be trusted at a broader ASN level by leveraging the outbound connections. Towards this end, we compiled the set of ASNs for which we observed an outbound connection to at least two addresses. We discovered 154 such ASNs during the first five months of data. We characterized them, finding 96 universities/research centers (60%), 16 cloud/hosting services (such as Amazon and Microsoft), 6 residential/commercial ISPs, and 36 that do not fall into any of the categories above. We maintain IP-level trust for the ASNs that do not fall into the universities/research centers category, as they are too broad to be trusted at an ASN level.

Of the 154 ASNs, 51% appeared in the first month, and 70% in the first two months. Only 11 new ASNs appeared in the fifth month, suggesting that the set starts to converge over time. The appearance of a new ASN that is *potentially* trust-worthy is rare enough that it is feasible for an analyst to manually inspect new ASNs and decide whether to add them to the trusted set.

Total unique SSH client version strings	274
Total unique SSH client classes	15

SSH client strings history	New events	Users
User-view	320	(220)
+ Network-view (per IP address)	252	(179)

Table 5.5: Breakdown of new SSH client versions seen over a five month period with two variations of history: (*i*) user-view and (*ii*) network-view.

After incorporating network-level trust in addition to user-level trust for login IP addresses, the new untrusted IP address events reduce to 489 across the five months. (Table 5.4 summarizes the reduction in the number of untrusted IP address events as we add different notions of trust).

5.2.3 SSH Client Version History

In this subsection, we explore whether the SSH client version used in a remote login can serve as a second factor for establishing that the legitimacy of login. The premise of this feature is that if users generally use a stable set of SSH client versions, then a login from a new SSH client version should indicate a *rare* and *anomalous* event. However, users may legitimately log in with new client versions for three reasons: (*i*) the user upgrades the SSH client on their machine, (*ii*) the user logs in from a new device or a new SSH client on their device, or (*iii*) the user steps through a server which runs a client version different than those the user has used in the past.

We can detect upgrade events (and account for them as legitimate) by comparing the current client string to those already present in the user's history. To this end, we developed a parser that extracts the class and the major/minor version of the SSH client from the client version strings. For example, we parse the string '`SSH-2.0-WinSCP_release_4.3.7`' to the class `SSH-2.0-WinSCP`, the major version `4`, and minor version `3.7`. We built this parser using five months of data — over this period we observed 274 unique client versions that map to 15 unique classes. To detect *potential* client upgrades, we then check whether the user's history contains a client version string that has the same class as that of the current login, but a version number lower than that of the current login string.

To flag *new SSH client version events*, we first check whether the string is in the user history or corresponds to a potential upgrade. Over the five months, we observed 320 new SSH client version events that were not potential upgrades.

We observe that similar to the notion of network-level trust for IP addresses, we can incorporate information from the activity of other users in this feature as well. If the network has seen the ⟨`IP address, SSH client version`⟩ pair in the past, we do not flag the login as a new client version event, even if the version string has never been observed for the user before.

This reduces the new SSH client version events from 320 to 252, but the number of alarms are still too high for this feature to be a useful indicator of compromise on its own.

5.2.4 Whether the User is Traveling

Having insight into whether a user is traveling can help in understanding some of the inconsistent location transition / new untrusted IP address events. In the case of travel, we expect the user to log in from a new (and potentially untrusted) IP address. Logins during travel will also likely manifest inconsistent location transitions if the user steps through a machine at one of the collaborator institutions concurrently with a login from their current location.

We can infer whether a user is traveling based upon indication of logins from locations that require the physical presence of the user. Two such locations that indicate physical presence are logins from the enterprise wireless network and logins from airports/flights. The former confirm that the user is not traveling, whereas the airport/flight logins confirm otherwise. Note that since it is not always possible to determine the state of the user at a given time, this feature can have three values: `true`, `false`, and `undetermined`.

To detect airport logins, we collected a list of 41 airport/in-flight network blocks by performing web searches and by looking for the keyword *airport* in the ASN names from which we observed an inbound login in our dataset. For each login, we determine whether the user is traveling by checking if the login IP address is within the list of *known airports/in-flight addresses* — if it is, we mark the state of the user as *traveling*. We then propagate this state to future logins and end the travel window for a user when they provide a signal that they are back: a *consistent* transition to a location within driving distance of the enterprise network. We manually inspected the travel windows computed using this inference, and found it to be correct in all but three cases. Those involved collaborators based at institutions in other cities, for whom we observed an in-flight login, and incorrectly waited to see a login from a local location.

During the five months, we observed airport logins from a set of 35 users, corresponding to 48 distinct travel events (i.e., those appearing on different days). In most cases, the users log in from the home airport or from a flight: we observed 30 home airport events, 15 in-flight, and 3 logins from other airports.

5.3 Combining Features to Find Anomalous Logins

In the previous section, we explored different variations of features and illustrated that each one in individuality would result in an overwhelming workload for an analyst to investigate. In this section, we discuss the base-rates for the combinations of the features. Since some of the features leverage global network history (such as the set of ASNs trusted at network-level), we use a training window of two months to learn properties of activity seen on the network.

User State	Feature			Events
	New SSH client (131)	Untrusted IP address (310)	New inconsistent transition (599)	
Undetermined	✓	✓		6
	✓		✓	16
		✓	✓	22
	✓	✓	✓	1
Traveling	✓	✓		0
	✓		✓	0
		✓	✓	3
	✓	✓	✓	0
			Unique suspicious events	47

Table 5.6: Base rates of combinations of features with two months of global training and three months of test data. (Total number of logins assessed: 120,015).

Note that this training is independent of the per-user training window of *five* days. For each new user that arrives after the two month global training period, we still require a training window of five active days to learn user-specific properties (such as the set of SSH client version strings they use).

We explore the combination of three boolean history-/context-based features, flagging an alarm if both features in the pair true:

1. ⟨ **new untrusted IP address, new inconsistent location transition** ⟩: This is a strong indication of ununsual activity since an untrusted IP address was seen to be *concurrently* accessing the account with the user. Combining these two features rules out the possibility that the inconsistent transition occurred because the real user was accessing a server at an institution that they normally use in their workflow.

2. ⟨ **new untrusted IP address, new SSH client version** ⟩: This combination of features allows us to detect when the user logs in from an IP address/ASN that has no reputation on the network, and in addition the client version does not indicate that the actual user might be the one logging in.

3. ⟨ **new SSH client version, new inconsistent location transition** ⟩: Since our notion of IP address trust is quite broad (at ASN level), this combination of features allows us to detect events that come from trusted IP addresses, but involve an anomalous SSH client version and an inconsistent geo-location transition.

We expect that even these combinations of features can occur legitimately. Table 5.6 shows the base-rates for combinations of the features. In total, the combinations raise 47 unique

alarms over three months of test data, establishing that we can use these three simple rules to derive a manageable workload for a site analyst for further investigation. Next, we assess how the state of the user (if known) informs the decision making process. In our test data, the state of the user was known only for three out of the 47 alarms. All of these involved the user logging in from an in-flight IP address, and raised an alarm because in-flight IP addresses cannot be geo-located correctly, hence resulting in an (incorrect) inconsistent location transition. Since our test data lacks enough instances where the user's travel state is known, we use this feature to only aid an analyst, flagging an alarm as low priority if the user is known to be traveling, and high-priority if the user is not traveling (i.e., known to be at the enterprise wireless network).

5.4 Evaluation

In this section, we discuss the results of running our detector on seven months of data that were not part of the characterization phase.

We evaluate our detector in terms of *analyst workload* and the quality of the feed it generates, since we envision that the output of our system will be consumed by the operational security teams. An alarm flagged by our system provides an indication for analysts that an anomalous login that deviates from the user's normal workflow has occurred, and they should investigate it further, either by assessing the surrounding activity or by confirming with the user that the login was from them. We expect that most of the times these *rare* anomalous events correspond to the user themselves — for example, logins while a user is traveling will deviate from the user's normal logins in terms of the IP address they use for the login, and may also deviate in terms of SSH client if the user logs in from a device different than the ones they normally use. Further, although the base rate of occurrence of compromises is extremely low (once in many months), when they occur the attacker's logins can fall in the same space as that of *rare* events from legitimate users. Therefore, each *rare* login event should be investigated to establish whether it is from the user or not.

We only have partial ground truth for the dataset to evaluate our detector. This is due to the nature of the problem: the site lacks an existing solution for detecting compromised credentials. According to the site operators, the current practice involves a threshold-based system to detect brute-force break-ins. In addition, analysts at the site manually investigate daily foreign login alerts. However, this foreign login alert stream is generated after heavy whitelisting — countries where collaborator institutions are located result in a high number of foreign login alerts, and are therefore whitelisted by the site. The site's current detection practices recorded only one break-in for our test data (Jun–Dec 2015). Given the current state of detection at the site, our work is a significant step towards systematically analyzing the login activity on the network, and improving the analyst feed to the set of logins that seem the most anomalous.

To assess the alarms generated by our detector, we used a two-step approach. First, we

⟨ new untrusted IP address, new inconsistent location transition ⟩	53
user traveling	7
⟨ new untrusted IP address, new SSH client version ⟩	18
⟨ new SSH client version, new inconsistent location transition ⟩	20
⟨ new SSH client version, new untrusted IP address, new inconsistent location transition ⟩	1
Total suspicious logins	92
Unique suspicious logins	89

Table 5.7: Breakdown of the logins flagged by our detector according to the rule that results in the anomaly.

Local movement (2 mobile)	8
City transition (6 mobile)	38
Foreign transition (1 mobile)	17
Foreign country and local movement	7
Other city and local movement	3

Table 5.8: Transition-type breakdown for alarms involving an inconsistent location transition. *Mobile* indicates that one of the IP addresses in the transition belongs to a mobile network.

analyzed the surrounding activity (both past and future) of the user to see if we can construct an explanation for the anomaly in the dataset. For example, we can establish that a ⟨new untrusted IP address, new SSH client version⟩ event occurred because the user was traveling to a foreign country at the time of login — the travel may manifest as the user logging in only from the foreign country for a few days following the anomalous login (and no login activity from the enterprise network during the travel). For the alarms for which we could not find a plausible explanation within the dataset, we worked with the site security operators, who sent emails to inquire of the users about the logins that seemed suspicious.

In total, our detector raised 89 alarms for the seven months of test data, corresponding to a reasonable analyst workload of 2–3 alarms per week. Table 5.7 gives a breakdown of the alarms according to the rules that flagged the alarm. The majority of the alarms (53 out of 89) involved an untrusted IP address in conjunction with an inconsistent location transition. 20 alarms involved a new SSH client version in conjunction with an inconsistent location transition, and 18 involved a new untrusted IP address in conjunction with a new SSH client version. One alarm involved all three features. For the 73 alarms that involved an inconsistent location transition, Table 5.8 gives a breakdown of the type of transition involved.

We were readily able to establish a plausible explanation for sixteen of the alarms, as these involved the user logging in from a mobile network. Six of these alarms involved the user logging in with a new SSH client version from a new untrusted IP address. (The client version string in three of these cases indicated that the user was using an SSH client called 'JuiceSSH'

Time span	Jun 2015–Dec 2015
Number of alarms	89
Analyst workload	≈ 3 alarms per week
Number of compromises	1
Number of shared accounts	2
User indicated possible VPN use	2
Concurrent access from mobile network	16
User traveling	7

Table 5.9: Summary of the alarms generated by our *anomalous* logins detector.

which is specific to mobile devices [8], while in the other cases it was device independent). The other nine cases involving access from a mobile network raised an alarm because of inconsistent location transition in conjunction with an untrusted IP address. The inconsistent transition was likely flagged incorrectly in these cases because we are unable to get accurate geo-location for IP addresses belonging to mobile networks. While this category of login anomalies due to access from a mobile network can reflect logins from attackers as well, with good likelihood these logins were from actual users. (We did not check these with the site-operators, in order to limit analyst workload to only the most suspicious ones for which we could not find any plausible explanation).

Next, the travel-state of the user helped us establish the legitimacy of seven anomalous logins. All of these alarms involved the users logging in from an untrusted IP address and making an inconsistent location transition in doing so. Four of the cases involved logins from in-flight IP addresses and two involved logins from mobile networks close-in-time with logins from an airport.[4] In both the in-flight and mobile network access cases, the logins were highly likely legitimate and incorrectly resulted in an anomaly because of our inability to geo-locate the IP addresses involved. Similarly, in the last instance involving user travel, the alarm resulted from the user logging in from an IP address in the city they were traveling to, and we did not have accurate geo-location for the IP address (although it was neither an in-flight nor a mobile network IP address). We identified twelve more such false alarms where Maxmind's database did not have a correct geo-location for non-mobile IP addresses,[5] and thereby potentially resulted in incorrect geo-location transitions.

For most of the others, we looked at the surrounding and future activity of the login and could establish their legitimacy based on either collaborator access or indication of travel. (Our travel-state feature missed these because the travel did not include an airport/in-flight login). For ten of the anomalous logins, we could not find an explanation that could establish

[4]This makes sense given that some airports provide free wi-fi only for a limited time, and users may use their mobile devices as hotspots while at the airport.

[5]These manifest as Maxmind returning the geo-location lat/long with a precision of only one decimal place. A number of IP addresses geo-locate to (38.0, -97.0); Central Kansas.

the legitimacy of the login using the dataset. For these, the site operators contacted the users to inquire further about the activity in question. We sketch the findings on these below:

- In four of the cases, the user confirmed that the login was indeed from them while they were traveling. All of these travel cases involved login from a new untrusted IP address in conjunction with an inconsistent location transition (three foreign transitions and one city transition). All four alarms involved external-external transitions, i.e., both the IP addresses involved in the transition were external to the site, and therefore the site lacked visibility into the user connecting from one external machine to the other.

- In two of the cases, users indicated that they sometimes use a VPN/proxy to access geo-specific content. Both of these cases manifested as the user logging in from an untrusted IP address in conjunction with having an inconsistent *external-external* location transition. Further, in both the cases, the IP address that raised the alarm had a singular appearance, i.e., we never saw any other activity from the IP address in the complete dataset. One of the alarms involved the following activity pattern: lab $\rightarrow$ foreign-country $\rightarrow$ user's home network, with seven hours between the first transition (not enough to reach the foreign country), and a few minutes between the second transition. This fits with the scenario where the user was connected to the VPN from their home on the first login, but disconnected from it on the second login.

- One of the cases involved an unexplained *concurrent* access from a trusted IP address in China, but with a new SSH client version. The IP address was trusted because the site had observed an outbound connection to the IP address in the past. The inconsistent transition occurred because the user logged in from San Francisco two hours before the anomalous login (so this was an external-external transition). Subsequently, we observed login activity from a local collaborator institution after the anomalous login. Upon checking with the account user, we found that the account in question was a shared credential in use by a collaborator in China.

- In another peculiar instance involving a new SSH client and an inconsistent location transition, the user confirmed that it was them accessing a super-computer in a foreign country. We observed the following activity pattern for this case: the user logged in from the enterprise wireless network, then from an IP address in France about three hours later, and again from the enterprise wireless network roughly 20 minutes after the login from France. The site had ASN-level trust for the IP address in France (due to outbound connections to the ASN in the past). However, we did not observe an outbound connection to the IP address in France around the time of the anomalous login, even though the user was at the site at the time of login. It is possible that they either had an existing long-lived connection or connected to the super-computer on a port different than that the standard `ssh` port. The user also pointed out that the client version string contained the name of the super-computer, explaining the reason why the client version string was rather unique.

- Finally, one anomalous login (currently under investigation by the site) involved an inconsistent location transition involving a local movement across two ISPs, and a very unique timestamped client version that the user had never used before: `SSH-2.0-PuTTY_-Local:_`*`timestamp`*. This client version only appeared once for the user in the entire dataset.

Finally, our detector also caught the compromised credential in the site incident database. Our detector raised an alarm for this break-in based on the new SSH client version and new untrusted IP address rule.

5.5 Summary

In this chapter we developed a rule-based detector to flag suspicious remote access logins. Our approach is based on engineering features for which the combination rarely occurs in benign settings, yet can provide a strong indication of compromise. We used a data-driven approach to guide the development of these features, drawing upon one year of remote access logs from LBNL. Our detection approach is practical enough to provide a high quality feed to analysts, while staying within a budget of 2–3 alarms per week. The detector successfully caught a *known* compromised account, discovered an instance of (benign) shared credential use, and flagged one login where the user logged in via a suspicious anonymous proxy.

Chapter 6

Conclusion

In this dissertation, we looked at "needle in haystack" problems for detecting credential compromise in enterprise settings. Although incidents involving a successful compromise might strike a given enterprise only once in months, if they remain undetected they can prove hugely expensive to the sites. Our endeavour of developing practical, deployable detectors for identifying stealthy compromise led us to comprehensively characterizing the login activity at the enterprise network of the Lawrence Berkeley National Lab (LBNL). This heavily empirical undertaking is the first effort to illuminate various aspects of login activity at a medium-sized enterprise network. In particular, we developed approaches to identify both automated/misconfigured activity and "nested" activity, which are highly prevalent in enterprise settings. We also empirically modeled the dynamics of how individual users as well as groups of users fail to authenticate in a network setting. This comprehensive characterization forms the basis of our detectors.

We proposed a general framework for detecting distributed, potentially stealthy malicious activity at a site. The foundation of the method lies in detecting change in a *site-wide* parameter that summarizes *aggregate* activity at the site. The detector first employs the statistical technique of changepoint detection to identify the *epochs* during which this site-wide parameter has shifted—indicating that, in aggregate, the site's activity reflects the presence of problematic activity. The second step then classifies the hosts appearing during the detected epochs as either participants or non-participants in the activity, based on both individual past history and *coordination glue*, i.e., the degree to which a given host manifests patterns of probing similar to that of other hosts during the epoch.

We explored this approach in concrete terms in the context of detecting stealthy distributed SSH brute-forcing activity, showing that an apt side-wide parameter for this detection problem is the *aggregate login failure rate*, and that this parameter is well-described using a beta-binomial distribution. This model enables us tune the detector to trade off an expected level of false positives versus time-to-detection. In running our detector on eight years of data collected at LBNL, we discovered several large-scale stealthy campaigns for brute-forcing password-based credentials. The campaigns sometimes went on for several months, and hit

the Internet wholesale, but sometimes were also targeted. Given the resources and persistence that the attackers bring to the operation, and given that these attacks are a rather continuous on-going process, it is just a matter of time before a weakly chosen password by a careless user will be discovered by attackers, opening up an opportunity for further infiltration of the network. Indeed our data reflects this—users do sometimes choose weak passwords, enabling brute-forcers to occasionally succeed. This finding further emphasizes the need for moving towards stronger authentication mechanisms, in addition to deploying monitoring approaches that can identify such stealthy attacks.

Our work on detecting anomalous logins is the first step towards developing a framework for site analysts to understand the login activity in their networks, beyond the current practice of monitoring only for high rate brute-forcing activity and foreign logins from users (and that too after heavy whitelisting at the level of countries). Our approach starts with first structurally grouping nested logins into sessions, so that activity involving geographically distant servers that is indeed from the user themselves does not result in an unreasonably high number of alarms due to spatial-temporal violations. Through our exploration of different features, we illustrated both the detection problem's unique challenges and its unique opportunities: the outlier logins from legitimate users share the space with those from attackers who may have successfully compromised credentials, however the global view of the network provides opportunities to extract signals that can indicate why a given anomalous login might be legitimate. Using these signals, such as whether the IP address is trusted for the network and whether the user is traveling, our detection framework flags suspicious logins with an easy-to-understand explanation of why the given login attempt is anomalous.

Our detection approach is practical enough to provide a high quality feed to analysts, while staying within a budget of 2–3 alarms per week. In running the detector over seven months, we successfully caught a *known* compromised account, discovered an instance of shared credential use, and flagged a case where the user logged in via a suspicious anonymous proxy.

Future Work. In developing the detector for stealthy, distributed brute-force attacks, our approach was to first develop a *general* framework for detecting distributed attacks that are potentially stealthy in nature, and then tailor the framework to the specific problem of detecting stealthy SSH brute-forcing. This generic framework can be applied to other problems in the field of security, for which current detection approaches employ thresholds to flag that a given candidate attack source has exhibited a suspiciously high level of activity (e.g., when conducting scanning or DoS flooding).

Our work on detecting credential theft by looking for anomalous logins explores some preliminary features reflecting the properties of login attempts. Future work on this problem can engineer more features, which can be plugged into the existing framework. For example, one such feature would be to model session bytes and durations for a given user using one-class SVMs, in order to detect outlier sessions. Similarly, it would be interesting to combine existing work on identifying masqueraders using command-line activity with the login-based features we explored in this work.

Bibliography

[1] BlockHosts. http://www.aczoom.com/blockhosts/. ↑3, ↑7

[2] DenyHosts. http://denyhosts.sourceforge.net/. ↑3, ↑7

[3] Ebury SSH Rootkit - Frequently Asked Questions. https://www.cert-bund.de/ebury-faq. ↑2

[4] Fashion blogger Rozalia Russian targeted in Instagram hacking and extortion. http://www.smh.com.au/lifestyle/fashion/fashion-blogger-rozalia-russian-targeted-in-instagram-hacking-and-extortion-20160404-gnxm8g.html. ↑1

[5] Fox News's hacked Twitter feed declares Obama dead. https://www.theguardian.com/news/blog/2011/jul/04/fox-news-hacked-twitter-obama-dead. ↑1

[6] Google Distance Matrix API. https://developers.google.com/maps/documentation/distance-matrix/. ↑58

[7] Hackers breach Clinton campaign chairman's Twitter account. http://www.cnn.com/2016/10/12/politics/john-podesta-twitter-hack-hillary-clinton. ↑1

[8] JuiceSSH. https://juicessh.com. ↑66

[9] Maxmind. https://www.maxmind.com/. ↑58

[10] Protocol Analyzers — Bro 2.4.1 documentation. https://www.bro.org/sphinx/script-reference/proto-analyzers.html. ↑15

[11] Spam through compromised passwords: can it be stopped? https://www.spamhaus.org/news/article/681/spam-through-compromised-passwords-can-it-be-stopped. ↑1

[12] sshguard. http://www.sshguard.net/. ↑3, ↑7

[13] Statement by OPM press secretary Sam Schumach on background investigations incident. https://www.opm.gov/news/releases/2015/09/cyber-statement-923. ↑1

[14] Stranded in London — Be careful. Your friend isn't really there and wasn't robbed at gunpoint. `http://voices.washingtonpost.com/securityfix/2007/09/your_money_or_your_email.html`. ↑1

[15] The Bro Network Security Monitor. `https://www.bro.org`. ↑15

[16] The Hail Mary Cloud Data - Data collected by Peter N. M. Hansteen (peter@bsdly.net). `http://www.bsdly.net/~peter/hailmary/`. ↑19, ↑22

[17] Verizon's 2016 Data Breach Investigations Report (DBIR). `www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf`. ↑1

[18] ICS-ALERT-12-034-01 — SSH scanning activity targets control systems. `http://www.us-cert.gov/control_systems/pdf/ICS-ALERT-12-034-01.pdf`, Feburary, 2012. ↑27

[19] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair. A comparison of machine learning techniques for phishing detection. In *Proceedings of the Anti-phishing Working Groups 2nd Annual eCrime Researchers Summit*, 2007. ↑3

[20] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krügel, and E. Kirda. Scalable, behavior-based malware clustering. In *Proceedings of the 16th Network and Distributed System Security Symposium, NDSS*, 2009. ↑3

[21] A. Bergholz, G. Paaß, F. Reichartz, S. Strobel, and S. Birlinghoven. Improved phishing detection using model-based features. In *Fifth Conference on Email and Anti-Spam, CEAS*, 2008. ↑3

[22] R. Bezut and V. Bernet-Rollande. Study of Dictionary Attacks on SSH. Technical report, University of Technology of Compiegne, `http://files.xdec.net/TX_EN_Bezut_Bernet-Rollande_BruteForce_SSH.pdf`, 2010. ↑8

[23] D. Brook and D. A. Evans. An approach to the probability distribution of CUSUM run length. In *Biometrika*, volume 59, pages 539–549, 1972. ↑31

[24] M. Chandrasekaran, K. Narayanan, and S. Upadhyaya. Phishing email detection based on structural properties. In *NYS Cyber Security Conference*, 2006. ↑3

[25] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. Semantics-aware malware detection. In *Proceedings of the 26th IEEE Symposium on Security and Privacy*, 2005. ↑3

[26] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna. Compa: Detecting compromised accounts on social networks. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium, NDSS*, 2013. ↑1, ↑3, ↑11

[27] I. Fette, N. Sadeh, and A. Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th International Conference on World Wide Web*, 2007. ↑3

[28] D. M. Freeman, S. Jain, M. Durmuth, B. Biggio, and G. Giacinto. A statistical approach to measuring user authenticity. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium, NDSS*, 2016. ↑4, ↑11

[29] S. Garera, N. Provos, M. Chew, and A. D. Rubin. A framework for detection and measurement of phishing attacks. In *Proceedings of the 5th ACM Workshop on Recurring Malcode*, 2007. ↑3

[30] Carrie Gates. Coordinated scan detection. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium*, 2009. ↑8

[31] D. Gerzo. BruteForceBlocker. `http://danger.rulez.sk/projects/bruteforceblocker`. ↑3, ↑7

[32] K. Gold, B. Priest, and K. M. Carter. An expectation maximization approach to detecting compromised remote access accounts. In *Proceedings of the 26th International FLAIRS Conference Cyber Security Track*, 2013. ↑4, ↑10

[33] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, 2008. ↑3

[34] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium*, 2007. ↑3

[35] D. M. Hawkins and D. H. Olwell. Cumulative sum charts and charting for quality improvement. Springer, 1998. ↑29, ↑30, ↑31, ↑38

[36] L. Hellemons. Flow-based detection of ssh intrusion attempts. In *16th Twente Student Conference on IT*. University of Twente, 2012. ↑3, ↑8

[37] C. Jacquier. Fail2Ban. `http://www.fail2ban.org`. ↑3, ↑7

[38] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *Proceedings of the 18th USENIX Security Symposium*, 2009. ↑3

[39] Brian Krebs. Email attack on vendor set up breach at target. `http://krebsonsecurity.com/2014/02/email-attack-on-vendor-set-up-breach-at-target/`. ↑1

[40] Brian Krebs. Hacked inboxes lead to bank fraud. `http://krebsonsecurity.com/2012/03/hacked-inboxes-lead-to-bank-fraud/`. ↑1

[41] Brian Krebs. Your money or your e-mail. `http://www.aarp.org/money/scams-fraud/info-07-2010/scam_alert_stranded_in_london.html`. ↑1

[42] M. Kumagai, Y. Musashi, D. Arturo, L. Romana, K. Takemori, S. Kubota, and K. Sugitani. Ssh dictionary attack and dns reverse resolution traffic in campus network. In *3rd International Conference on Intelligent Networks and Intelligent Systems*, 2010. ↑3, ↑7

[43] E. L. Malecot, Y. Hori, K. Sakurai, J. Ryou, and H. Lee. (visually) tracking distributed ssh bruteforce attacks? In *3rd International Joint Workshop on Information Security and Its Applications*, 2008. ↑8

[44] R. A. Maxion. Masquerade detection using enriched command lines. In *Proceedings of the International Conference on Dependable Systems and Networks, DSN*, 2003. ↑9

[45] R. A. Maxion and T. N. Townsend. Masquerade detection using truncated command lines. In *Proceedings of the International Conference on Dependable Systems and Networks, DSN*, 2002. ↑9

[46] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2005. ↑3

[47] J. Owens and J. Matthews. A study of passwords and methods used in brute-force SSH attacks. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET*, 2008. ↑8

[48] A. Pecchia, A. Sharma, Z. Kalbarczyk, D. Cotroneo, and R. K. Iyer. Identifying compromised users in shared computing infrastructures: A data-driven bayesian network approach. In *30th IEEE Symposium on Reliable Distributed Systems, SRDS*, 2011. ↑1, ↑3, ↑4, ↑10

[49] P. Prakash, M. Kumar, R. R. Kompella, and M. Gupta. Phishnet: predictive blacklisting to detect phishing attacks. In *Proceedings of INFOCOM*, 2010. ↑3

[50] M. Schonlau, W. DuMouchel, W. Ju, A. F. Karr, M. Theusan, and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 16(1):58–74, 2001. ↑9

[51] A. V. Siris and F. Papagalou. Application of anomaly detection algorithms for detecting SYN flooding attacks. In *Proceedings of the Global Telecommunications Conference GLOBECOM*, 2004. ↑8

[52] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. In *Proceedings of the 7th ACM Conference on Computer and Communications Security, CCS*, 2000. ↑8

[53] K. Thomas, F. Li, C. Grier, and V. Paxson. Consequences of connectivity: Characterizing account hijacking on twitter. In *Proceedings of the 21st ACM Conference on Computer and Communications Security, CCS*, 2014. ↑12

[54] J. Vykopal, T. Plesnik, and P. Minarik. Network-based Dictionary Attack Detection. In *International Conference on Future Networks*, 2009. ↑3, ↑7

[55] H. Wang, D. Zhang, and Shin K. Detecting SYN flooding attacks. In *Proceedings of IEEE INFOCOM*, 2002. ↑8

[56] K. Wang and S. J. Stolfo. One-class training for masquerade detection. In *Proceedings of the 3rd ICDM Workshop on Data Mining for Computer Security, DMSEC*, 2003. ↑3, ↑9

[57] C. M. Zhang and V. Paxson. Detecting and analyzing automated activity on twitter. In *Proceedings of the 12th International Conference on Passive and Active Measurement, PAM*, 2011. ↑35

[58] J. Zhang, R. Berthier, W. Rhee, M. Bailey, P. P. Pal, F. Jahanian, and W. H. Sanders. Safeguarding academic accounts and resources with the university credential abuse auditing system. In *Proceedings of the International Conference on Dependable Systems and Networks, DSN*, 2012. ↑4, ↑10