UNIVERSITY OF CALIFORNIA SAN DIEGO

Addressing Device Compromise from the Perspective of Large Organizations

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

in

Computer Science (Computer Engineering)

by

Louis Floyd DeKoven

Committee in charge:

Professor Stefan Savage, Co-Chair
Professor Geoffrey M. Voelker, Co-Chair
Professor Kirill Levchenko
Professor Ramesh R. Rao
Professor Alex Snoeren

2019

The Dissertation of Louis Floyd DeKoven is approved and it is acceptable in quality and form for publication on microfilm and electronically:

Co-Chair

Co-Chair

University of California San Diego

2019

DEDICATION

To my parents:

Beverly and Benjamin

and to my family:

Florence, Melissa, Chris, Ezra, and, Leron

EPIGRAPH

The important thing is to not stop questioning.

Curiosity has its own reason for existing.

Albert Einstein

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisers: Professors Stefan Savage, and Geoffrey M. Voelker. I feel so incredibly privileged to have had their support and guidance throughout my time in graduate school. Under their guidance, I have learned how to conduct and present research. Their support for a highly collaborative community within computer science, and broad research interests has made my graduate school experience one that I will always cherish.

I would like to thank Professors Alex Snoeren, Kirill Levchenko, and Ramesh Rao for being on my doctoral committee and being available whenever I needed any help.

My colleagues at UCSD and CESR made my six years of graduate school a fun and memorable experience. I would like to thank: Brown Farinholt, Sen Zhang, Neha Chachra, Brian Johannesmeyer, Vector Li, Gautam Akiware, Sunjay Cauligi, Ariana Mirian, Gary Soeller, Audrey Randall, Stew Grant, Anil Yelam, Ansel Blume, Nadah Feteih, Nishant Bhaskar, Shu-Ting Wang, Rob McGuiness, John Sarracino, Steven Hill, Rajdeep Das, Shelby Thomas, Liz Izhikevich, Danny Huang, Tristan Halvorson, Joe DeBlasio, David Kohlbrenner, Paul Pearce, Frank Li, Grant Ho, Damon McCoy, Alexandros Kapravelos, Karyn Benson, David Wang, Jake Maskiewicz, Edward Sullivan, Benjamin Braun, Ian Foster, and Nima Nikzad.

I would like to thank the systems and networking research staff, and UCSD Information Technology Services. Cindy Moore and Brian Kantor helped me configure and harden servers, and experiment with Hadoop cluster deployments. Jennifer Folkestad provided me with continued support. Coop Nelson, Nick Colias, Jim Madden, and Michael Corn went beyond their responsibilities to enable my research by providing support for our passive network monitoring infrastructure.

Throughout my time in graduate school, I collaborated with Facebook and Instagram security to conduct research. I would like to thank those who made this possible, as well a fun experience. In particular: Nektarios Leontiadis, Trevor Pottinger, Mark Hammell, Chad Greene, Gregg Stefancik, Matt Henley, Jenn Lesser Henley, Kyle Barry, Eoghan McKee, Jesse Kornblum,

Mark Vilanova, Lauren Berger, Meagan Kruman, Brandon Dixon, and Matt Richard.

Additionally, I would like to thank to Professors: Lawrence Saul, Aaron Schulman, and Mohan Paturi. Lawrence helped me understand and apply statistical learning. Aaron is incredibly fun to work with, and helped with our understanding of networked devices. Lastly, during my first year of graduate school Mohan took time beyond what I expected to help me learn.

Graduate school is a long process, and challenging at times. I am grateful for the love and support of my friends and family. There are no words to describe how much you've helped me. *Thank you for enabling me to accomplish my goals*.

Chapter 1, in part, is a reprint of the material as it appears in *Proceedings of Workshop on Cyber Security Experimentation and Test (CSET)*. Louis F. DeKoven, Stefan Savage, Geoffrey M. Voelker, Nektarios Leontiadis, 2017. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in part, is a reprint of the material as it appears in *Proceedings of the ACM Internet Measurement Conference (IMC)*. Louis F. DeKoven, Trevor Pottinger, Stefan Savage, Geoffrey M. Voelker, Nektarios Leontiadis, 2018. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, has been submitted for publication of the material as it may appear in *Proceedings of the ACM Internet Measurement Conference (IMC)*. Louis F. DeKoven, Audrey Randall, Ariana Mirian, Gautam Akiwate, Ansel Blume, Lawrence K. Saul, Aaron Schulman, Geoffrey M. Voelker, Stefan Savage, 2019. The dissertation author was the primary investigator and author of this paper.

VITA

2013	Bachelor of Science in Computer Engineering, California State University, Chico
2013–2019	Research Assistant, University of California, San Diego
2015	Master of Science in Computer Science, University of California, San Diego
2019	Doctor of Philosophy in Computer Science (Computer Engineering), University of California, San Diego

ABSTRACT OF THE DISSERTATION

Addressing Device Compromise from the Perspective of Large Organizations

by

Louis Floyd DeKoven

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California San Diego, 2019

Professor Stefan Savage, Co-Chair
Professor Geoffrey M. Voelker, Co-Chair

Addressing compromised device is a problem for virtually all large organizations. Compromised devices can propagate malware resulting in theft of computing resources, loss of sensitive data, and extortion of money. Unfortunately, large organizations do not have an oracle into device compromise. Instead, organizations must address compromise without straightforward answers to critical questions such as: “Is this device compromised?”, “Why/How is this device compromised?”, “What’s the best intervention?”. This problem, in part, results from limited observational vantage points, differences in intervention capabilities, and evolving adversaries with differing incentives. In this dissertation, I develop systems that empirically

address multiple types of device compromise using large-scale observations within different organizations, thus placing us on a stronger footing to devise better interventions.

I first describe an approach used at Facebook for detecting malicious browsers extensions. I present a methodology whereby users exhibiting suspicious online behaviors are scanned (with permission) to identify extensions in their browsers, and those extensions are in turn labeled based on the threat indicators they contain. Employing this methodology at Facebook I identify more than 1,700 lexically distinct malicious extensions, and use this labeling to drive user device clean-up efforts as well notify browser vendors.

Next, I examine for-profit services offering to artificially manipulate a user's social standing on Instagram. I identify the techniques used by these services to drive social actions, detail how they are structured to evade straightforward detection, and characterize the dynamics of their customer base. Finally, I construct controlled experiments to disrupt these services and analyze how different approaches to intervention can drive different reactions, thus providing distinct trade-offs for defenders.

Lastly, I describe a large-scale measurement of 15,000 laptop and desktop devices on a university's network to characterize the prevalence of security "best practices" and security-relevant behaviors, and quantify how they relate to device compromise. I use passive network traffic analysis techniques to infer a broad range of device features and per-machine compromise state. I find a number of behaviors positively correlate with host compromise, and few "best practices" exhibit negative correlations that would support their value in improving end user security.

Introduction

Large organizations connect a multitude of devices, and house sensitive information. Unsurprisingly, they also attract unwanted malicious and abusive actions originating from compromised devices using their platforms. Some of these actions result in the propagation of malware [10, 9, 18, 55], theft of computing resources [59, 73], and extortion of money [15]. Indeed, large organizations dedicate significant effort to preventing many types of compromise from impacting their platforms.

Unfortunately, large organizations do not have an oracle into device compromise. Instead, in many cases organizations must address compromise without straightforward answers to vitally important questions: Is a device compromised? Why/How is device a compromised? What's the best intervention?, etc. This problem, in part, results from limited observations available at each organization's vantage point (e.g., actions within a social network, encrypted network traffic connections, etc.), the types of interventions at an organization's disposal, and continuously evolving adversaries with differing incentives. In some cases, traditional countermeasures such as blacklisting or rate limiting are sufficient to address compromise. However, a motivated adversary may be able to subvert these interventions. Deriving an empirical understanding around device compromise is critical for developing long-lasting and effective interventions. Specifically, the prevalence of differing types of compromise coupled with the constrained vantage points and intervention capabilities of large organizations creates a need to develop empirically-grounded systems, that in-turn, can help drive intervention efforts.

In this thesis, I address the challenges inherent in developing systems aiming to remediate compromised devices engaging with large organizations. In particular, I demonstrate multiple

approaches for developing systems to address different types of device compromise using large-scale observations within organizations, thereby placing us on a stronger footing to devise better interventions. I describe multiple real-world and large-scale studies where I develop empirically-driven systems at three large organizations. In each system, the organization's vantage point, intervention techniques, and types of compromise differ. Specifically, I develop systems to: detect and cleanup browser malware, understand and disrupt abusive underground services selling inauthentic actions on online social networks, and explore how a range of recommended security practices (e.g., using antivirus, etc.) and behaviors correlate with ground truth security outcomes.

In the first study, I develop a system to detect and mitigate malicious browser extensions (MBE) impacting the online social network Facebook (Chapter 1). This work is motivated by previous studies that develop techniques to detect MBEs, and find that online social networks are a commonly targeted by MBEs. In 2014, Kapravelos et al. [50] develop a system that detects MBEs using dynamic analysis to study an extension's behavior when accessing honey Web pages that try and trick browser extensions into exhibiting malicious behavior. Similarly, in 2015, Jagpal et al. [47] describe a system used at Google to quickly detect MBE uploaded to the Chrome Web Store (an extension marketplace) by periodically monitoring the static and dynamic behaviors of each extension. In both related studies, the systems process browser extensions directly and evaluate each extension on a periodic basis.

From the perspective of an online social network the extensions themselves are not directly accessible. Additionally, how or why the extension is installed is unknown. Instead, malicious behaviors from browser extensions targeting online social network are observable. By leveraging these malicious behaviors first-hand, I drive detection and cleanup efforts of MBEs. I first discuss some examples of how Facebook can identify user accounts that are likely compromised. Next, I describe the workings of a custom malware scanner that uses static analysis to process browser extensions. From the processed extensions, I develop an automated MBE labeling system that uses malicious indicators (e.g., URLs, IP addresses, etc.) identified

by Facebook to classify MBEs. Deploying this system at Facebook over six weeks, I identify more than 1,700 new lexically distinct MBEs, show that existing anti-malware and anti-abuse mechanisms offer limited effectiveness against the MBEs, and notify anti-malware and browser vendors.

In the second study, I develop a system for the online social network Instagram to detect and disrupt underground markets offering to artificially inflate an Instagram account's social status (Chapter 2). Users with more followers are able to reach larger audiences with their posts and thus can be seen as carrying more "weight" in some abstract social hierarchy. Since this standing is directly monetizable via advertising, it is unsurprising that this aspect of social media has attracted organized abuse. These underground services have evolved adversarially as a result of prior interventions and the demand for more "real looking" actions: from producing inauthentic actions by way of fake or bot accounts, to utilizing more sophisticated techniques involving real Instagram accounts. Similar to my work on MBEs, the actions from abusive services can be observed directly by an online social network's vantage point. However, unlike browser extensions the malicious software is not required to run directly on each device making the use of a malware scanner-like approach impractical.

I describe two techniques used by popular underground services to artificially inflate the social status of an Instagram user's account: collusion networks and reciprocity abuse. I then develop a honeypot account framework for Instagram that allows me to engage with abusive services as a customer. I design the honeypot account framework in a way that enables the systematic creation of both fake and real looking honeypot accounts. I use the honeypot account framework to attribute abusive actions to each underground service, to evaluate how services use their customer's Instagram accounts, and to quantify the natural effects of reciprocation that are exploited by reciprocity abuse services. Next, by using a set of signals produced by Instagram I am able to identify the broader set of actions produced by each service. This rich data set allows me to estimate each service's customer population size, customer stability, monthly gross revenue, the types of service that are most popular among customers, and biases in how

reciprocity abuse services operate. Finally, using my understanding of how these service operate I develop two controlled interventions to compare the effectiveness of different countermeasures (e.g., blocking actions, etc.). The first explores how a small fraction of service customers react to different countermeasure techniques, and the second measures the service-wide reaction to different countermeasure techniques when applied to a large fraction of abusive actions. I find that underground services are able to attract a large clientele, and generate over $1M in monthly revenue. Additionally, different approaches to intervention (i.e., transparent interventions such as blocking abusive services vs. more opaque approaches such as deferred removal of artificial actions) can drive different reactions and thus provide distinct trade-offs for defenders.

In the third and final study, I develop a system that passively monitors a university's residential network to explore how a range of security "best practices" (e.g., using antivirus, updating software quickly, etc.) and behaviors (e.g., the types of Websites visited, etc.) correlate to ground truth security outcomes (Chapter 3). Security is a discipline that places significant expectations on lay users. In fact, there are a wide array of technologies and behaviors that end users are expected to adopt and thereby reduce their security risk. However, the adoption of these "best practices" — ranging from the use of antivirus products to actively keeping software updated — is not well understood, nor is their practical impact on security risk well-established.

I describe the architecture and implementation of a large-scale passive network monitoring system that produces per-device models measuring a range of security practice and behavioral features. Unlike the vantage point of online social networks which is constrained to social network actions, passive network traffic contains a broader set of device actions (e.g., software updates, social network activity, etc.). However, this activity is encoded within network traffic, and fine-grained detail into the context of each action is commonly obscured by encryption. Consequently, network traffic requires considerable processing to identify device features. To address these challenges, I develop network traffic signatures that enable the detection of a range of security practices and behaviors. I then use operational security logs and network intrusion detection system (IDS) alerts to identify compromised machines on the network. Combining this

information, I investigate how recommended security practices and behaviors correlate to device compromise. Lastly, I describe a logistic model that compares device features in terms of their ability to predict security outcomes relative to one another. Analyzing months of longitudinal data I find that a number of recommended security "best practices" are followed, however, they do not negatively correlate with device compromise. Most positively correlated is the type of web sites a device visits (e.g., adult content, video games, etc.), and the volume of traffic devices produce. Subsequently, using a logistic model I find that behavioral features such as visiting web sites related to gaming and illegal content are relatively more useful for distinguishing compromised devices.

In this dissertation, I demonstrate multiple approaches to develop empirically-grounded systems that address device compromise within different organizations. I present solutions that take advantage of analytic data to determine: what is measurable under the limitations in each organization's vantage point, as well as the trade-offs across different types of intervention. This dissertation is organized as follows. In Chapter 1, I develop a system for an online social network to detect and cleanup MBE. In Chapter 2, I develop a system for an online social network to understand and disrupt underground markets selling inauthentic actions. In Chapter 3, I develop a system for an enterprise network to explore how recommended security practices and behaviors correlate to ground truth security outcomes. Finally, I conclude in Chapter 4.

Chapter 1

Malicious Browser Extensions at Scale

In this chapter, we develop an automated framework for Facebook that identifies and mitigates malicious browser extensions (MBE) targeting the online social network. Online social networks are not in a position to easily detect and remove known MBEs from infected devices. From the vantage point of an online social network, malicious activity can be directly experienced. However, the extensions themselves are not readily available. We overcome these obstacles by developing a custom malware scanner that labels extensions using threat indicators previously associated with abusive behavior. We describe the scanner's methodology for propagating malicious indicators across browser extensions, and present an evaluation of the system when deployed at Facebook over a few weeks. Our system quickly detects and mitigates new MBEs. We also find that existing anti-abuse and anti-malware offer limited effectiveness against MBEs.

1.1 Introduction

Today, Web browsers encapsulate dynamic code, interact with users and are implicated in virtually every activity performed on computers from e-mail to game playing. While some of these activities have been made possible by enhancements to the standard languages and capabilities supported by the browsers themselves, many others are made possible via browser extensions designed to augment this baseline functionality.

Notably, browser extensions enable customization not only with respect to visual appearance (e.g., by changing the look and feel of the browser), but also on a deep behavioral level (i.e., in the way the browser interacts with Web sites). Browsers enable the latter functionality by allowing extensions to use a set of permissions that Web sites do not normally have. For example, extensions are capable of modifying HTTP headers, bypassing the Content Security Policy (CSP) [96] set by Web site owners and hiding the results of any actions by rewriting Web site content before it is displayed. These capabilities allow extensions to offer complex and rich modifications to the user experience and support the implementation of services that would otherwise be impossible to implement. However, these same capabilities can provide a powerful vehicle for performing malicious attacks [20]. Unsurprisingly, this problem has evolved from one of abstract potential into a concrete threat, and today the problem of MBEs is widely understood to be real and growing [81, 20, 58, 50, 47]. We consider MBEs to be extensions that take actions on behalf of a user without their consent, or replace Facebook's key functionality or content.

Unfortunately, detecting MBEs is challenging because the malicious nature of a given extension can manifest dynamically and the online targets of its abuse have no natural way to attribute those behaviors back to particular extensions. More concretely, while a browser vendor or extension marketplace is in a position to inspect extension code, inferring malicious intent may not be possible from that vantage point. In addition to the traditional challenges with such code analysis approaches (e.g., polymorphic encoding), extensions routinely fetch resources from third-party sites and, as a result, an extension may only exhibit malicious actions at certain times or when certain Web services are visited. Conversely, from the vantage point of a targeted Web service, abusive actions may be clear, but the source of those actions can be murky. Extensions frequently hide by emulating a normal user's interactions and there are no standard mechanisms to link browser actions back to a particular extension (or even to enumerate the extensions present on a user's browser). Indeed, because the viewpoint of the Web service provider is limited to the Document Object Model (DOM) there is no shared language by which they can crisply share threat intelligence with browser vendors or extension marketplaces.

In this paper, we have started to bridge this gap between the Web site and browser vantage points, which we believe will enable more effective interventions against the threat of MBEs. In particular, we examine MBEs from the perspective of Facebook — which, among others, is extensively targeted by such extensions (e.g. [47]). We describe our approach for automatically collecting browser extensions of interest, detecting the malicious ones (within seconds of reaching Facebook's infrastructure) and then working to remove such extensions from our customer ecosystem. In particular, this paper describes the following contributions:

- A methodology for collecting browser extensions from devices suspected of malware compromise.
- A methodology for automated labeling of malicious extensions using indicators we extract from the collected samples and threat indicators previously associated with abusive behavior.
- Deploying this methodology at Facebook, we identify more than 1 700 malicious Chrome and Firefox extensions[1] out of a total of more than 34 000 scanned extensions during a 6-week period spanning late 2016 into 2017.
- We show that existing anti-malware and anti-abuse mechanisms only offer limited effectiveness against MBEs, addressing a small fraction of the samples we detect (and with far greater delay when they do).

The remainder of this paper is organized as follows: Section 1.2 provides an overview of browser extensions and related work; Section 1.3 outlines Facebook's approach for detecting compromised user accounts, and collecting and analyzing browser extensions; Section 1.4 describes the automated extension labeling system, and Section 1.5 evaluates it, characterizing the volume of extensions Facebook deals with and system behavior over time; Section 1.6 motivates the need for the new labeling system; and Section 1.7 concludes.

[1] While most recent browsers support extensions (e.g. Safari, Opera, Internet Explorer, etc.), we focus on Chrome and Firefox since visitors using these two browsers constitute 54% and 13% of browser traffic respectively seen per day at Facebook.

1.2 Background

Browser authors have tried to provide as dynamic and programmable experience as possible, but inevitably some applications have required capabilities beyond those provided via standard Web programming interfaces. To address this need, virtually all major browsers support extension interfaces. Extensions written to these interfaces are allowed to execute code, interact with the browser core and initiate network calls — all independent of particular Web pages being viewed. While extensions can use a variety of technologies and languages, for this paper we will be focusing on HTML and JavaScript (JS) which are used predominantly in the development of Chrome and Firefox extensions.

Because browser extensions are given permission to interact with the browser in manners that would otherwise be classified as "high-risk" [56], there is a range of opportunities to enable malicious behavior. For example, extensions can violate typical *cross-site request forgery* (CSRF) or *cross-site scripting* (XSS) protections, inject arbitrary code in a page's DOM, rewrite its content, and access Web traffic as a page is being loaded (including all cookies and POST parameters). Indeed, the permissions available are sufficiently powerful that they can even prevent the user from removing an extension once loaded.

Users frequently load browser extensions via online marketplaces (e.g., the Chrome Web store), which try to vet both code and authors, and remove extensions that are clearly abusive. However, browsers also allow a range of alternate "sideloading" options including manual installation, operating system administrative policies, and native binaries. While browser vendors are actively reducing such sideloading opportunities, attackers have shown great creativity in bypassing ad hoc limits. Moreover, even when the browser is configured to prevent sideloading, we have observed one class of malware (BePush) enabling sideloading by simply installing an older version of the browser that lacks such protections.

These issues have been understood for some time, with Dhawan and Ganapathy identifying early malicious extensions in 2009 and proposing techniques to protect against them [20].

Researchers have shown similar problems exist in modern browsers [56, 27, 90, 13] and large-scale empirical measurements [50] and operational experience [47] show that malicious browser extensions are a widespread problem. Much of the existing research in this space has focused on how to either better harden the browser [58, 21, 56, 34] or to provide a better mechanism for vetting code in extension marketplaces [4].

Our work builds on ideas from these prior efforts. Our approach is data driven, like the work of Jagpal et al. [47] and Kapravelos et al. [50], but is based on static analysis using Facebook's own contemporaneous threat indicator data (e.g., abusive domains / URLs) to label extensions. This allows our approach to be browser-agnostic and adapt quickly to changes in the kinds of abuse being perpetrated on Facebook. Of course, Rice's theorem says there is no way to figure out whether a piece of code will be malicious, so there is no way to make a promise to catch every MBE. We further describe a soup-to-nuts operational workflow — including how we obtain samples, process and label them, and remediate affected users.

1.3 Collecting Browser Malware

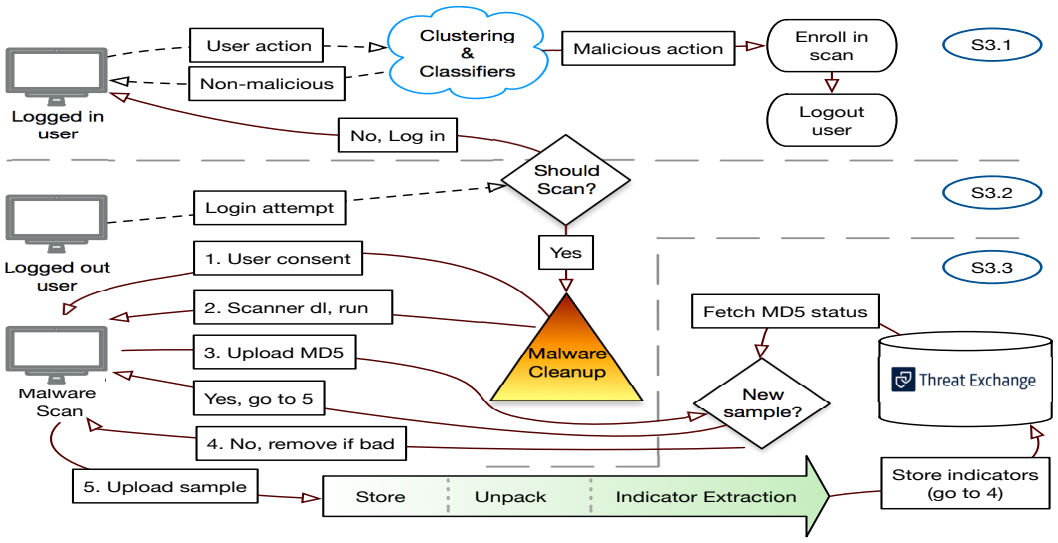

Figure 1.1. An overview of our system highlighting the detection, malware scanner, and static analysis steps. The dashed arrows describe normal user interaction, and solid arrows are transitions within the described system.

Facebook has collected more than *1 700* unique malicious samples over the 6-week analysis period.[2] Naturally, manual analysis of extensions at this scale is infeasible, so Facebook has developed new techniques to automate the collection and analysis of samples.

In this section we describe some of the ways Facebook detects malware-compromised user accounts. We further outline what happens after detecting such accounts and, specifically, how we collect and analyze the responsible malware samples from malware victims' computers via a custom malware scanner. Finally, we report on the initial analysis we perform on the collected extensions.

1.3.1 Detecting Compromised User Accounts

The process of acquiring new malware samples starts by detecting user accounts suspected of being compromised with malware. At a high level, this process is guided by the clustering and classification systems (shown in Figure 1.1) using as input (i) signals of abnormal activity, (ii) client-side third-party injected code in Facebook's DOM, (iii) and user-reported objectionable content. While the detailed process of detecting malware-compromised accounts is mainly beyond the scope of this work, in the following paragraphs we present some examples of related signals.

Negative Feedback. In the event a user account is compromised with malware, the malware may attempt to either use the compromised account for monetization — e.g. by posting links that redirect to ad-filled pages — or to spread the infection by posting links to malware. The latter usually happens via *clickbait*. In either case, Facebook users have the ability to report the content as objectionable (e.g. abusive, malicious, etc.), and links to such content may eventually get blacklisted.

Spiking Content. Facebook's real-time abuse detection systems monitor the time series of user activity to detect anomalies based on diurnal patterns of normal activity. Such anomalies fall into two high-level categories: anomalies that can be remediated automatically, and ones that

[2]Uniqueness is based on MD5 hashes of extension contents.

need an analyst to examine and take action.

An example of the latter case would be auto-generated objectionable content being shared on Facebook (e.g. adult content) with similar characteristics, e.g. directing viewers to the same external domain that results in a drive-by malware infection. In such a case, the analyst would typically add the related domains to a blacklist and enqueue the users participating in such activity into a malware cleanup flow. Anomalies that can be auto-remediated are either simple anomalies that make use of other high quality signals (e.g. spiking negative feedback) or anomalies that have been previously seen and the responses are already codified.

DOM-based indicators. Facebook uses client-side code to self-inspect its own rendered DOM for injected third-party code. One challenge with client-side code is that a MBE may attempt to prevent such code from running. In the event third-party code is discovered, code-specific features are analyzed by Facebook's clustering and classification systems. When features related to malware are identified, users' devices containing such features may get enrolled into a malware cleanup flow.

1.3.2 Malware Scanner and Cleanup

Once Facebook identifies an account suspected of having been compromised by malware, the account may be enrolled in a process that is capable of detecting and remediating malware via an online scan session.[3] This process is shown in Figure 1.1 under "Malware Cleanup". Following user consent (see Figure 1.2), the user downloads a one-time malware scanner that runs on the potentially compromised system. If user consent is not provided, the user can continue to access their account using other devices. After a cool-down period the potentially-infected device is allowed to access Facebook again.

Once the scanner process starts, it inspects locations on the file system known to hold Firefox and Chrome extensions. For each observed sample, the scanner communicates the file hash with Facebook's infrastructure, which in turn provides a verdict on whether Facebook

[3] If the account was recently enrolled, it may not be re-enrolled.

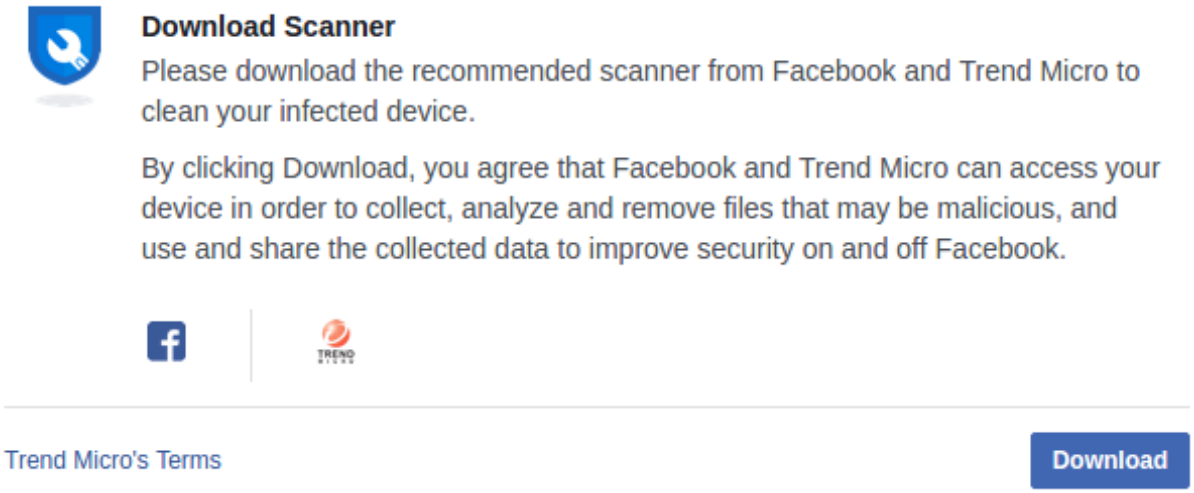

Figure 1.2. The user consent prompt explaining actions the Facebook scanner will take if the user agrees. In this instance the scanner is paired with a third party scanner responsible for removing other types of infections.

believes the sample is malicious. Suspicious extensions or files that have not been seen before (e.g. based on their extension ID) are uploaded to Facebook's infrastructure for real-time analysis. When Facebook's server-side infrastructure indicates to the scanner that a sample is malicious, the scanner attempts to start a cleanup routine that removes the offending sample.

1.3.3 Static Analysis

After the we collect and store samples on ThreatExchange[4] – Facebook's threat intelligence infrastructure – and while the malware scanner is still running on the user's device, we initiate a static analysis pipeline that extracts threat intelligence from the samples.

Our decision for using a static versus a dynamic analysis is based on the understanding that the specific malicious extensions being analyzed are already exhibiting their malicious behavior at collection time. Consequently, we do not have to overcome issues of, e.g., time-gating that a dynamic analysis would be helpful for [47]. Although we execute several distinct analysis functions, the following three are relevant to this paper.

Unpacking. We start by unpacking the sample. Then, we recursively schedule analysis

[4]https://developers.facebook.com/products/threat-exchange

for any files contained within the extracted object. This function can be unsafe in the case where the archive is compressed and has malicious intent, which we handle via sanity checks, such as a limited recursion depth.

Indicator extraction. We attempt to extract threat indicators from each potential malware sample without parsing binaries or code. Instead, we treat each file as a plain text document. This approach, although naive, still generates actionable intelligence from each sample.

We use a series of regular expressions to extract Uniform Resource Locators (URLs), IP addresses, domain names, cryptographic hashes, browser extension IDs, and email addresses. In addition, we attempt to deobfuscate, decompress, decode, and otherwise clean up the code contained in the extensions we analyze. We also make reasonable effort to repair broken URLs and other malformed data.

Once we have the set of initially extracted indicators, we make a second pass, but this time on the extracted data. During this pass we attempt to find more indicators using the type of the original indicator as a clue. For example, for URL indicators that contain API keys, the first pass extracts the URL and the second pass extracts the API key from the URL.

External sharing. We share the full collected samples with ThreatExchange and VirusTotal only if either of the following two conditions are met: (i) the number of users having the specific sample are beyond a specific threshold, or (ii) in the case of Chrome extensions, if the extension is live on the Chrome store. We are able to detect the latter by constructing and accessing a URL that points to the extension on the Chrome Web store.

1.4 Browser Extension Labeling

In this section we describe our methodology for labeling browser extensions. Labels represent a status of maliciousness, and we assign extensions one of two values in decreasing order of severity:

- `MALICIOUS` samples are those deemed with high confidence to be malicious. The malware

scanner described in Section 1.3.2 will subsequently remove them when users agree to an anti-virus scan.

- `UNKNOWN` is the default status for all samples for which we do not have a definitive opinion.

We describe the rules our system uses to propagate labels from individual indicators all the way to entire extensions, and also how changing labels propagate through the system. While the system automatically extracts indicators and propagates labels, there are some situations where traditional manual analysis still plays a role and we end by discussing how the system incorporates input from analysts.

1.4.1 Automated Extension Labeling

We start by assigning high-quality labels to individual threat indicators (e.g. URLs). These indicators come primarily from the system responsible for identifying spam activity, as described in Section 1.3.1, which the labeling system assumes to be ground truth. In essence, the malware labeling process is designed to apply these vetted threat labels onto the indicators extracted from samples via the static analysis pipeline (Section 1.3.3).

All indicators receive an initial label, but Facebook also maintains a feedback process to flag and re-evaluate them over time if it learns new information. As a result, a URL erroneously marked as `MALICIOUS`, for example, will be appropriately re-labeled. This update will then automatically propagate to the relevant samples, which will subsequently be queued for re-labeling.

Propagating Maliciousness Labels

At a high level, our automated browser extension labeling system operates under the basic assumption that if a text file (e.g. a JS file) contains indicators marked as `MALICIOUS` in the ground truth data, then we can deterministically propagate this label to the containing file. Furthermore, if a file labeled as `MALICIOUS` is a part of container (e.g. a browser extension), then we can deterministically propagate that label to the container. For example, if the URL

http://www.example.com/evil.php is considered `MALICIOUS`, and a file *background.js* contains this URL, then the file will be labeled as `MALICIOUS`. And if a Chrome extension *goats.crx* contains *background.js*, then the extension will also be labeled as malicious.

In practice, there are also cases that require an explicit policy decision on how to propagate labels. Although the policies we have chosen may introduce noise into the analysis, our experience has been that overall the system has a sufficient number of strong indicators that it overcomes that noise when it ultimately labels extensions.

Shared resources. If an indicator represents a shared resource — e.g. an IP address used as a Network Address Translation protocol (NAT) gateway — it can be used by both benign and bad actors concurrently. In this case, labeling a file that contains the named IP address as `MALICIOUS` would be equivalent to erroneously marking all traffic originating from that IP as `MALICIOUS`. For simplicity of implementation, our policy is to still propagate labels even on indicators for shared resources, rather than to try to identify and differentiate between shared and non-shared situations.

Inactive code. Another example are inactive blocks of code referencing `MALICIOUS` indicators. Indeed, the malicious block of code is not executable, why label the file as `MALICIOUS`? Our policy is to still propagate the label from indicators on inactive code to the containing object. We argue that, if an actor has the capability to add any type of code into a file, then they may also have the ability to activate previously inactive malicious code.

Gating. Finally, indicators with geographically or temporally gated malice have the potential of erroneously labeling samples when the labeling action occurs outside such boundaries. For geographic gating, our policy is to disregard the boundary and apply globally the label of the indicators with the highest severity. If any users experience malicious behavior, our goal is to protect all users. However, temporal gating requires more attention, specifically for indicators that have been malicious in the past, but, after re-evaluation, we can positively characterize them as not malicious.

Cleaning Up False Positives

The system needs to react quickly and automatically to the discovery of false positives. If the system incorrectly labels an extension as `MALICIOUS`, then the extension will be removed by the malware scanner the next time devices with the extension are scanned.

Using the same rule engine used for propagating labels from indicators to files and extensions, we also create a set of rules to automate correcting false positives. Specifically, if an indicator changes status from `MALICIOUS` to a lesser severity (e.g., `UNKNOWN`), (i) we identify all malware samples containing the indicator, (ii) we filter out all samples that have any status other than `MALICIOUS`, and (iii) for the remaining samples we set their status to `UNKNOWN` if they do not have any remaining `MALICIOUS` indicators. Similarly, when the a `MALICIOUS` sample receives a new, less severe status, we re-compute the status of its containers by applying the same set of rules used for updating indicators.

Known False Positives

Throughout the 6-week measurement period our system collected over 34k unique extensions of which 124 are *known* to have been incorrectly labeled as `MALICIOUS`. Additionally, the median time to identify a false positive is 18 days. As a result, in 0.8% of total scan sessions, our system removed one or more of these 124 extensions erroneously. After the extension is removed, the user can re-install the extension and likely will not be re-enrolled in the malware cleanup process described in Section 1.3.2. We consider this number of false positives as small in number and of an acceptable magnitude.

1.4.2 Manual Labeling

While we consider our automated MBE labeling system highly effective, there are also cases where a threat analyst may need to manually examine a sample to decide its status. Such cases include: (i) Suspicious extensions with highly obfuscated code that circumvents the static analysis pipeline's ability to extract threat indicators. (ii) Suspicious samples that may

contain evolved adversarial capacity to bypass our detection capabilities. Even if the sample was correctly labeled as `MALICIOUS`, in such cases analysts are responsible for examining the samples and communicating their findings within Facebook. (iii) Malicious indicators or samples that are responsible for labeling multiple samples within a short period of time and beyond a certain alerting threshold. Such cases are indicative of a commonly reused, erroneously labeled `MALICIOUS` sample, and the manual analysis step will prevent many false positives.

Any threat status manually applied by an analyst always dominates labels originating from the automated systems. Therefore, such systems are configured to never overwrite an analyst-originating threat label.

1.4.3 A Real World Example

To make this process more concrete, we conclude with an end-to-end example of a botnet that targets Facebook users to disseminate malicious content. A user's browser becomes infected when the MBE[5] is installed via the Chrome Web Store (it is unknown if installation is a result of user choice, or through another attack vector). Once installed, the extension monitors all Web content that the user accesses by sending the URLs to a command-and-control (C&C) server. Additionally, the extension periodically requests from a C&C server remote resources that are executed in the user's browser. Most of the time, the resources do nothing, allowing the MBE to appear benign until the botnet operator initiates an attack. This behavior helps explain why VirusTotal's 57 anti-virus engines consider the extension to be non-malicious, and why it was on the Chrome Web Store.

When active, the MBE manipulates Facebook's DOM with side effects that are detectable both by the DOM scanner, as well as by users themselves (who subsequently reported issues to Facebook). As a result, when the botnet began executing malicious payloads, these side effects provided the first signals of its existence to our system, which resulted in automated classification of the sample as `MALICIOUS`.

[5] e.g. MD5 `a369ecc2e8ca5924ddf1639993ffa3aa`

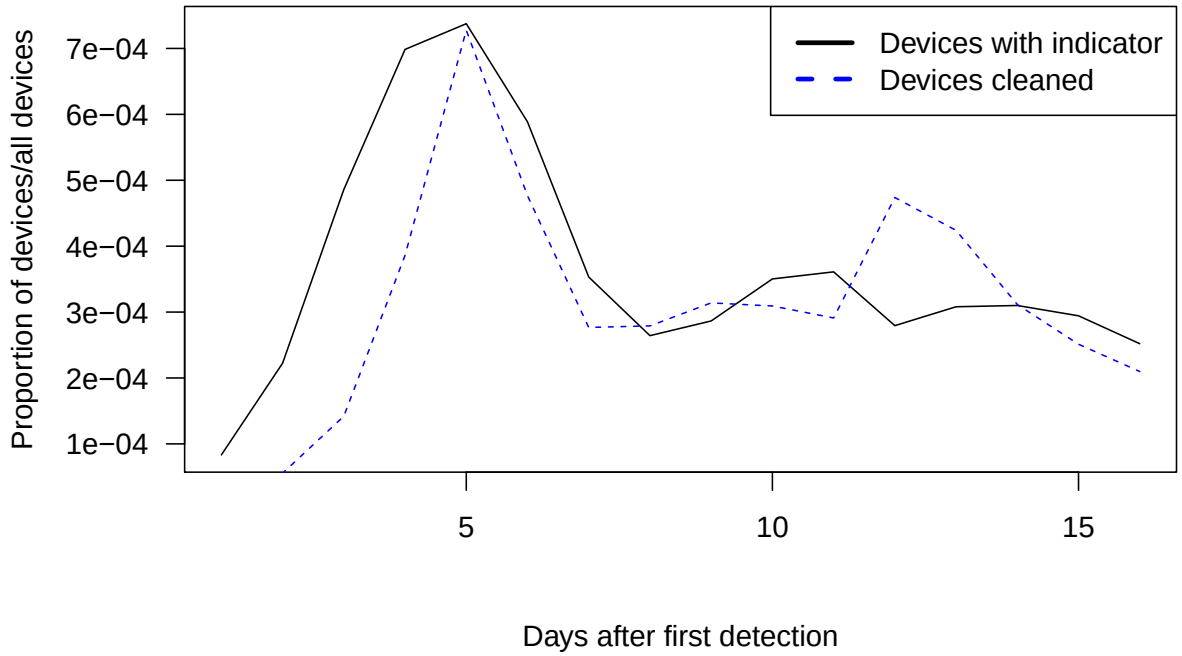

Figure 1.3. Daily proportion of user devices detected with a DOM-based indicator of the botnet, and the proportion of user devices that have the botnet remediated.

Figure 1.3 shows the detection and remediation of this MBE over time. The x-axis shows the number of days after the MBE was first detected by Facebook. The y-axis, normalized by the number of devices scanned daily, shows both the proportion of scanned devices detected as being infected, and the proportion of devices with the MBE cleaned from their system. The lag from when an indicator is detected on a device, and when the device performs malware cleanup, is due to the two processes being independent. In less than a week it peaks while Facebook's malware scanner actively cleaned infected devices. After two weeks, almost all extensions had been removed from the browsers of Facebook's users.

1.5 System Evaluation

We now evaluate our system for automatically labeling malicious browser extensions using extension data collected over a period of six weeks, spanning the end of 2016 through early 2017. We start by characterizing the volume of data our infrastructure processes, focusing on Chrome and Firefox extensions. The volume underscores the need for an automated system.

Table 1.1. Collected browser extensions broken down by browser name, status, contained samples and indicators, and by number of scan sessions reporting a specific type of extension. A scan session may collect both Firefox and Chrome extensions if both browsers are present on a given machine, and thus these percentages add up to more than 100%.

	All extensions		**Malicious extensions**		**Extension contents**		**Extracted indicators**		**Scan sessions**	
	#	%	#	% of total	JS	HTML	Total #	Malicious (#/%)	#	%
Chrome extensions	23 376	67.6	1 697	7.3	67 380	720	66 134	1 559 (2.4%)	718 497	96.9
Firefox extensions	11 183	32.4	88	0.8	17 979	16	19 004	609 (3.2%)	257 164	34.7
Total unique	34 559	100.0	1 785	5.2	84 905	733	73 281	1 516 (2.1%)	741 276	100.0

We then show the system in operation and its behavior over time.

1.5.1 Extensions Collected

Table 1.1 shows a high-level breakdown of the browser extensions we collected over the six-week period. Overall, Facebook's malware scanner collected a total of 34 559 distinct browser extensions from $741k$ distinct scan sessions. We uniquely identify a browser extension by its XPI identifier for Firefox extensions, and by its CRX identifier for Chrome extensions. Extensions are more popular among Chrome users as the majority of collected distinct extensions (67.6%) came from Chrome.

As shown in Table 1.1, throughout the six-week period our system extracted more than 85 000 unique HTML and JS files, with 79.5% of them originating from Chrome extensions. Note that a small number of JS files (454 in total) appear both in Chrome and Firefox extensions, and are cases of libraries like jQuery commonly shared among JS-based applications. Additionally, three HTML files appear in both Chrome and Firefox extensions, and are related to the Potentially Unwanted Program (PUP) `Conduit.A`.

Among the collected extensions, our infrastructure extracted a total of 73 281 unique indicators. Most of these indicators (90%) were embedded in samples extracted from Chrome extensions. As with the JS files, 9 398 indicators overlap across the two types of extensions due to common references to certain resources like domains, URLs, and email addresses.

For Chrome extensions, we find 2 200 (9.4%) of the 23 376 total extensions to have been on Google's Web Store at least once. The high proportion of extensions likely installed via

sideloading (90.6%) is not surprising as Facebook's malware scanner runs on devices suspected of being infected, and Google removes extensions they consider malicious from the Web Store.

1.5.2 Malicious Extensions Detected

We now consider the methodology for automated MBE labeling we presented in Section 1.4.1, and examine its application to the extensions we collected and the files we extracted during the six-week measurement period.

Of the 34 559 extensions from this period, we classified 5.2% of them as malicious. As expected, attackers clearly target Chrome more often. From the 11 183 Firefox extensions, only 0.8% of them are labeled malicious. Yet, of the 23 376 Chrome extensions, 7.3% are malicious. This bias naturally reflects browser market share, as Facebook sees predominantly more traffic from Chrome and attackers concentrate on the platform most popular with users. Of the malicious Chrome extensions identified, 24.9% have been accessible on the Web Store at least one time throughout the measurement period.

The small portion of extracted threat indicators labeled malicious in Table 1.1 (2.1% of all extracted indicators) highlights the effectiveness of our labeling methodology in trickling up known badness. The malicious indicators used to label MBE are primarily domains and URIs, with the exception of a single email address that resulted in labeling one Chrome extension as malicious.

Figure 1.4 shows the behavior of the automated labeling system over time as it detects and labels MBEs. Each point represents the number of new extensions labeled as malicious on a given day (x-axis), even if the extension was first seen on different day. The spike spanning days 32–35 is linearly correlated with the fluctuation in the number of users clearing the malware checkpoint at the same period. On an average day the system labels 39.5 (median: 37) Chrome extensions and 2 (median: 1) Firefox extensions as malicious.

In general, identifying new malicious extensions is immediate: for over 90% of newly-collected browser extensions, the system labels them as `MALICIOUS` with a median time of 21

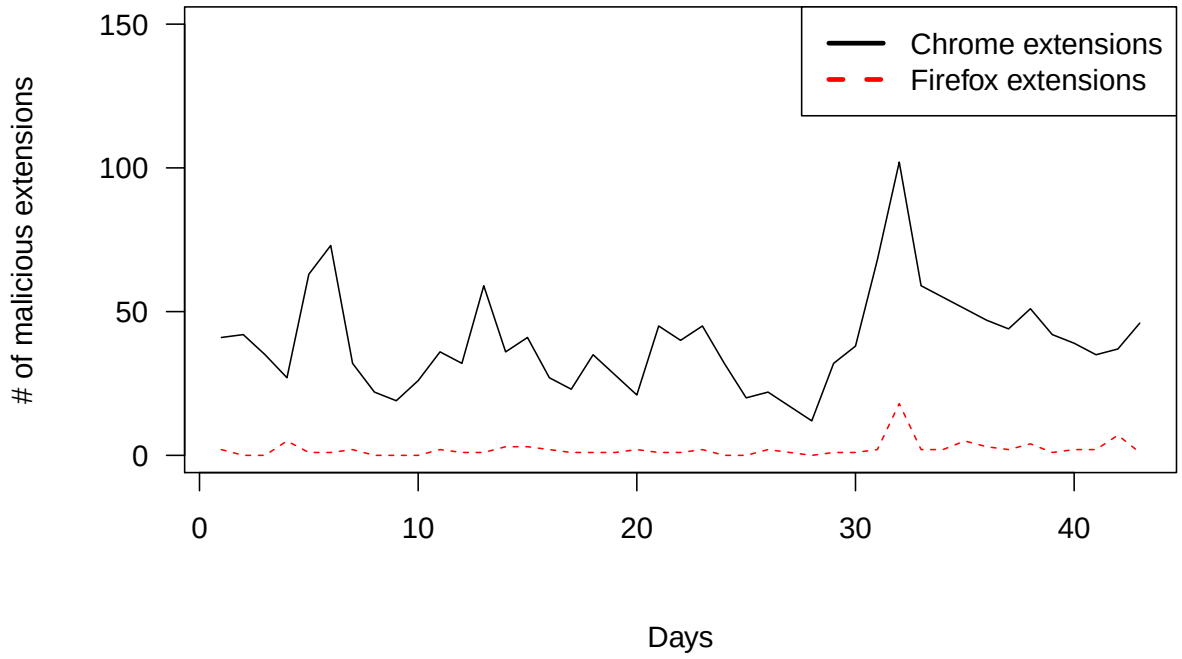

Figure 1.4. The number of unique extensions labeled as malicious each day of the six-week measurement period.

seconds after collection. However, some extensions are initially labeled benign and are only later discovered to be malicious when their embedded indicators are associated with abusive behavior. In our measurement period, we only found 143 (8.0%) extensions that are eventually labeled `MALICIOUS` more than 1 day after they are first collected, and these extensions are found on $\approx 9\%$ of all users cleaned during the measurement period. Delayed discovery is expected with an indicator-based labeling system as the status of an indicator can change over time, and we consider the number to be acceptably low for an operational system.

1.6 Evaluating Alternatives

The system evaluation shows that Facebook's MBE labeling is effective at detecting, labeling, and cleaning malicious extensions. A related question is whether it is necessary to create a new system to perform this task. Next we evaluate alternatives to underscore the need for developing a new system to protect Facebook and its users from large-scale abuse via browser extensions. For this evaluation, we focus on Chrome extensions since they dominate what

we encounter on user's devices. In particular, 2 200 extensions once available as "public" or "unlisted" on the Chrome Web Store, of which Facebook labeled 422 (19.2%) as malicious, and 1 778 (80.8%) as unknown. Recall that these are extensions from users that exhibited suspicious activity on Facebook and triggered an anti-virus scan, so we would expect a greater concentration of malicious extensions in this smaller set.

1.6.1 VirusTotal

We first use VirusTotal to evaluate whether Facebook can use general databases of malware to detect malicious extensions. VirusTotal is a popular online system owned by Google that analyzes malware files using a suite of 57 anti-virus products, and reports which A/V products label a file as malicious (if any).

We initially use the set of new extensions publicly available on the Chrome Web Store overlapping with our measurement period. The authors of the Hulk system [50] kindly shared these extensions with us, and they total 9 172 unique CRXs. As a baseline we submitted the shared public extensions their system collected to VirusTotal. VirusTotal was aware of only 73 (0.8%) of them, and considered only 5 (0.1%) as malicious.

Additionally, out of the 422 `MALICIOUS` extensions as labeled by Facebook, only 22.7% are identified as malicious by one or more anti-virus engines. We conclude that a general malware database like VirusTotal is insufficient for detecting MBEs for sites like Facebook.

1.6.2 Chrome Web Store

Google also has a vested interest in maintaining the health of the Chrome extension ecosystem, and therefore also actively removes extensions that it determines to be malicious. Since another option for Facebook would be to rely upon Google's efforts, as a final step we quantify the benefits that Facebook's MBE labeling system is able to provide by focusing on just its service beyond what Google provides to all Chrome users.

When an extension is removed from the Chrome store, we conservatively assume that

the extension was removed because Google considered it malicious. Since extensions may be removed for other reasons (e.g., developers removing their own extensions), this represents an upper bound of Google's detection capability. By the end of the measurement period, Google removed 367 of the 9 172 extensions from the Chrome store (70 `MALICIOUS` and 297 `UNKNOWN` based to our labels).

In addition to cleaning up the malicious extensions, another goal of our MBE labeling system is to reduce the time that they are active and profitable to attackers. Using the public user counts listed on the Web Store we estimate that these 70 malicious extensions have been installed 1 009 806 times. Of the 70 MBEs, Facebook always labels the extensions as malicious before Google removes them, with a median difference of 67.3 hours. Thus reducing the median monetization window of malicious extensions by over 2.8 days.

1.7 Conclusions

Malicious extensions are a vexing problem and one that is challenging to address from any single vantage point. While browser vendors are in a position to restrict which extensions are distributed and, in principal, which extensions may be installed, they have limited insight into which extensions act abusively in the wild. Indeed, some extension's malicious code is only loaded at run-time and even then may only be activated for particular sites. Conversely, abused sites directly experience malicious behaviors but they are not in a position to identify which extensions are implicated in a given attack because this information is not available through the Web interface.

In over six weeks of deployment at Facebook our system has identified more than 1 700 malicious Chrome and Firefox extensions. Comparing our findings with both contemporaneous anti-malware detections (as reflected in VirusTotal) and takedowns from the Chrome Web Store, reveals a considerable detection gap in the existing abuse ecosystem. We hope that by highlighting this issue and sharing our data we can encourage a broader and more collaborative

focus on this under-addressed attack vector [6].

Acknowledgements

Chapter 1, in part, is a reprint of the material as it appears in *Proceedings of Workshop on Cyber Security Experimentation and Test (CSET)*. Louis F. DeKoven, Stefan Savage, Geoffrey M. Voelker, Nektarios Leontiadis, 2017. The dissertation author was the primary investigator and author of this paper.

[6] MD5 hashes of the 422 identified Chrome MBE available in VirusTotal and ThreatExchange: https://pastebin.com/nzVGPLnr

Chapter 2

Following Their Footsteps: Characterizing Account Automation Abuse and Defenses

In this chapter, we develop a system to disrupt abusive underground services selling inauthentic actions on Instagram. Similar to our first study (Chapter 1), the actions produced by these services are observable from the vantage point of an online social network. However, the corresponding malicious software is not required to run directly on each customer's device; making the use of a malware scanner-like approach impractical. As a result, we explore different intervention techniques to disrupt the abusive actions themselves.

We focus on five popular underground services. From these services, we identify two techniques used to inflate the social status of a customer's account: collusion networks and reciprocity abuse. We then develop a honeypot framework for Instagram that allows us to engage with abusive services as a customer. Using this framework and signals produced by Instagram, we measure a number of operational characteristics for each underground service. Lastly, we develop two controlled interventions to compare the effectiveness of different approaches to countermeasure. We find underground services are able to attract large clientele, and generate over $1M in estimated monthly revenue. Additionally, we experimentally demonstrate that transparent interventions (e.g., blocking actions from a given account automation service) quickly provokes adversarial adaptation, while deferred interventions (e.g., removing service actions a day later) is far more likely to go unanswered.

2.1 Introduction

Social media, as with all forms of mass communication, provides a platform whereby a single message can reach large audiences. However, the reach of any given message is determined by the popularity of the user who publishes it. Concretely, users with more followers are able to reach larger audiences with their posts and thus can be seen as carrying more "weight" in some abstract social hierarchy. Since this standing is directly monetizable via advertising, it is unsurprising that this aspect of social media has attracted organized abuse. Indeed, the medium has engendered a large underground service market that focuses on bypassing the organic nature of social relationships and instead advertises the ability to create artificially enhanced social network status in exchange for payment.

In this paper we explore this phenomena in the context of the popular Instagram photo-sharing service. To wit, searching for "Instagram likes" in a search engine will produce pages of sites with inducements such as "Buy Instagram Likes from $2.97 only!" or "Instant Instagram Likes — 100% Real & Genuine Likes". However, the precise mechanism by which such services ply their trade is unclear and, in fact, simplistic "bot-based" approaches (whereby a service creates fake accounts and uses them to initiate social actions to customer content) are easy to detect and filter. In our work, we focus on the more sophisticated segment of this market, AASs in which users provide their Instagram credentials to third party actors who, in turn, use those credentials to perform actions on the user's behalf in a manner that violates Instagram's Terms of Use [42].

We have explored these services through a variety of techniques. Using a broad array of independent "honeypot accounts" we engaged (on behalf of these accounts) with five large account automation services: Instalex, Instazood, Followersgratis, Boostgram and Hublaagram. By requesting a range of "social actions" from each AAS, and then monitoring activity to and from the associated accounts, we inferred the mechanisms each service uses to achieve its ends. Notably, we distinguish two distinct techniques — collusion networks and reciprocity abuse —

used to artificially create social connectivity. Using our service characterizations we were then able to identify all accounts used by customers of each service. Collecting data on this corpus over several months, we were able to characterize the dynamics of their customer populations and the underlying revenue of each business. Finally, we performed controlled experiments to evaluate different kinds of interventions (e.g., blocking such services from accessing Instagram vs. removing their actions at a future date) and the reactions each kind of intervention evoked from the services and their customers.

We believe our work is the most comprehensive study of this kind to date on Instagram, and that our analysis provides several insights that were not previously understood or lacked empirical validation in the broader space of social network abuse:

- **Social action laundering.** We identify two techniques designed to artificially create social actions while evading traditional detection mechanisms. The first, *reciprocity abuse*, leverages the tendency of some users to issue complementary `follows` or `likes` in response to an unknown user `following` them or `liking` their content. This reciprocity effect allows services to quickly inflate the `follower` or `like` counts of their customers by automating *outbound* actions to a curated set of recipients.

  The second approach, *collusion networks*, uses the entirety of a service's population to orchestrate the exchange of social actions. Thus, each customer account is used to issue `follows` or `likes` to other customers, and they in turn receive inbound actions from yet other customers (similar, in principal, to the notion of a mix network [16]).

- **Commercial scale.** We find that these services are quite successful as business entities and we estimate the gross revenue among three of the five AASs alone to be over $1M *per month*. Moreover, we show that long-term customers (i.e., customers who repeatedly contract for services over multiple months) provide the lion's share of these proceeds (i.e., that the core set of customers is stable and customer churn is modest).

- **Intervention impacts.** We experimentally demonstrate that transparent interventions (e.g., blocking actions from a given account automation service) quickly provokes adversarial adaptation, while deferred interventions (e.g., removing service actions a day later) is far more likely to go unanswered. Somewhat unintuitively, our results suggest that related abuse interventions will be most effective and long-lived precisely when they do not visibly undermine the business model of the abusive service.

In the remainder of this paper we provide background on how such social networks operate and are abused, describe the set of AASs we explored, and provide a detailed description of our measurement methodology. We provide an analysis of both user dynamics and service revenue, and then describe a series of controlled intervention experiments that explore how for-profit service abuse businesses respond to different varieties of disruption.

2.2 Background

Online social networks (e.g., Twitter, Facebook, Instagram, Youtube, Snapchat, etc.) are targeted by abusers that engage in activities spanning from selling fake actions to hijacking user accounts. We are not the first to identify this phenomenon, and other researchers have characterized a range of such practices that we build on in our own work. Javed *et al.* characterized generic traffic exchange services that provide customers with inflated view counts—including for social media—from large pools of IP addresses, and find many exchanges which pay users in return for views to their content [49]. This work establishes both the commercial nature of such abuse and the use of live humans as traffic sources. Hooi *et al.* develop a bipartite graph algorithm to detect abusive actions on the Twitter follower-followee graph, where miscreants may camouflage their abusive actions by producing actions to non-customers [37]. Again, this work identifies the use of organic (i.e., non-bot) accounts as a critical challenge in social network abuse and uses statistical techniques to try to distinguish legitimate and illigitmate actions from such accounts.

Other researchers have tried to overcome this issue by using honeypot accounts to crisply identify abuse targeting across a range of social networks including MySpace, Twitter, and Facebook [54, 83, 88, 1, 93]. Moreover, in some cases, this data has then been used to successfully train classifiers to identify those accounts complicit in collusion networks [1, 88]. Our work builds on both of these techniques—the use of honeypots to obtain abuse data and using this data to train abuse classifiers—in our analysis of Instagram abuse.

The honeypot approach has also been combined with active purchasing from third-party services to investigate commercial abuse. For example, De Cristofaro et al's analysis of Facebook services [19] and Stringhini et al's analysis of Twitter following services [84] both use this approach and characterize the nature of the fraudulent social networks they find. Our work is distinct, not only due to the different social network examined (Instagram), but also because we focus on more complex (i.e., non-bot) forms of abuse in our work. As well, we are able to provide a grounded analysis about service revenue that informs how we consider the nature of the threat and focused experiments exploring the impact of different interventions.

Mislove *et al.* identified the existence of high degrees of reciprocated actions within online social networks (e.g., Flicker, YoutTube, etc.) which, a decade later, forms the basis for the reciprocity abuse we identify in this work [63]. Finally, most closely related to our work is that of Farooqi *et al.* who describe a collusion network abusing third-party application OAuth tokens on Facebook, and the results of large-scale network-level blocking of the organizations behind this activity [25]. Our work brings a related analysis to a distinct social network and extends it by analyzing reciprocity abuse as well as collusions networks, quantifying the underlying business and revenue model for multiple abuse groups, and performing active experiments with finer-grained (i.e., account-level) interventions.

For this paper, we focus squarely on Instagram, a popular online social network structured around sharing and discussing photos posted by its 800 million users [41]. In normal use, each Instagram user can upload photos and videos, apply visual filters and tag photos with hashtags. A user's followers will see the media the user has posted, and can interact by `liking` the media

and posting comments. Thus, users with more followers will have their content exposed to a broader audience and will receive on average more interactions.

Typically, differences in social status (e.g., the number of `likes` per photo, `followers`, etc.) are an organic byproduct of each user's own authentic activity. However, in addition to the implicit psychological factors that drive users to desire increased social standing, there can be strong economic incentives as well. Notably, after reaching a social status commonly referred to as an "influencer", outside businesses may offer to pay users thousands of dollars in exchange for posts (e.g., for marketing purposes) [67, 61]. It is a popular belief in this community that, to become an influencer, a user of Instagram needs an account with both a high engagement (i.e., a large number of other Instagram users that interact with posted content), and thousands of followers [61]. The potential for such inducements leads some users to pursue increased social status via abusive means, and gives rise to third-party services that perform this function for a fee. Indeed, such services formalize this notion and promote a metric called the "engagement rate" to evaluate the quality (and hence potential profitability) of an influencer [45]. They argue that users should try to maximize this metric:

$$ER = \frac{\text{Number of likes \& comments}}{\text{Number of followers}}$$

and commonly offer to manipulate one or more of its components as a key aspect of their service offering (with one such service claiming that each $1 spent produces a return of $6 in marketing revenue).

One approach for achieving this end is to create a range of synthetic Instagram accounts and use them to `follow` the accounts of paying customers, `like` their content, and so on. However, this kind of purely synthetic account manipulation can be easy to detect. Indeed, over the last year Instagram has worked to disrupt a range of popular bot services including Instagress, MassPlanner, PeerBoost, InstaPlus, and FanHarvest [31, 60, 66, 86]. The more sophisticated players in this ecosystem perform "account automation" whereby their customers provide access

to their Instagram login credentials, and the service performs actions on their behalf. In fact, Instagram provides a public OAuth-based API that allows a Web site to perform actions on behalf of users that grant permission. However, this API is rate limited in a manner that precludes broad abusive use. Thus, most commercial account automation services bypass these limitations by reverse engineering the private API used by the Instagram mobile client and generating spoofed requests to appear as valid mobile client actions.

2.3 Account Automation Services

Based on our observations, AASs use two distinct approaches to achieve their ends: (i) Reciprocity Abuse and (ii) Collusion Networks. The former aims to provide *authentic* actions (i.e., `likes`, `follows`, etc.) to their customer's Instagram account, while the latter provides customers with *inauthentic* actions to their Instagram account. In this section we describe each approach, and then detail the particular set of services we studied in this effort.

2.3.1 Reciprocity Abuse

Reciprocity Abuse AASs provide their customers with organic actions from other Instagram user accounts by exploiting the concept of social *reciprocity*. For example, when Instagram user $A1$ receives an (inbound) action from Instagram user $B2$, $A1$ will be notified in real-time about $B2$'s action, and $A1$ may reciprocate by performing an action to user $B2$. This "you follow me, I follow you" behavior is an organic response taken by some subset of Instagram users. Reciprocity Abuse AASs abuse this behavior by automating large numbers of (outbound) actions from their customer's Instagram account in the hope that a subset of users receiving an action will return the favor in kind — thus providing their customer with inbound actions, such as `follows`.

2.3.2 Collusion Networks

By contrast, Collusion Network AASs provide their customers with inbound *inauthentic* actions on their Instagram accounts. A collusion network is a group of Instagram accounts used in concert to orchestrate actions to one another. Accounts participating in the collusion network will produce outbound actions to other accounts in the network, as well as receive inbound actions from the network. Customers of Collusion Network AASs are hoping to strictly increase the number of actions on their Instagram account and they are willing to have their account used in the same manner on behalf of others to serve this goal.

2.3.3 Studied services

We study five popular AASs in detail that we discovered through a combination of searching popular underground forums for popular recommendations from the community, together with repeated complaints from Instagram users caused by unsolicited AAS advertisements. Three use the reciprocity abuse approach (Instalex, Instazood and Boostgram), while the other two implement collusion networks (Hublaagram and Followersgratis). For each service, we explored its Web site in fall 2017 to understand the registration process, what features are offered, and the advertised business model [43, 46, 8, 39, 29]. Figure 2.1, for example, shows a screenshot of the Instalex customer control panel. During this process, we also discovered that the Instalex and Instazood services were independently operated franchisees of the same parent organization (which offers franchising services ranging from $1,990 to $30,990 per month [44]). Since they appear to be operated independently, we evaluate these two services separately until Section 2.5 where we combine the two services when we cannot separate their actions.

Registration Process

Both Reciprocity Abuse AASs and Collusion Network AASs produce automated activity from the Instagram accounts of their customers. Therefore, a required step when registering for any AAS is for the customer to provide their Instagram account credentials (e.g., in contrast to

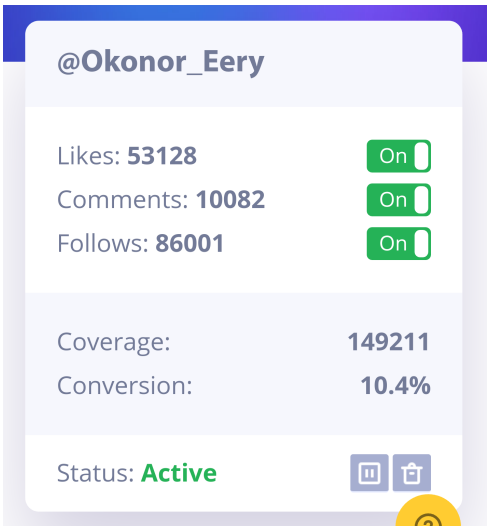

Figure 2.1. Instalex Web site providing an example account control panel with action counts performed on Instagram.

abuse methods where the AASs depends on the ability to use customer OAuth tokens [25]). By sharing their Instagram credentials the customer gives an AAS *full* control over their Instagram account, while resetting the password revokes AAS access to the account.

Table 2.1 shows the different AASs by name, service type, and what services are available to customers. All offer `like` and `follow` services, 60% offer `comment` and `unfollow` services, and 40% offer `post` services. It comes as no surprise that every AAS offers at a minimum `likes` and `follows` as these are the most frequent actions on Instagram. Some AASs provide `comment` and `post` services as additional ways for their customers to attract other Instagram users to engage with their content. Lastly, all Reciprocity Abuse AASs provide `unfollow` services that allow their customers to remove the outbound `follows` performed by the AAS in an effort to retain only the inbound `follows` they receive.

Many Reciprocity Abuse AASs allow their customers to target groups of Instagram accounts that will receive automated actions, allowing their customers to obtain reciprocated actions from users with common interests. Customers can provide either a list of Instagram users, or a list of hashtags to narrow the accounts that a AAS will interact with. When signed into a Collusion Network AAS, customers are typically given the option to request a specific

Table 2.1. Services offered to customers of Reciprocity Abuse AASs and Collusion Network AASs.

	Reciprocity Abuse AASs				
Service	Like	Follow	Comment	Post	Unfollow
Instalex	★	★	★		★
Instazood	★	★	★	★	★
Boostgram	★	★		★	★
	Collusion Network AASs				
Service	Like	Follow	Comment	Post	Unfollow
Hublaagram	★	★	★		
Followersgratis	★	★			

type and *quantity* of inbound actions (e.g., $2,000$ `likes`, etc.) to other customers of the network, but cannot specify the interests of accounts they receive actions from.

AAS Business Model

The primary revenue source across the studied AASs is customer payments for the services they offer.[1] In turn, there are two different techniques used by AASs to attract customers in the hope that they become paying customers: trial periods, and free services.

First-time customers of Reciprocity Abuse AASs are commonly offered a free variable-length trial period. During the trial period customers have access to all of the service's features. However, as soon as the trial period expires the service is discontinued, and if the customer wants to continue service they are required to pay. Reciprocity Abuse AASs have a relatively straightforward cost structure where customers pay for each of their Instagram accounts to gain full use of the service for a specified time period. Table 2.2 presents the free and paid service options for customers of the Reciprocity Abuse AASs we study.

Collusion Network AASs offer customers the ability to periodically request small quantities of actions onto their Instagram account for "free". Soon after a customer provides their Instagram credentials the service will begin to use the account in the collusion network. Hublaa-

[1] There is also a minor revenue stream arising from advertisements shown to customers, but it does not appear to be significant by comparison (Section 2.5.2).

Table 2.2. For Reciprocity Abuse AAS we show the free trial length, the minimum number of days that service can be purchased for, and the corresponding cost per Instagram account.

Service	Trial Days	Min Paid Days	Cost
Instalex	7 days	7	$3.15
Instazood	3 days	1	$0.34
Boostgram	3 days	30	$99

Table 2.3. All per-account costs for Hublaagram services. Hublaagram allows customers to pay a one-time fee that prevents their Instagram account from participating in the collusion network. Services with an immediate duration are applied as fast as possible to a single post, and services with a month duration have the purchased quantity of `likes` applied to *each* new photo posted on the account throughout the month.

Description	Cost	Duration
No collusion network	$15	Life
2,000 Likes	$10	Immediate
5,000 Likes	$20	Immediate
10,000 likes	$25	Immediate
250 – 500 Likes	$20	Month
500 – 1,000 Likes	$30	Month
1,000 – 2,000 Likes	$40	Month
2,000 – 4,000 Likes	$70	Month

gram provides free `likes`, `follows`, and `comments`, while Followersgratis only offers free `follows`. Free service, though, is rate-limited; Hublaagram, for instance, has a 30-minute timeout between requests. Naturally, both Collusion Network AASs encourage customers to pay money to receive a larger quantity of actions. We present the different paid service options for Hublaagram and Followersgratis in Tables 2.3 and 2.4, respectively.

2.4 User Experience

In this section we evaluate the experience of using Account Automation Services from a user's perspective using a collection of fully-instrumented honeypot accounts.

Table 2.4. The Followersgratis payment options. With `likes`, customers who select the less expensive option receive `likes` from Instagram accounts located around the world on five different photos. The more expensive `like` option provides `likes` from Instagram accounts located in Indonesia, and the `likes` are spread across ten photos. The duration for `likes` is specified explicitly on the Followersgratis Web site without explanation.

Description	Cost	Duration
500 Follows (300 free likes)	$3.15	1 Day
1,000 Follows (500 free likes)	$5.25	1 Day
500 Likes (250 free likes)	$2.10	Instant
500 Likes (500 free likes)	$5.25	Fast

2.4.1 Methodology

To identify abusive actions generated by the AASs, we registered multiple distinct honeypot accounts with each service described in Section 2.3.3. Thus, for each account, we register it with an AAS, request that the service perform either inbound or outbound actions on the account, and then monitor the resulting actions. Since they neither generate nor receive organic actions, honeypot accounts are particularly useful because we can attribute all activity to the linked AAS. We describe our methodology for using honeypot accounts in more detail below.

Account Types

We developed a honeypot account framework to programmatically manage a large number of Instagram accounts. Our framework supports campaign-specific accounts, account creation, posting content, deletion, and data collection of all inbound and outbound actions on the account. When deleting a honeypot account, all actions to or from the account are eventually removed from Instagram.

For each service, we created two different types of honeypot accounts to determine if AASs differentiate between fake or real-looking Instagram accounts (they do not), and if there is a difference between reciprocated action rates from Instagram users that receive an outbound

action from AASs (there is; more below in Section 2.4.3).

The two types of honeypot accounts we register are "empty" and "lived-in" accounts. Empty accounts contain the minimum information required to use all of the AASs that we study. In particular, we populate honeypot accounts with 10 or more photos from one of the following categories: dogs, cats, lizards, and food. Lived-in accounts, in addition to having uploaded photos, are fully populated Instagram accounts with a profile picture, biography, and name, all unique. Lived-in accounts follow $10-20$ high-profile Instagram accounts ($>$1M followers), but do not themselves have followers when created. Beyond enrolling them in the AAS services, we do not use them to perform actions on Instagram after being created.

Account Registration

We registered 10 honeypot accounts for every service type offered by each AASs listed in Table 2.1, specifying that the account be used *only* for that service type. For example, as Instalex offers three different services, we registered 30 accounts in the service. Among each set of 10 accounts, nine are empty and one is lived-in. In total we registered over 150 honeypot accounts during the course of a month of manual registration effort. Moreover, some of our accounts engaged with the free services offered by each AAS while others explicitly paid for contracted services. For AASs that require target information for particular actions (e.g., targets of `likes` and `follows`), we created a static list of hashtags and Instagram accounts that could be used in common. We chose relatively high-profile hashtags and Instagram accounts (e.g., having more than 1M followers) to reduce the impact of the temporary actions produced from our honeypot accounts. We also made a point to use a diverse set of commercial and residential IP addresses when accessing each AAS's site in the unlikely event that any of the services actively monitor and correlate connections to their site. Finally, we deleted our honeypot accounts after the measurement period, which removed all of their actions from Instagram.

Attribution

When using honeypot accounts with AASs, we attribute the activity on those accounts solely to their involvement in the AASs. To rule out the possibility that the activity could be due to other users of Instagram, we used a separate set of 50 inactive honeypot accounts to establish a baseline of background activity on Instagram. The inactive accounts are not registered with an AAS, and we never used them to produce actions that are visible to other users of Instagram.

For each account we similarly uploaded at least 10 photos at the time of creation. We then actively monitored whether any inbound action (i.e., `likes`, `follows`, etc.) took place on these accounts. For the duration of our study, we did not observe any activity on any of the inactive honeypot accounts. As a result, for the honeypot accounts we register with AASs we attribute all activity on those accounts to their involvement with the services.

2.4.2 How Accounts Are Used

Using the honeypot accounts, we examine how AASs use the accounts registered with their services.

Since customers provide their Instagram credentials to an AAS during registration (Section 2.3.3), it is possible for the AAS to abuse the Instagram account to produce additional, potentially undesired actions. We compared the types of actions we requested with the types of actions the services actually performed with our accounts (e.g., when requesting `likes` does a service use the account for anything other than `like` actions?). The services all perform as advertised. Across the AASs we study, they only perform actions of the type we requested, and no AASs used our accounts to produce visible un-requested actions.

In later analyses in Section 2.5, such as estimating revenue, it is important to distinguish between users using the free trial periods on services and those users paying money for service. Although the services advertise the lengths of their trial periods (Table 2.2), we also experimentally evaluated their durations using the honeypot accounts. Trial service starts immediately, with

our accounts becoming active within minutes of requesting free service. And with one exception, we confirmed that free trial service lasts for the advertised period, and that activity with accounts stops no more than 12 hours beyond the expected end time. Instazood, however, advertises a three-day trial period, yet all of our honeypot accounts received seven days of trial service. As a result, for Instazood we assume that trial period activity is seven days.

2.4.3 Quantifying Reciprocation

As a final experiment we use our honeypot accounts to measure the probability that an outbound `like` or `follow` will spontaneously generate a reciprocated action. Previous work has shown how collusion networks use their control over the accounts in the network to serve as both the source and target of actions [25]. In contrast, Reciprocity Abuse AASs fundamentally rely upon natural social behavior in online networks to fulfill their customer requests. As discussed in Section 2.3.1, these services produce outbound actions from user accounts under their control, but the targets of these actions are other Instagram accounts that are *not* under the control of the service. The underlying assumption is that, for each action, there is some probability that the target of the action will naturally reciprocate with a similar action. With a sufficiently high volume of outbound actions, these services can then organically induce reciprocating actions to satisfy their customer requests.

Table 2.5 shows the probability of receiving a reciprocated action given an outbound `like` or `follow` for the three Reciprocity Abuse AASs. We separate the results for the two different kinds of honeypot accounts, empty (E) and lived-in (L). For example, generating an outbound `like` with our empty Boostgram honeypot accounts has a 1.5% chance of inducing a reciprocating `like` and a 0.1% chance of inducing a reciprocating `follow`. These results quantify the reciprocity effect of users on Instagram, and from them we make a number of observations.

First, the reciprocation rates are for the most part very consistent across the services. Although Instalex and Instazood are franchises of the same service, they also exhibit recipro-

Table 2.5. The probability of receiving a reciprocated inbound action given an outbound action of a specific type. For each service, we show the reciprocation ratio for both empty (E) and lived-in (L) honeypot accounts.

Service	Outbound	Inbound	
		Likes	Follows
Boostgram (E)	Likes	1.5%	0.1%
Instalex (E)	Likes	2.1%	1.4%
Instazood (E)	Likes	2.1%	0.2%
Boostgram (L)	Likes	3.9%	0.2%
Instalex (L)	Likes	3.7%	1.8%
Instazood (L)	Likes	3.5%	0.4%
Boostgram (E)	Follows	0.0%	10.3%
Instalex (E)	Follows	0.0%	12.8%
Instazood (E)	Follows	0.0%	13.0%
Boostgram (L)	Follows	0.0%	12.0%
Instalex (L)	Follows	0.0%	13.7%
Instazood (L)	Follows	0.0%	16.1%

cation rates that are similar with those on Boostgram. These results are consistent with these services tapping into fundamental underlying online social behavior on Instagram. Moreover, the reciprocation rates are relatively high for `follows`. For just 6–10 outbound `follow` actions, our honeypot accounts receive a new inbound `follow` from a real user. (In Section 2.5.3, we show that the services appear to specifically target users who are more likely to respond to inbound `follows` to increase the probability of reciprocation.)

The one anomaly is inbound `follows` to outbound `likes` on Instalex, which has a reciprocation rate many times greater than the other services. Exploring further, though, we found no significant features in the accounts targeted by Instalex compared to the other services that might explain the difference: The inbound actions come from hundreds of autonomous systems, the time between when the actions take place and when the honeypot account was registered in the service is uniformly distributed throughout the trial period, the inbound actions come from dozens of countries, etc. As a result, we currently do not have an explanation for this one difference.

Second, users primarily reciprocate with the same action, e.g., Instagram users reciprocate with a `like` when receiving a `like` from one of our accounts. Much less often, users will reciprocate to an outgoing `like` by `following` one of our accounts (an order of magnitude less often for Boostgram and Instazood). And users never reciprocate with `likes` when `followed` by one of our accounts.

Finally, Instagram users are sensitive to the differences in honeypot accounts. Confirming expectations, empty accounts have a significantly smaller probability of receiving reciprocal inbound actions than lived-in accounts, particularly for `likes`. Lived-in accounts range from 1.6× as likely on Instazood to 2.6× as likely on Boostgram to generate inbound `likes`. This difference confirms the utility of more realistic honeypot accounts.

2.5 Business Perspective

Our honeypot accounts gave us insight into the AASs from a user's perspective. They were also valuable in providing us with ground-truth on AAS activity, which we were then able to use to identify all activity generated by all Instagram accounts used by the AASs. Based on features gathered from our honeypot accounts, such as the type of action (e.g., `like`, `follow`, account login, etc.), commonly tracked information about the client (e.g., IP address, ASN, etc.), and additional signals produced within Instagram, we can identify the actions initiated by each AAS. The signals produced by Instagram identify abusive services, including the AASs we study during the time of our measurement. While Instagram believes that their signals accurately characterize the entire activity of an AAS, we do not have a way to verify completeness and, as such, the levels of abuse we characterize in this section constitute a lower bound. Throughout our study, though, we *never* detect any changes in the signals tracked by Instagram for our honeypot accounts. We also periodically register additional trial honeypot accounts in each AAS as another method for observing the tracked account signals; these signals are consistent with our original honeypot accounts and also do not change during the course of our study (we delete

these accounts immediately after the AAS starts generating activity on them).

In this section we analyze every action that takes place on Instagram originating from the AASs we study over a 90-day period in late 2017. This rich data set allows us to characterize the magnitude of abuse and revenue generated from AASs. We also present the types of actions performed by each service, as well as the users targeted by these actions to understand how different AASs select their targets.

Note that, for the remainder of the paper, we combine activity from Instalex and Instazood since we cannot differentiate actions performed by individual franchises (Section 2.3.3). To minimize confusion, we refer to their combined activity as "Insta*". Additionally, we exclude Followersgratis from the remaining analyses as the service was already well-policed by pre-existing abuse detection systems that prevent high volumes of abuse originating from a small number of IP addresses. As a result, activity generated by Followersgratis has very limited impact on Instagram in practice.

2.5.1 Customer Base

We explore a range of account-based measurements that help us better understand AAS operating characteristics.

Popularity. How popular are these services? Table 2.6 shows the number of Instagram users who were active in each AAS during our measurement period. Demand for these services is large: Boostgram has more than 10,000 users, Insta* an order of magnitude more, and Hublaagram just over a million. One explanation for Hublaagram's much larger popularity is that it offers prolonged free features compared to the other AASs, and users naturally prefer no-fee services.

Since nothing constrains users from engaging with multiple services, we looked at how many Instagram users enroll their account in more than one service. Overall, account overlap is small. Fewer than 200 accounts generate any activity in the three AASs, $1,963$ participate in two distinct Reciprocity Abuse AASs, and $4,485$ accounts participate in at least one Reciprocity

Table 2.6. Customers participating in each AAS over a 90-day period. Long-term customers of Reciprocity Abuse AASs are active beyond a trial period, and long-term Collusion Network AAS customers request service for more than four days.

Service	Customers	Long-term	Short-term
Insta*	121,661	41,891 (34%)	79,770 (66%)
Boostgram	11,959	3,975 (33%)	7,984 (67%)
Hublaagram	1,008,127	501,428 (50%)	506,699 (50%)

Abuse AAS as well as the Hublaagram collusion network. In these cases, nearly all are users experimenting with free trials (fewer than 100 accounts are long-term customers of any AAS).

Table 2.6 also breaks down the active customers into short-term and long-term categories. For Insta* and Boostgram — both of which rely on reciprocity — we define long-term users as those who participate for more than seven consecutive days, strictly longer than the length of the free trial period (Section 2.4.2).[2] For Hublaagram, the collusion network, we define long-term users as those who request service for more than four consecutive days. All other users are considered short-term users who only briefly engage with the services and then disappear.

One third of customers of both Insta* and Boostgram are long-term, while nearly half of Hublaagram users are long-term. Having a significant fraction of long-term uses is not surprising since, again, they offer extended services without a fee. And by far most of the actions attempted by the services come from long-term users. For Insta* and Boostgram, 91.6% and 89.7% of actions are from long-term users, and for Hublaagram it is 92.3%.

User Stability. Are AASs growing in popularity over time, or does the market appear to be saturated? Over the course of three months, we examine the rate at which new long-term users appear in each service (birth rate), the rate at which long-term users appear to have dropped out of the service (death rate), and the daily number of active long-term users in each service. Both Boostgram and Hublaagram shrank slightly over our measurement period, losing a small percentage of long-term users over time (death rate slightly higher than birth rate). In contrast,

[2]If an Insta* customer pays for exactly seven days of service but does not use the free trial in our measurement period, then our methodology incorrectly labels the customer as a short-term account. We expect such behavior to be infrequent, though.

Table 2.7. The operating location for each AAS as reported on their Web site and the ASNs from service activity originates.

Service	Operating Country	ASN Location
Insta*	Russia	USA
Boostgram	United States	USA
Hublaagram	Indonesia	GBR, USA

Insta* grew in size by more than 10% and the number of active long-term users per day steadily increased over the period.

Similarly, we measure the probability that a new AAS user will become a long-term user within the month they begin service. We find the long-term user conversion rate in the first month of service to be stable across our measurement period for each AAS, although the rates vary across services: the conversion rate for Boostgram is 12%, Insta* is 21%, and Hublaagram is 37%. It is not surprising that Boostgram has the lowest new long-term user conversion rate since they have the most expensive service (Table 2.2).

Service and Customer Location Where are customers geographically located? For each AASs we compare the country location of the service with the location of its customers. We determine the location of a service using geographic information reported on its Web site and the ASNs from which service activity originates. We define the location of an Instagram account to be the most frequent country used to login to the account, as determined by Instagram's IP geolocation system.[3]

Table 2.7 shows the locations of each AAS, and Figure 2.2 shows the countries that account for 5% or more of the user population. For each AAS, the advertised country is also where the largest number of Instagram accounts are located. Insta* has most of their users in the "other" category, which we suspect is an artifact of undiscovered franchised services around the world (Section 2.3.3).

[3]Note that, while AASs might affect their customer's geolocation by logging in to their Instagram accounts, they do so infrequently.

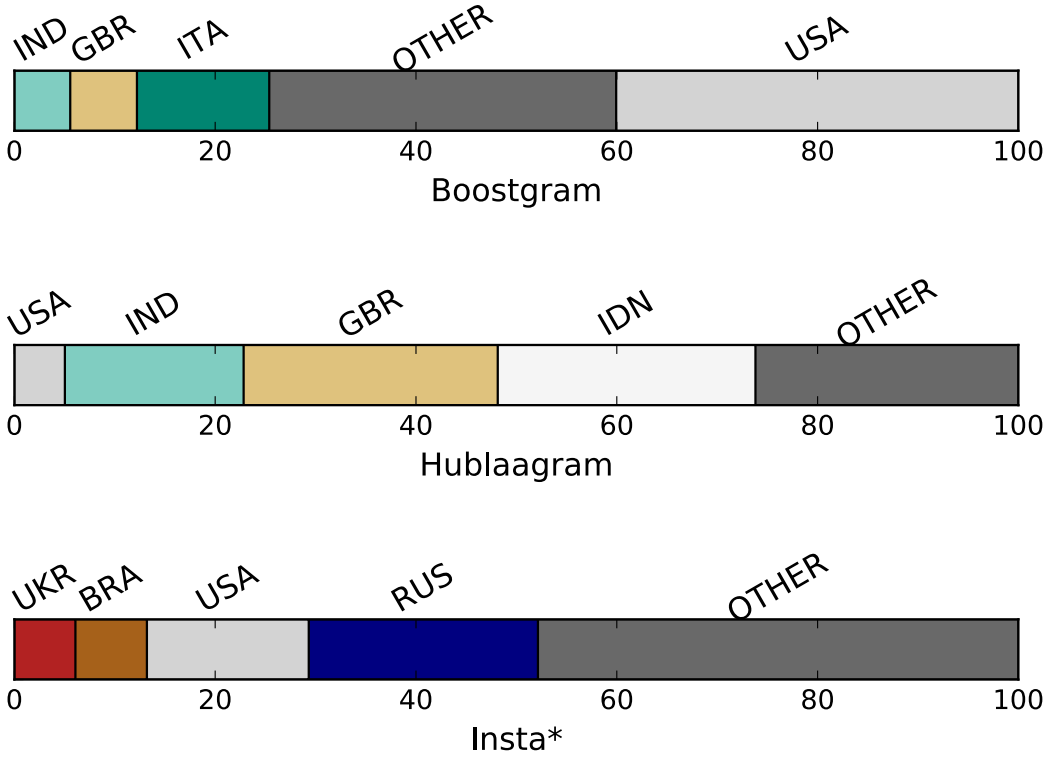

Figure 2.2. Percentage of AAS customer Instagram account locations by country. "OTHER" includes all countries that contribute less than 5% to the total distribution.

2.5.2 Revenue

To estimate the gross monthly revenue of each service we classify the accounts participating in each service into free and paid accounts.

For Reciprocity Abuse AASs we know the account is paid when it is active in the AAS for longer than the trial period (Section 2.3.3). For each paid account we estimate the amount of money paid to the service by measuring the number of days the account is active beyond a trial period, and use the minimum paid duration as a way to convert the number of days active into money paid to the AAS. For Insta* we provide an estimated revenue range as each service (Instalex and Instazood) has a different cost and minimum service duration even though they are franchises of the same company. Table 2.8 shows our estimate of the monthly gross revenue for Reciprocity Abuse AASs. On average each service has a significant gross revenue approaching

Table 2.8. Estimated monthly gross revenue for Reciprocity Abuse AASs.

Service	Accounts	Service Fee	Revenue
Boostgram	3,016	$99/month	$298,584
Insta* (Low)	25,122	$0.34/day	$195,017
Insta* (High)	25,122	$3.15/week	$223,785

$200,000 to $300,000 per month.

For the collusion network Hublaagram, distinguishing between free and paid accounts is more challenging and requires a more detailed accounting methodology. Since customers can request free service for an unbounded number of days, we cannot distinguish between free or paid solely based on the number of days they are active as we could with the other AASs. Instead, to estimate Hublaagram's monthly gross revenue we developed a model tailored to their cost structure (Table 2.3).

To identify accounts that pay a one-time fee to not participate in the collusion network, we count those accounts that only receive inbound actions from Hublaagram and never produce outbound actions from the service. In our measurement period, 24,420 active accounts paid the one-time fee to prevent their accounts from being used in the collusion network.

There are multiple `like` services offered by Hublaagram. To identify paying customers, for each user we count the hourly median number of `likes` generated by Hublaagram across each photo on the customer's account. Using observations from paid honeypot accounts (Section 2.4.1) in Hublaagram, we know that paid customers exceed the 160 `likes`/hour rate-limit imposed by Hublaagram for free customers. Therefore, we count accounts that have ever received more than 160 `likes` in an hour on any of their photos as paid since they must have purchased one of the `like` services.

For accounts classified as paid, we then distinguish among the one-time and monthly `like` services. To identify customers that purchase one-time `likes` for a single photo, we count the number of photos that have more than 2,000 `likes` for accounts that have a daily median of fewer than 250 likes per photo. Similarly, to identify customers that pay for monthly `like`

services, we count accounts that have a median value of `likes`/photo that fall within the various tiers of Hublaagram's service options (e.g., we estimate an account with a median `likes`/photo ratio in the 250–500 range to be paying $20/month). We identify just 182 users who paid for one-time `likes`, while 31,901 paid for one of the monthly `like` services.

Lastly, when a customer visits Hublaagram's Web site to request free actions, they may be shown multiple advertisements that generate additional revenue for the service. The site publishes pop-under advertisements[4] from the PopAds network [71]. To increase their ad revenue, Hublaagram's Web site occasionally shows visitors pop-under advertisements on *every* Web site interaction (e.g., clicking on a radio button triggers an advertisement in a new window).[5] Hublaagram provides ≈ 40 `follows` or ≈ 80 `likes` per free service request, limited to two requests per hour. We estimate the number of advertisement impressions by counting multiples of 40 `follows` or 80 `likes` performed by Hublaagram. We conservatively exclude paying customer accounts in this analysis as we are unable to differentiate paid or free `like` actions, and assume that for each request only a single advertisement was shown since we do not know how the customer interacts with the Web site. Based on PopAd's revenue model, we estimate that for every 1,000 impressions (CPM) Hublaagram receives between $0.60 and $4.00 since their customers are located around the world (Figure 2.2) and geolocation affects CPM [7, 23, 71].

Table 2.9 lists the number of paid Hublaagram accounts in each of the service categories and their contribution to overall Hublaagram's revenue.[6] Considering Hublaagram's large user base, the fraction of paid users is small. While Hublaagram had over a million active users within the measurement period, and half of them were long-term users, only about 5% of users paid fees for some kind of service beyond the free options that Hublaagram offers. Even so, Hublaagram still has an impressive estimated gross revenue of well over $800,000 per month. Most of

[4]Pop-under ads typically appear when closing a Web page.

[5]Hublaagram's Web site shows between 1–4 pop-under ads per free service request.

[6]Fewer than 20 customers mapped to the 5,000 or 10,000 one-time `like` service categories, and we exclude them from Table 2.9 since their revenue contribution is negligible.

Table 2.9. Gross revenue estimates for Hublaagram. The "No outbound" service has a one-time fee for the lifetime of the account, and the remaining services have monthly fees.

Service	Accounts	Fee	Revenue
No outbound	24,420	$15	$366,300
Total One-Time Revenue			$366,300

Service	Count	Fee	Revenue
Ads Shown			
Low CPM	5,769,537	0.06¢	$3,461
High CPM	5,769,537	0.4¢	$23,078
Likes Once			
2,000	182	$10	$1,820
Likes / Photo			
250 – 500	11,249	$20	$224,980
500 – 1,000	18,009	$30	$540,270
1,000 – 2,000	2,488	$40	$99,520
2,000 – 4,000	155	$70	$10,850
Total Monthly Revenue		$880,901 – $900,518	

Hublaagram's monthly revenue derives from customers paying for 250–1,000 `likes`/photo per month, while few customers purchase one-time `likes` for a single photo (reflecting how poor a bargain that option is). Similarly, while many ads are shown, we estimate that the resulting ad revenue is dwarfed by the other revenue sources.

Interestingly, users *do* care about not receiving fake outbound actions from other accounts in the collusion network, and are willing to pay for preventing it. Of the active accounts in our observation period, such users collectively paid Hublaagram more than $350,000 in one-time fees.

A related question is if the majority of monthly AAS revenue is generated from customers that pay for service only once, or ones that renew. Table 2.10 shows the fraction of new paid customers versus customers that have paid for service before. Across all services, the majority of gross revenue is generated from AAS customers who repeatedly pay for service.

Table 2.10. Breakdown of revenue between new and existing paying customers for each AAS over one month.

Service	New	Preexisting
Insta*	31.4%	68.6%
Boostgram	10.8%	89.2%
Hublaagram	16.5%	83.5%

Table 2.11. Action types performed from each AAS over a 90-day period. We normalize each value by the total number actions performed by each service.

Action	Insta*	Boostgram	Hublaagram
Likes	30.8%	64.0%	63.0%
Follows	38.6%	19.3%	35.3%
Comments	5.6%	0%	1.7%
Unfollows	25.0%	16.7%	0%

2.5.3 Activity Generated

We now analyze the *actions* performed by each AAS to understand which types are most popular among users, and how Reciprocity Abuse AASs target specific kinds of users to obtain better organic reciprocation rates.

Table 2.11 shows the proportion of action types performed by each AAS throughout the measurement period. `Likes` are the most requested action for Boostgram and Hublaagram, 1.8–3.4× more popular than `follows`. Insta* customers request more `follows` to `likes` (1.3×). Across all AASs, `comments` and `posts` are infrequent, suggesting that customers of these AAS either acquire these actions through other means, or do not consider them as valuable. The Reciprocity AASs perform a significant number of `unfollows`, which users can optionally request to happen automatically after a `follow`.

Reciprocity AASs depend on general Instagram users to generate reciprocating `follows` and `likes` to their customers' requests. As a result, if these services can target Instagram users who are more likely to reciprocate, they can more easily meet their customer demands. To evaluate whether Reciprocity Abuse AASs have any biases in the accounts that they target, we

compare accounts targeted by actions from AASs with accounts from all of Instagram as a baseline. Specifically, we compare the following and follower counts of a random sample of $1,000$ accounts that received an action from AASs with a random sample of $1,000$ from all Instagram accounts that receive actions during our measurement period.

For both metrics we see differences in the account populations. Figure 2.3 shows a CDF of the number of Instagram accounts followed by the accounts in each sample (account out-degree). For example, the median AAS accounts have a higher out-degree than a random Instagram account: Boostgram accounts follow 684 other Instagram accounts and Insta* accounts follow 554.5, while the median sample of all of Instagram accounts follow just 465. Similarly, Figure 2.4 shows a CDF of the number of followers of the accounts in each sample (account in-degree). By this metric, the distributions have even more pronounced differences: The accounts targeted by the Reciprocity AASs have significantly fewer followers than the broader Instagram population. Boostgram and Insta* accounts are followed by just a median of 498 and 384 accounts, respectively, whereas the median sample of all Instagram accounts are followed by 796 accounts.

These results indicate that the Reciprocity AASs do have a selection bias in the accounts that they target, selecting for accounts with higher out-degree and much lower in-degree to increase the likelihood of a reciprocated action. In other words, accounts targeted by the AASs are already inclined to follow other users, but have far fewer followers themselves and, as a result, are presumably more open to reciprocating when targeted.

2.6 Interventions

Having characterized AAS from a user perspective and as business entities, we subsequently actively engage with the abusive services by deploying countermeasures. Our goal is not to completely disrupt the AASs immediately, but rather we start by evaluating how AASs react to interventions. This understanding can then provide insight for improving operational abuse

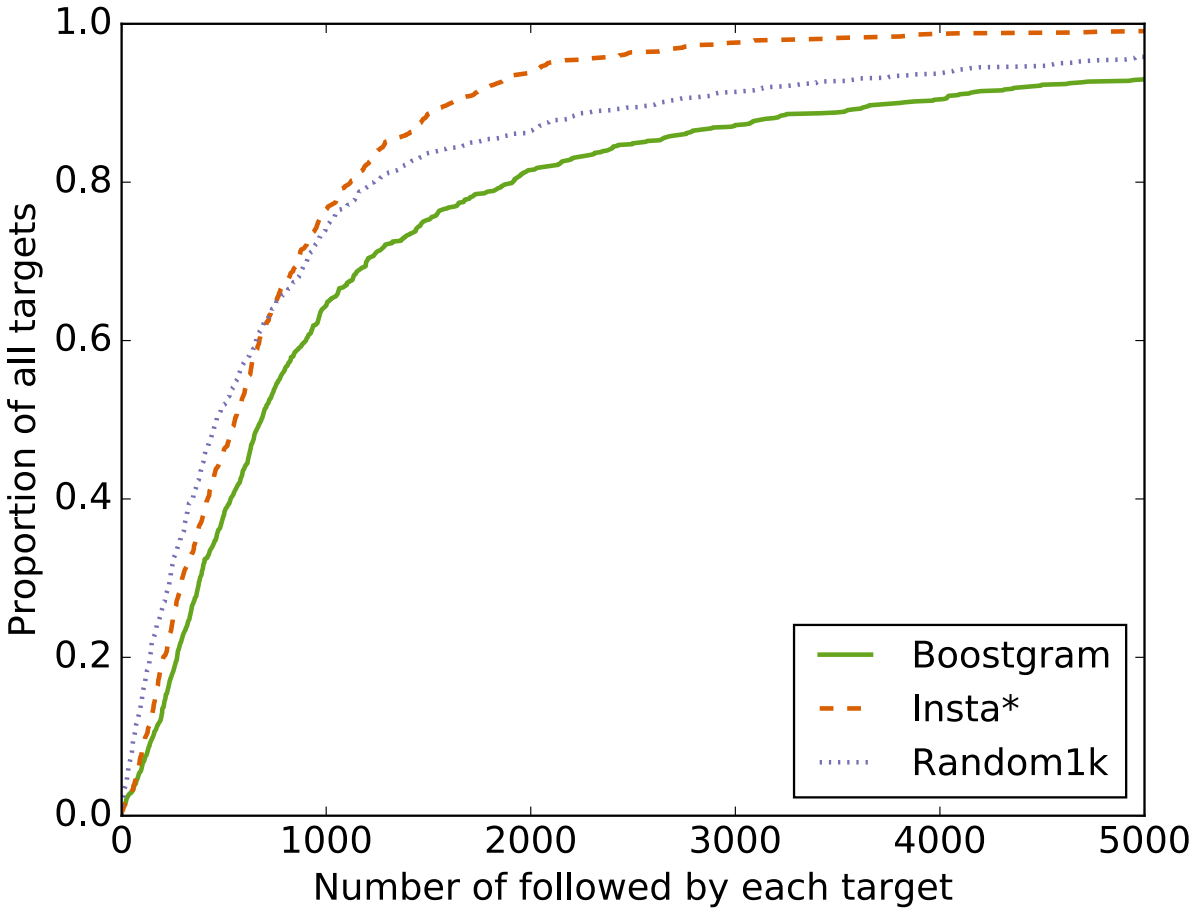

Figure 2.3. CDFs of the number of users followed by each target for three samples of accounts: 1,000 random accounts targeted by the two Reciprocity AASs, and 1,000 random Instagram users.

detection and prevention systems. While Instagram is in a position to identify all AAS customer accounts, blocking these accounts is not a desirable outcome since Instagram users still use them to initiate legitimate actions that should not be blocked (even while they are also enrolled in an AAS). Additionally, as our interventions show in Section 2.6.3, AASs quickly attempt to evade interventions. As such, we derive a new signal for performing countermeasures (Section 2.6.2), rather than relying on the signals used to identify AAS customers in the first place. We perform two interventions, first on a narrow set of AAS activity over a six-week period, and a second on a broad set of AAS activity over a subsequent two-week period.

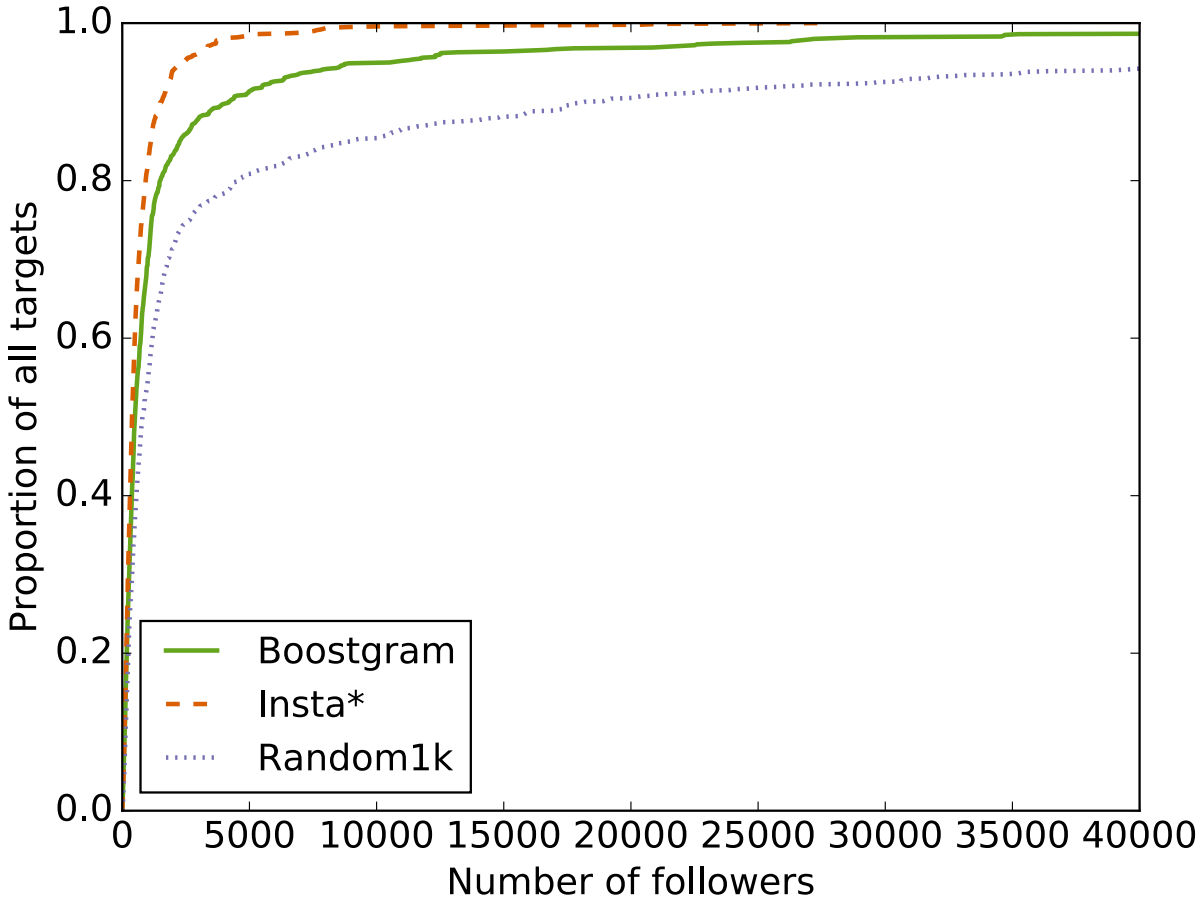

Figure 2.4. CDFs of the number of followers for a random sample of 1,000 targets selected by two third-party applications compared to a sample of 1,000 Instagram users.

2.6.1 Countermeasures

Instagram has a variety of options for reducing or disrupting the impact of an abusive action, and we experiment with two. Each countermeasure response comes with a trade-off between its effectiveness at disrupting abuse, and the ease with which an adversary detects the intervention.

Synchronous Block. When blocking AAS actions, the actions are not successful and do not reach users of Instagram. Such a countermeasure directly undermines the perceived value of using an AAS. At the same time, though, the transparent aspect of the synchronous response serves as an oracle of what actions Instagram can detect as abusive. The AAS can use this oracle

to easily test and possibly adjust their strategy for delivering their actions to accommodate or sidestep the countermeasure within a short period of time.

Delayed Removal of Follows. With the delayed removal countermeasure, `follows` from accounts used by AASs are initially successful but then are removed by Instagram one day after taking place. The deferred nature of the delayed response helps mask the countermeasure as it is more difficult for AASs to realize their actions are being detected. Note that we only apply this countermeasure to `follow` actions, as it was not possible to apply a delayed countermeasure on `likes`.

2.6.2 Identifying Eligible Actions

As with all anti-abuse measures, from spam filtering to anti-virus, one must balance the value provided in addressing abusive behavior against the unintentional misclassification of a benign action. Thus, while AASs are insidious in undermining the confidence in the integrity of the content being posted, so too must we consider and be sensitive to users whose legitimate actions might be inadvertently blocked or removed. To this end, we have carefully designed our interventions to minimize these risks; throughout the duration of our experiments we identified a handful of false positives and these were remediated manually.

In particular, we start by focusing on actions from the small number of ASNs that the AASs use. Then we define a per-account daily activity threshold for each ASN, and only actions above that threshold are candidates for a countermeasure. The threshold is defined in terms of legitimate activity, so activity by an account above the threshold strongly suggests abusive behavior by that account. Specifcially, we track the number of outbound actions from Instagram accounts used by the Reciprocity Abuse AASs, and we track the number of inbound actions from accounts used by the Collusion Network AAS. We use the same methodology from Section 2.5 combined with paid honeypot accounts to track AAS activity and reactions to countermeasures.

Note that we compute the activity thresholds differently across ASNs since some ASNs have only AAS traffic while others have benign user activity blended in. For ASNs with both

AAS and benign traffic, we measure the daily 99th percentile of `likes` and `follows` produced by Instagram accounts that are not participating in AASs. Since accounts involved in AASs produce significantly more actions than non-AAS accounts, using the daily 99th percentile of non-AAS activity represents an upper bound of 1% false positives. For ASNs with only AAS traffic, we use a threshold of the daily 25th percentile of actions since there is no legitimate user traffic from those ASNs.

We computed the activity level thresholds at the start of each experiment and did not change them to prevent an adversary from affecting the false positive rate. Throughout both experiments we actively monitored complaints to Instagram from users who *might* be affected by our experiments. We received only a handful of complaints from legitimate users who were inadvertently impacted which we worked to address. In contrast, we also monitored complaints to the AASs from their customers, and some of the interventions generated highly voluble complaints.

2.6.3 Narrow Interventions

In our first intervention we evaluate how AASs react to the countermeasures from Section 2.6.1 when they are continuously applied for six weeks to the same subsets of AAS customers. To define different sets of Instagram accounts that may receive a countermeasure response, we deterministically partition Instagram accounts into 10 equally-sized bins. We assign separate bins for each countermeasure response (block and delay) and another for a control. By partitioning Instagram accounts into 10 bins, each bin contains at least 5% of long-term customers (for each AAS) that produce actions eligible for a countermeasure (Section 2.6.2). Throughout a six-week period in 2017, we continuously apply each of the two countermeasure responses to all eligible AAS actions that go above the daily activity threshold when the Instagram account is within a particular countermeasure bin. Accounts in the control bin never receive a countermeasure even when actions go beyond the activity threshold. In total, this experiment applies countermeasures to at most 20% of the customers in each AAS.

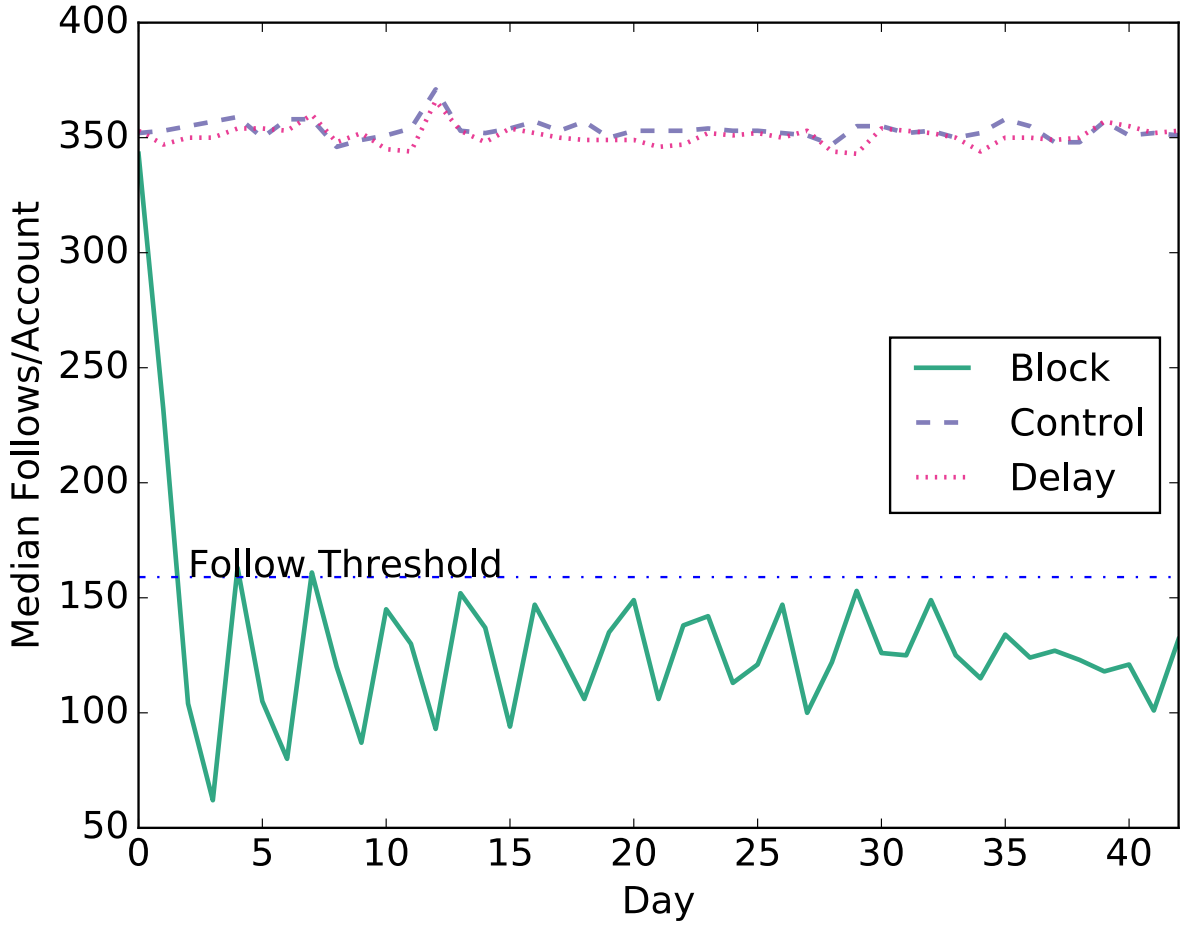

Figure 2.5. Median follows per user each day participating in Boostgram. We show the countermeasure threshold as a dashed line, and the median actions for both users who are blocked by countermeasures, and in our control (no countermeasures)

When applying the countermeasures to `follow` actions, all of the AASs react similarly. Figure 2.5 shows Boostgram activity as a representative example. Each curve shows the median number of actions per Instagram account in each countermeasure bin and the control bin for each day of the six-week period of the experiment. The dashed "Follow Threshold" line shows the threshold above which the countermeasure affects actions in Instagram. The service reacts immediately to blocking `follows`, dropping the number of actions below the threshold and probing it thereafter. Boostgram (and the other services) clearly detect that blocking is taking place, and the reaction patterns across services strongly suggests that it is an automated process; indeed, we found an openly available implementation of one of these services with block

detection logic. Countermeasures that provide a strong signal to the services unfortunately enable them to adapt, and adapt quickly.

Even more interesting, though, is that the services *do not* react to delayed removal of `follows`, even though the countermeasure undoes all of the activity one day later. Ironically, delayed forms of countermeasure satisfy both sides: the services successfully perform `follows` and continue on apparently unaware that the countermeasure cleans them up shortly afterwards as if they never happened. (Customers of the services, though, lose out.) Blocking and delayed removal both ultimately have the same benefit to Instagram— `follow` actions are truncated to the threshold — but blocking provides a signal to services, while delays do not.

Only Hublaagram reacts when we apply the countermeasures to `likes`, presumably since `likes` are its primary source of income. Figure 2.6 shows the proportion of daily `likes` above the threshold that the countermeasures can affect. Again, Hublaagram only reacts to blocking and, because blocking is straightforward to detect, it is able to drop its `like` activity and discover the threshold under which blocking does not take place. Hublaagram does take three weeks into the intervention period to react, perhaps because it had to implement blocked `like` detection.

2.6.4 Broad Interventions

Our first intervention applied each countermeasure to a narrow 10% of users, perhaps so narrow that the services did not fully notice or react to countermeasures (delay removal in particular). Consequently, our second intervention applied the delay and block countermeasures broadly to 90% of the AAS user accounts, keeping the same 10% bin of control accounts as before. In this experiment we apply the delayed removal for one week, and then blocking for another.

The reactions of the AASs to the broad intervention are similar to their reactions for the narrow intervention. As representative behavior, Figure 2.7 shows the proportion of daily Boostgram `follows` above the activity threshold that are subject to countermeasures. The control bin, with 10% of accounts, appropriately has 10% of the actions above the threshold throughout.

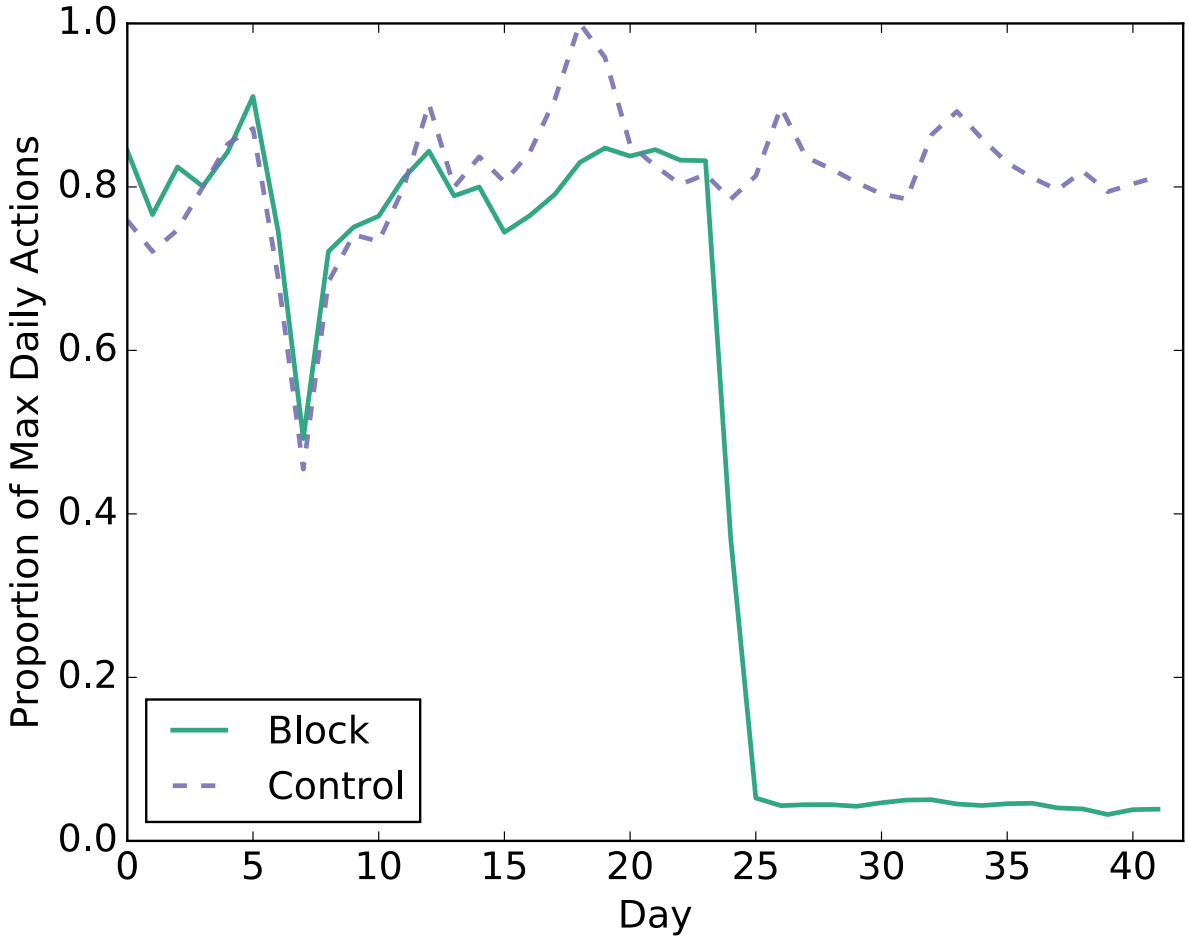

Figure 2.6. The proportion of Hublaagram likes each day that are eligible for a countermeasure. We noticed at around the third week the service makes a strict adjustment significantly reducing the number of eligible likes.

In the first week we deploy the delay countermeasures to the remaining 90% of accounts, again with no reaction by Boostgram — even though the countermeasure now applies to actions above the threshold for nearly all of their users. We then replace delay with the block countermeasure for the second week. As with the narrow intervention, Boostgram detects that their `follows` are being blocked and scales back their actions to the threshold.

Epilogue. The broad intervention remained active, continuing to block `likes` and delay `follows` above the activity threshold for additional months. Since the services immediately detected blocked actions, all AASs eventually moved their `like` traffic to different ASNs — one

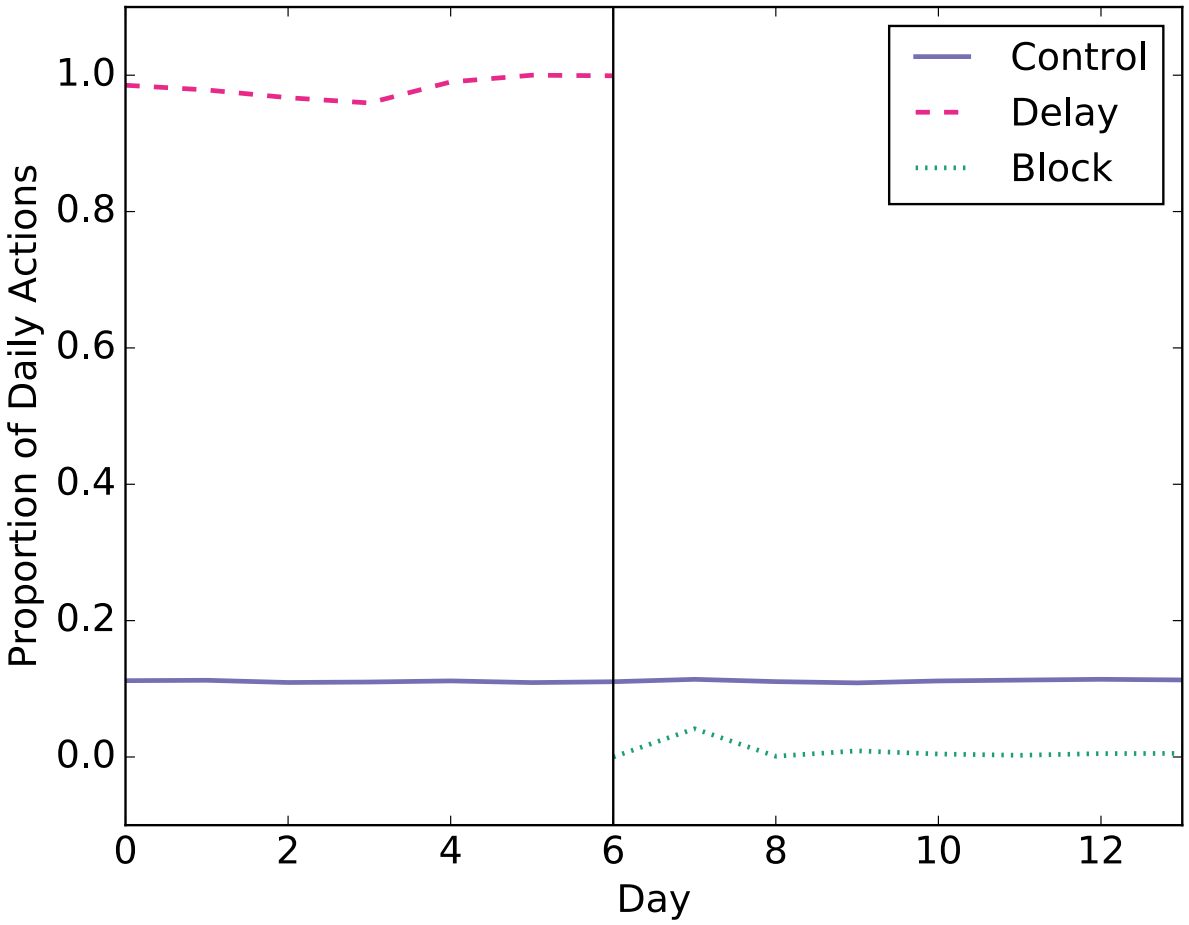

Figure 2.7. Proportion of Boostgram follows eligible for countermeasures each week during the experiment. On day 6, we switched the countermeasure response from delay to block, shown by a vertical line.

of them going so far as to use an extensive proxy network to drastically increase IP diversity. As a result, the `like` actions from the AAS were subsequently out of reach of the blocking countermeasure we employed, underscoring the risks of a countermeasure so easily detected.

After a few months, Hublaagram, unable to produce sustainable unblocked actions, stopped accepting customer payments by listing all offered services as "out of stock". Insta*, on the other hand, eventually moved their `follow` actions back into the original ASN in which we applied the delayed intervention.

2.7 Conclusion

Social networks such as Instagram attract abuse because they provide a mechanism for attracting and focusing the attention of large groups. Whether for social or economic reasons, a range of users are interested in artificially inflating their standing in such networks — paying to acquire thousands of `follows`, pervasive `likes` of their photos and so on. Simplistic approaches to manipulate social standing (i.e., using fake accounts) can be readily detected and thus sophisticated services have emerged that remotely "drive" the accounts of their customers to manipulate their social standing in a manner more likely to appear organic. We have identified two common techniques used to achieve this end on the Instagram network — driving outbound `follows` to attract reciprocal `follows` (reciprocity abuse) and laundering social actions across a network of customer participants (collusion networks). We've shown that services using these techniques have been successful in attracting and maintaining long-term customers generating per-service revenues between $200k-900k per month. Finally, we have shown through controlled experiments that blocking such services, while effective in the short term, quickly drives adaptation and can make it difficult to amortize the cost of developing accurate abuse classification. Consequently, from the standpoint of protecting non-abusive users from artificial content, a more effective long-term strategy can be built on deferred interventions (e.g., removing synthetic actions after at a future point). Such approaches greatly increase the "debug time" for services seeking to reverse engineer how they are being detected and are less likely to drive the customer complaints that incentive services to pursue such adaptations.

Acknowledgements

Chapter 2, in part, is a reprint of the material as it appears in *Proceedings of the ACM Internet Measurement Conference (IMC)*. Louis F. DeKoven, Trevor Pottinger, Stefan Savage, Geoffrey M. Voelker, Nektarios Leontiadis, 2018. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Measuring Security Practices and How They Impact Security

Thus far, we have presented systems that disrupt well defined types of compromise (i.e., malicious extensions, and underground services) from the vantage point of online social networks. In our final study, we measure the prevalence of security "best practices" and device behaviors (e.g., the types of web sites visited), along with how they correlate to device compromise. In particular, we develop a system to empirically evaluate how recommended security "best practices" and behaviors relate to device compromise by analyzing the passive network traffic of over 15,000 laptop and desktop devices using a university's residential network. Unlike the vantage point of online social networks which is constrained to social network actions, passive network traffic contains a broader set of device actions (e.g., software updates, social network activity, etc.). However, this activity is encoded within network traffic, and fine-grained detail into the context of each action is commonly obscured by encryption. Consequently, network traffic requires considerable processing to identify device features. To address these challenges, we develop network traffic signatures that enable the detection of a range of security practices and behaviors.

We describe the design and implementation of a system that builds per-device feature models identifying security practices and behaviors. Next, we present a methodology using quantitative measurement, network traffic signatures, and HTTP User-Agent strings to isolate

laptop and desktop devices on the network. By combining our per-device feature models with operational security logs used to identify compromised devices, we develop a data-driven understanding around features that correlate with security outcomes. Lastly, we describe a logistic model that ranks the features we detect in terms of their ability to predict security outcomes relative to one another. Analyzing months of longitudinal data we find that a number of recommended security "best practices" are followed, however, they do not negatively correlate with device compromise. Most positively correlated with device compromise is the types of web site devices visit (e.g., adult content, video games, etc.), and the volume of traffic devices produce. Subsequently, using an interpretable logistic model we find that behavioral features such as visiting web sites related to gaming and illegal content are relatively more useful for distinguishing compromised devices.

3.1 Introduction

Ensuring effective computer security is widely understood to require a combination of both appropriate technological measures and prudent human behaviors; e.g., rapid installation of security updates to patch vulnerabilities or the use of password managers to ensure login credentials are distinct and random. Implicit in this status quo is the recognition that security is not an intrinsic property of today's systems, but is a byproduct of making appropriate choices — choices about what security products to employ, choices about how to manage system software, and choices about how to engage (or not) with third-party services on the Internet. Indeed, the codifying of good security choices, commonly referred to as policy or "best practice" has been a part of our lives as long as security has been a concern.

However, establishing the value provided by these security practices is underexamined at best. First, we have limited empirical data about which security advice is adopted in practice. Users have a plethora of advice to choose from, highlighted by Reeder et al's recent study of expert security advice, whose title — "152 Simple Steps to Stay Safe Online" — underscores

the variability in such security lore [78]. Clearly few users are likely to follow all such dicta, but if user behavior is indeed key to security, it is important to know which practices are widely followed and which have only limited uptake.

A second, more subtle issue concerns the efficacy of security practices when followed: Do they work? Here the evidence is scant. Even practices widely agreed upon by Reeder's experts (e.g., keep software patched) are not justified beyond a rhetorical argument. In fact, virtually all the most established security best practices — including "use antivirus software", "use HTTPS/TLS", "update your software regularly", "use a password manager", and so on — have attained this status without empirical evidence quantifying their impact on security outcomes. Summarizing this state of affairs, Herley writes, "[Security] advice is complex and growing, but the benefit is largely speculative or moot", which he argues leads rational users to reject security advice [36].

To summarize, our existing models of security all rely on end users to follow a range of best practices. However, we neither understand the extent to which they are following this advice, nor do we have good information about how much this behavior ultimately impacts their future security.

This paper seeks to make progress on both issues—the prevalence of popular security practices and their relationship to security outcomes—via longitudinal empirical measurement of a large population of computer devices. In particular, we monitor the online behavior of 15,291 independently administered desktop/laptop computers and identify per-device security *behaviors*, for instance: what software they are running (e.g., antivirus products, password managers, etc.), is the software patched, and what is their network usage (e.g., does the machine contact file sharing sites), as well as as concrete security *outcomes* (i.e., whether a particular machine becomes compromised). In the course of this work, we describe three primary contributions:

- Large-scale passive feature collection. Our results are based on passive monitoring which is what allows large-scale measurement. This has required us to develop and test a large

dictionary of classification rules to indirectly infer software state on monitored machines (e.g., that a machine is running antivirus of a particular brand, or if its OS has been updated). In addition, to ensure that features are consistently associated with particular devices, we describe techniques for addressing a range of aliasing challenges due to DHCP and to DNS caching.

- Outcome-based analysis. We use a combination of operational security logs and network intrusion detection alerts to identify the subset of machines in our data set that are truly compromised. This allows us to examine the impact of adopted security practices in terms of individual security outcomes and with respect to concrete time periods surrounding the likely time of compromise.

- Prevalence and impact of security practices. Using our data we establish the prevalence of a range of popular security practices as well as how these behaviors relate to security outcomes. We specifically explore the hypotheses that a range of existing "best practices" are negatively correlated with host compromise or that "bad practices" are positively correlated. We consider both behaviors that could *directly* lead to compromise and those which may *indirectly* reflect a user's attentiveness to security hygiene.

Finally, while we find a number of behaviors that are positively correlated with host compromise, few "best practices" exhibit the negative correlations that would support their value in improving end user security.

3.2 Background

This study follows a large body of prior work that empirically relates user activity to various risk factors, which we highlight in five categories below.

Small scale studies of individuals. In 2008, Carlinet et al. [12] analyzed three hour long packet traces of ADSL customers (covering between 900 and 200 customers) and correlated

hosts that experienced at least one Snort IDS alert with other factors. This revealed a relationship between those machines raising alerts, and their use of the Windows operating system as well as heavy web browsing habits. Our study is similarly based on passive network data collection, but we operate at a significantly larger scale (in number and diversity of hosts as well as duration) and we also explicitly try to control for a range of confounding factors.

Aggregate studies of user behavior.

Others have studied risk factors in aggregate across large organizations. Notably, Yang et al. [57] correlated publicly declared data breaches and Web site hacks with external measurements (e.g., misconfigured DNS or HTTPS certificates). They found that evidence of organizational failures to police security is predictive of attacks. Xiao et al. [98] similarly showed by user patterns of security activity can be a predictor of future malware outbreaks in an ISP.

Web access behavior. Other researchers have investigated how a user's web browsing habits reveal risk factors. Levesque et al. [53] monitored web browser activity for 50 users over four months and found that the likelihood of visiting a malware hosting site was correlated with the other kinds of sites a machine visited (e.g., with peer-to-peer (P2P) and gambling sites) Canali et al. [11] replicated this study using antivirus telemetry (100,000 users), and Sharif et al. [82] describe a similar analysis for 20,000 mobile users. Both found that frequent, nighttime, and weekend browsing activity, are correlated with security risk.

Software Updates. Another vein of research has correlated poor software update habits with indicators of host compromise. Kahn et al. [51] used passive monitoring of roughly 5,000 hosts to infer software updates and used the Bothunter traffic analysis tool [33] to infer likely infected hosts based on suspicious traffic patterns (e.g., based on outbound scanning). They found a positive correlation between infection indicators and a lack of regular updating practice.

At a larger scale, Bilge et al. [6] used antivirus logs and host telemetry from over 600,000 enterprise hosts to retrospectively relate software updates to subsequent infections. They found that devices that do not patch correlate with those that were at some point infected. Finally, Sarabi et al. [79] used a similar data set of 400,000 Windows hosts and found that patching faster

provides limited benefit if vulnerabilities are frequently added into product code.

Human factors. Finally, there is an extensive literature on the human factors issues involved in relating security advice to users, the extent to which the advice leads to changes in behaviors, and how such effects are driven by both individual self-confidence and cultural norms [30, 80, 89, 75, 76, 77, 91, 92].

3.3 Methodology

We use passive network traffic monitoring to model security and behavioral practices among tens of thousands of devices that use a university residential network. Passive network monitoring has a number of advantages, enabling us to remove potential personally-identifiable information (PII) from the traffic before it is processed, to scale data collection and analysis to tens of thousands of hosts, and to implement custom network traffic analysis rules to extract fine-grained features (e.g., whether an application is being updated to the latest version).

In this section, we describe our system that converts network traffic into per-device models comprised of features associated with security and behavioral practices. We start by describing the steps we take to ensure the privacy and security of the network data we process. We then describe in detail the three stages of our data collection and analysis system, as shown in Figure 3.1.

3.3.1 Protecting User Privacy

Due to the sensitive nature of processing raw network traces, we have taken a number of steps to ensure the privacy and security of the data we use in this study.

Institutionally, we had extensive discussions with our campus network operations and security groups about this study, both about the operational mechanics of mirroring network traffic from the campus switches and about the security and privacy issues of processing raw packet data. As one of their motivations for engaging, the campus security group in particular was interested in the outcome of this research as a way to provide more empirical insight into

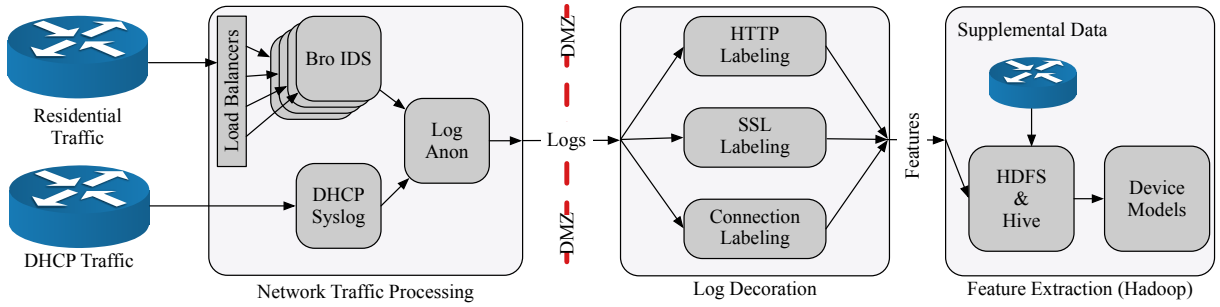

Figure 3.1. System architecture overview. Network traffic is first processed into logs and anonymized. The next stage replays the network traffic logs to extract further information and label each connection with MAC address information. The decorated logs are then stored in Hive where they are labeled with security incidents, security practice features, and behavioral features. Lastly, device models are created for analysis.

security risks and outcomes of devices using the campus network.[1] We have also engaged with our campus institutional review board (IRB), and obtained approval from a campus-wide cybersecurity governance committee.

Operationally, we anonymize and secure the data we process. The raw network traffic from which features are extracted is the most sensitive data artifact flowing through our system. To minimize risks, as soon as each connection has been processed, we discard the raw content and log only metadata from the connection (e.g., a feature indicating that device X is updating antivirus product Y). To anonymize IP and MAC addresses, we use a keyed format-preserving encryption scheme [5]. We encrypt campus IP addresses and the identifying last 24-bits of each MAC address (preserving the organizationally unique identifier (OUI) to derive the network device manufacturer).[2] This encryption transform has the property that it is impossible to obtain the original address without the encryption key. Only our campus network operations is in possession of the key, and we remain ignorant of the identities of the particular machines involved. Yet, the resulting data is in the same format as normal IP and MAC addresses and is consistent over time, so we can use standard trace processing and analysis tools unchanged. The

[1] Indeed, during the course of our work we have been able to report a variety of unexpected and suspicious activity to campus for further action.

[2] Thus, the IP address `192.168.0.1` may be replaced with `205.4.32.501` and the MAC address `00:26:18:a5:38:24` may become `00:26:18:b5:fe:ba`.

anonymization takes place on the server that processes the network traffic, thus only anonymized logs are ever transmitted to our analysis server (across the "DMZ" in Figure 3.1).

Moreover, we use a combination of physical and network security to restrict access to the server processing raw network traces. The server is physically located in a secure campus machine room with restricted access, and the server firewall restricts access to a few on-campus machines using multi-factor authentication. We also instrument the server with a log monitoring application [52] that reports daily activity. We use the same network security steps for the other servers that process only anonymized data, but the servers are physically located in our local machine room.

3.3.2 Network Traffic Processing

The first stage of our system takes as input 4–6 Gbps of raw bi-directional network traffic from the campus residential network, and outputs anonymized logs of processed network events at the rate of millions of records per second. To track the contemporaneous mapping of IP addresses to device MAC addresses, this stage also collects and anonymizes Dynamic Host Configuration Protocol (DHCP) syslog traffic.

Residential Network Traffic

As shown in the Network Traffic Processing stage of Figure 3.1, our server receives network traffic mirrored from a campus Arista switch using two 10G fiber optic links. In addition to load balancing, the switch filters out high-volume traffic from popular Content Delivery Networks (CDNs) (e.g., Netflix, YouTube, Akamai, etc.) resulting in a load of 4–6 Gbps of traffic on our server.

To minimize loss while processing traffic, we experimented with a number of network processing configurations before settling on the following. We use the PF_RING ZC (Zero Copy) framework [68] to move traffic from the network card directly into user-level ring buffers, bypassing the kernel. We then use the `zbalance_ipc` application from PF_RING ZC to locally

perform 4-tuple load balancing across many virtual network interfaces. Instances of the Bro (now Zeek) intrusion detection system (IDS) [69] then read from each virtual network interface, consuming and processing the network traffic into a custom log format. This configuration results in an average daily loss of 0.5% of received packets throughout our six-month measurement period.

While IDSs are typically used for detecting threats and anomalous network behavior, we use Bro to convert network traffic into logs since it is extensible, discards raw network traffic as soon as a connection is closed (or after a timeout), and is able to parse numerous network protocols [99]. We also customize the Bro output logs to record only information needed to identify security practice and behavioral features.

In total, we use the `HTTP`, `SSL`, `DNS`, and `Connection` protocol analyzers. The `HTTP` analyzer provides a summary of HTTP traffic on the network, including components such as the HOST and URI fields. The `SSL` analyzer extracts the Server Name Indication (SNI) field out of TLS connections. SNI is an extension of the TLS protocol enabled by most modern browsers, and allows a client to indicate which hostname is being contacted at the start of an encrypted connection. The SNI field is particularly useful for inferring the destination of connections that otherwise are encrypted. The `DNS` analyzer provides a summary of Domain Name System (DNS) requests and responses. Lastly, the `Connection` analyzer summarizes information about TCP, UDP, and ICMP connections.

Every thirty minutes Bro rotates the previous logs through our log anonymization tool (Section 3.3.1) that encrypts campus IP addresses. At this stage of processing, the logs contain IP addresses and not MAC addresses since DHCP traffic is not propagated to our network vantage point. After being anonymized, the logs are rotated across the DMZ to another server for further processing (Section 3.3.3).

DHCP Traffic

The server also runs a syslog collector that receives forwarded DHCP traffic from the residential network's DHCP servers. DHCP dynamically provides an IP address to a device joining the network. The IP address is leased to the device (by MAC address) for a specified duration, typically 15 minutes. Since we need to track a device's security and behavioral practices for long time periods, we utilize this IP-to-MAC mapping in later processing.

Similar to the Bro IDS logs, every thirty minutes we process the previous DHCP traffic into a (MAC address, IP address, starting time, lease duration) tuple. Then, the entire IP address and identifying lower 24-bits of the MAC address are encrypted using our anonymization tool (Section 3.3.1). The anonymized DHCP logs are then rotated across the DMZ to the Log Decoration server.

3.3.3 Log Decoration

The second stage takes as input the anonymized network event and DHCP logs, and processes them further to produce a single stream of network events associated with device MAC addresses and domain names.

Associating Flows to Devices. Our goal is to model device behavior based upon network activity over long time spans. While we identify unique devices based upon their MAC address, the network events that we trace are collected with dynamically assigned IP addresses. As a result, we must track dynamic IP address assignments so that we can map IP-based network events to specific device MAC addresses.

We use a Redis key-value store [74] to build a DHCP cache by replaying campus DHCP logs. We use the DHCP cache to assign a MAC address to the inbound and outbound IP of each connection. We consider an IP-to-MAC mapping valid if a connection takes place during the time when the IP address was allocated and the lease is still valid. In the event that there is not a valid mapping (e.g., the IP address is a non-University IP, or a the device uses a static IP), we do

not assign a MAC address to the IP.

Associating Flows to Domains. When using network activity to model device behavior, it useful to know the domain name associated with the end points devices are communicating with (e.g., categorizing the type of web site being visited). We also extract the registered domain and top-level domain (TLD) from each fully qualified domain name using the Public Suffix List [64]. Again, since the network events we observe use IP addresses, we must map IP addresses to domain names. And since the mapping of DNS names to IP addresses also changes over time, we also dynamically track DNS resolutions as observed in the network so that we can map network events to the domain names involved.

Due to our network vantage point, the DNS traffic our collection server observes generally has the source IP address of our local DNS resolver, and not the IP address of the host which will subsequently make a connection to the resolved IP.[3] Therefore, one of the steps in this stage is to build a local DNS cache by replaying the logs in chronological order and labeling the domain name of observed connections where it is not already provided (i.e., excluding HTTP and SNI-labeled connections).

We use another Redis key-value store to build a DNS cache by replaying DNS traffic. The cache tracks the mappings of each IP address to domain name at the time the IP address was observed. We consider a mapping to be valid as long as it has not expired — the log time falls between the time at which the DNS request was observed plus the response time to live (TTL) — and there is one registered domain name mapped to the IP address.

When sites use virtual hosting, it is possible that an IP address has multiple domain names associated with it. In this case, we first check if the registered domain names match (e.g., bar.bar.com and car.bar.com share a registered domain of bar.com). If the registered domains match, we label the connection using the longest suffix substring match (e.g., ar.bar.com) and set a flag indicating that the fully qualified domain name has been truncated. In the case where there is more than one registered domain with a valid mapping to the IP address, we do not use

[3]The primary exceptions are devices configured to use remote DNS resolvers.

the mapping to label connections until enough of the conflicting mappings expire such that they share a registered domain, or there is only one mapping.

User Agent. We parse HTTP user agent strings using the open-source `ua-parser` library. From the user agent string we extract browser, OS, and device information when present.

3.3.4 Feature Extraction

In the final stage of our system we store the log events in a Hive database [2] and process them to extract a wide variety of software and network activity features associated with the devices and their activity as seen on our network. The last critical feature is device outcomes: knowing when a device has become compromised. We derive device outcomes from a log of alerts from a campus IDS appliance, and also store that information in our database.

Software Features

To identify features describing application use on devices, we crafted custom network traffic signatures to identify application use (e.g., a particular peer-to-peer client) as well as various kinds of application behavior (e.g., a software update). To create our network signatures we use virtual machines instrumented with Wireshark [97]. We then manually exercise various applications and monitor the machine's network behavior for a unique signature. Fortunately most applications associated with security risk frequently reveal their presence when checking for updates. In total, we develop network signatures for 69 different applications, including OSs. For a subset of applications, we are also able to detect the application's version. Knowing application versions allows us to compare how fine-grained recommended security practices (i.e., updating regularly) correlates with device compromise.

Antivirus Software. Using antivirus software is virtually always recommended. We created network signatures for 12 popular antivirus products, seven of which are recognized as offering the "Best Protection" for 2019 [65].

Antivirus Software

1. Avast
2. AVG
3. Avira
4. BitDefender
5. ESET
6. Kaspersky
7. McAfee
8. Microsoft Security Essentials
9. Microsoft Windows Defender
10. Norton
11. Sophos
12. Webroot

Operating System. We created six signatures to identify the OSs running on devices. Since regular OS updating is a popular recommended security practice, we also created signatures to detect OS updates. While Windows and Mac OS operating system updates are downloaded over a CDN that is removed from the network traffic before reaching our system (Section 3.3.2), we can use OS version information from the host header and User-Agent string provided in HTTP traffic to infer that updates have taken place.

Operating Systems

1. CentOS
2. Debian
3. Mac OS
4. Red Hat
5. Ubuntu
6. Windows

Applications. Through a combination of network and User-Agent string signatures we detect 41 applications, including those commonly perceived as risky such as Adobe Flash Player, Adobe Reader, Java, Tor, P2P applications, and more. We also detect other popular applications, including browsers, Spotify, iTunes, Outlook, Adobe AIR, and more.

Applications

1. Adobe AIR
2. Adobe Flash Player
3. Adobe Reader
4. Airmail
5. Android Browser
6. Apple Mail
7. Chrome Browser (desktop and mobile)
8. Chromium Browser
9. Dominant Web Browser (from User-Agent strings)

10. Edge Browser (desktop and mobile)
11. Firefox Browser (desktop and mobile)
12. Internet Explorer (IE) Browser(desktop and mobile)
13. Internet Rely Chat
14. Jabber/Google Chat
15. Java
16. MSN Chat
17. Netscape Browser
18. Nokia Browser Browser
19. Opera Browser (desktop and mobile)
20. Outlook
21. P2P Ares
22. P2P Azureus
23. P2P BTwebclient
24. P2P BitTorrent
25. P2P Kazaa
26. P2P Kugoo
27. P2P Soulseek
28. P2P Thundernetwork

29. P2P Vuze
30. P2P ed2k
31. P2P edonkey
32. P2P emule
33. P2P libTorrent
34. P2P uTorrent
35. Safari Browser (desktop and mobile)
36. Samsung Internet Browser
37. Skype
38. Spotify
39. Thunderbird
40. Tor
41. iTunes

Password Managers. As password managers are frequently recommended to minimize collateral damage of leaked passwords, we also crafted network signatures for nine popular password managers [14].

Password Managers

1. 1Password
2. Dashlane

3. Keeper
4. LastPass
5. LogMeOnce
6. Password Boss
7. RoboForm
8. Sticky Password
9. Zoho Vault

Network Activity

We track a wide variety of features on network activity to quantitatively measure how much of a protocol is used (e.g., HTTP, and HTTPS), the types of content being accessed (e.g., Adult content, etc.), and when devices are most active. In doing so, we implement a set of features similar to those implemented by Canali et al. [11] and Sharif et al. [82] that focused on web browsing activity. As our data set also includes traffic beyond HTTP, we can measure additional behaviors (e.g., remote DNS resolver usage, HTTPS traffic usage, etc.).

Content Categorization. We use the IAB Tech Lab Content Taxonomy to categorize every registered domain in our data set [40]. The domain categorization was generously provided by WebShrinker [95, 22]. The IAB taxonomy includes 404 distinct domain categories [94]. We use the domain categorization to measure the fraction of unique domains each device accesses in a specific category. We also built a list of file hosting sites, and URL shortening services that we use to identify when a device accesses these types of services.

Usage Patterns. We also develop a number of behavioral features that describe the quantities of HTTP and HTTPS traffic in each TLDs, and the number of network requests made. Additionally, we develop features that quantify customized or non-standard behaviors such the

use of remote DNS resolvers, and the proportions of HTTP requests made directly to IP addresses (instead of a domain name).

Network Features

1. Percentage distinct registered domains in IAB categories
2. Total distinct HTTP TLDs
3. Total distinct HTTP registered domains
4. Total distinct HTTP FQNDs
5. Total HTTP URLs
6. Total distinct HTTP URLs
7. Total distinct HTTP URLs with host as IP address
8. Fraction of HTTP connections: in TLDs
9. Fraction of HTTP connections: day of the week
10. Fraction of HTTP connections: hour of day
11. Fraction of HTTP connections: hour of day of week
12. Total distinct HTTPS TLDs
13. Total distinct HTTPS registered domains
14. Total distinct HTTPS FQDNs
15. Fraction of distinct HTTP registered domains in TLDs
16. Fraction of distinct HTTP registered domains in TLDs: com net org

17. Fraction of distinct HTTP registered domains not in TLDs: com net org
18. Fraction of distinct HTTPS connections in each TLD
19. Fraction of distinct HTTPS connections in TLDs: com net org
20. Fraction of distinct HTTPS connections not in TLDs: com net org
21. Total distinct HTTP and HTTPS TLDs
22. Total distinct HTTP and HTTPS registered domains
23. Total distinct HTTP and HTTPS FQDNs
24. Organizational Unique Identifier (OUI) vendor
25. Percent TCP connections to local IP
26. Percent TCP connections to remote IPs
27. Percent UDP connections to local IP
28. Percent UDP connections to remote IP
29. Unique /24 networks visited
30. Unique /24 networks visited using TCP
31. Unique /24 networks visited using UDP
32. Uses a remote DNS resolver
33. Number of remote DNS resolvers
34. URL shorteners
35. Email providers
36. File hosting sites

Detecting Security Incidents

While previous work has relied on the use of blacklists or Google Safe Browsing to identify devices that expose users to potential risk, we are able to identify compromised devices with high-confidence as a result of post-infection behavior typically in the form of command and control (CNC) communication. To identify compromised devices (i.e., ones with a security incident) we use alerts generated by a campus network appliance running the Suricata IDS [85]. The campus security system uses deep packet inspection with an industry-standard malware rule set to flag devices exhibiting post-compromise behavior [72].

The IDS rules also detect network activity that might occur before a device becomes compromised (e.g., possible phishing attempts, exploit kit landing pages, etc.). Since we focus on compromised devices, we reduce the rules we consider to ones that explicitly detect post-infection behavior. We then manually remove rules that are frequently triggered, but do not indicate that a device has been compromised.

3.4 Data Set

We analyze six months of data from our passive network traffic processing system from June 2018 to December 2018. In this section we describe our approach for identifying the laptop and desktop devices for use in analyzing security risk factors, and determining the dominant OS of devices used in our analysis. In the end, our data set consists of 15,291 devices. In Table 3.1, we characterize our data set in terms of connections processed, and inbound and outbound bytes.

3.4.1 Device Filtering

The university allows heterogeneous devices on its network, including personal computers, mobile phones, printers, IoT devices, and more. Recommended security practices, however, are commonly offered for laptop and desktop computers, and we focus our analysis solely on such devices. As a result, we develop techniques to identify laptop and desktop computers among

Table 3.1. Data set characterization. Our network vantage point provides DNS request from the local resolver which includes DNS traffic from devices in this paper as well as other devices using the university's networks.

Name	Value
Date Range	June 2018 – December 2018
Total Filtered Devices	15,291
DNS Connections	17.1 B
Non-DNS Connections	1.92 B
Total Connections	19 B
Outbound Bytes	38.4 TB
Inbound Bytes	720 TB
Total Bytes	758 TB

the many other devices on the network. We remove devices that are easily identifiable, and then develop heuristics to filter remaining devices.

We first remove devices that are not active for a minimum of 14 days, and ones that never provide a major web browser's User-Agent string (removing 13.1% of all devices). For studying security practices, devices need to have a modicum degree of network activity to be able to model behavior, and devices without any web browser activity are a strong indication that they are not laptops or desktops.

Next, we use User-Agent strings to identify a device's OS [32]. Since applications are not required to provide accurate User-Agent string information, to identify a device's OS we consider User-Agent strings from major browsers, and require that a device's OS is consistent on 95% of all requests. We filter 40.8% of the total devices that we identify as having mobile or IoT OS. For the fraction of devices that fall below the 95% requirement, we remove ones that contact domains which are not regularly accessed by laptop or desktop devices[4] (4.1% of all devices).

We also compile a list of network hardware vendors used within devices other than laptops and desktops (e.g., Vizio, etc.), and remove devices with a matching OUI vendor (2.2% of devices).

[4]We manually label domains that are contacted by TVs, printers, game consoles, and iPhones. We also exclude devices that never make a single connection to any university web site.

Lastly, we filter some of the remaining IoT devices using network traffic-based heuristics. Our intuition is that most of these devices[5] will either make close to the same number of connections each day, a small number of daily connections, or connections within a limited number of /24 network subnets. We pick each threshold by manually inspecting the three network traffic distributions, and select the value corresponding to the first peak of the distribution. We remove devices that make the same number of connections each day ± 7, on-average 40 daily connections, or contact on-median 31 distinct /24 networks each week (4.2% of all devices).

To validate our device filtering heuristics in practice, we manually label a sample of 100 devices (50 laptop and desktop, and 50 that are removed). We find our filtering methodology to be sufficiently accurate: one laptop is incorrectly removed, and four mobile phones are incorrectly included.

3.4.2 Identifying Dominant OSes

Since different OSs have different risk profiles, identifying the OS used by a device is an important step. Being able to observe device network traffic makes OS identification an interesting task. The majority of devices are straightforward: using signatures of OS update events, we can immediately identify a single unambiguous OS for 79.1% of devices.

The remaining devices either have no OS update signatures, or have more than one.[6] For these devices, we use a combination of OS update signatures, OS User-Agent strings, and OUI vendor name information to identify the dominant OS of a device (e.g., the host OS with virtual machines, Windows if tethering an iPhone, etc.). We assume that devices with an Apple OUI vendor name will be using Mac OS (7.2%). We then use the dominant OS extracted from User-Agent strings to assign an OS (11.5%). The remaining 340 devices (2.1%) have both Windows and Mac OS updates. We choose to assign Windows as the dominant OS in these cases because of strong evidence of tethering, in which iTunes allows users to update their other Apple

[5] With the exception of user-directed IoT devices (e.g., Chromecasts, etc.)

[6] There are a number of legitimate reasons why a device can have more than one OS detected, including dual-booting between different OSes, using virtual machines, device tethering, etc.

Operating System	*Incidents*	*Total Devices*
Windows	538 (7.0%)	7,668
Mac OS	140 (1.9%)	7,339
ChromeOS	1 (0.5%)	205
Linux Variant	3 (3.8%)	79

Figure 3.2. Device OS classification after removing IoT and mobile devices, including the total number of devices with each OS and the number with a security incident.

devices (e.g.iPhone, iPad, etc.) using the network connection of their computer [3].[7] For each of these heuristics, we confirmed the labeling by manually checking the traffic profile of a random sample of devices.

3.5 Recommended Practices

There are a variety of security practices widely recommended by experts to help users become safer online. Prior work has explored some of these practices in terms of users being exposed to risky Web sites. Since our data includes actual security outcomes, we start our evaluation by exploring the correlation of various security practices to actual device compromises in our user population: operating system choice, keeping software up to date, Web sites visited, using HTTPS, and using antivirus software.

3.5.1 Operating System

Different operating systems have different security reputations, and as a result it is not surprising that experts have recommendations of the form "Use an uncommon OS" [78]. Part of the underlying reasoning is that attackers will spend their efforts targeting the devices with most

[7]We measure the baseline of iTunes installs across devices with only Windows to be 11.9%, whereas the install rate for these 340 devices is 67%.

common systems, so using an uncommon operating system makes that device less of a target.

In terms of device compromise, as with previous work and experience, such advice holds for our user population as well. Using the OS classification method described in Section 3.4.2, Table 3.2 shows the number of devices using major operating systems and the number of each that were compromised during our measurement period. Most devices use Windows and Mac OS, split nearly equally between the two. The baseline compromise rate among devices is 4.5%, but Windows devices are 3.9 times more likely to be compromised than Mac OS devices. The Chrome OS population is small, but only one such device was compromised.

Of course, modulo dual-booting or using virtual machines, this kind of advice is only actionable to users when they are choosing which device to use, and is no help once a user is already using a system.

3.5.2 Update Software

Among hundreds of security experts surveyed, by far the most popular advice is to "Keep systems and software up to date" [78]. In this part we explore the operating system, browser, and Flash update characteristics of the devices in our population, and how they correlate with device compromise.

Operating System

Mac OS. We start by analyzing the update behavior of devices running Mac OS. Our system labels each HTTP connection of a device with the type of operating system and its current version number, both extracted from the User Agent string. However, if a device leaves the network and returns with an updated version number in the UA string, then we cannot accurately tell when the device was updated. Thus, we only utilize devices that are absent for less than four days to bound the error on update times.

We see 7,268 (47.5%) devices that identify as Mac according to the User Agent string. Of these devices, we see at least one update for 2,113 (29.1% of all Mac OS devices). Figure 3.3

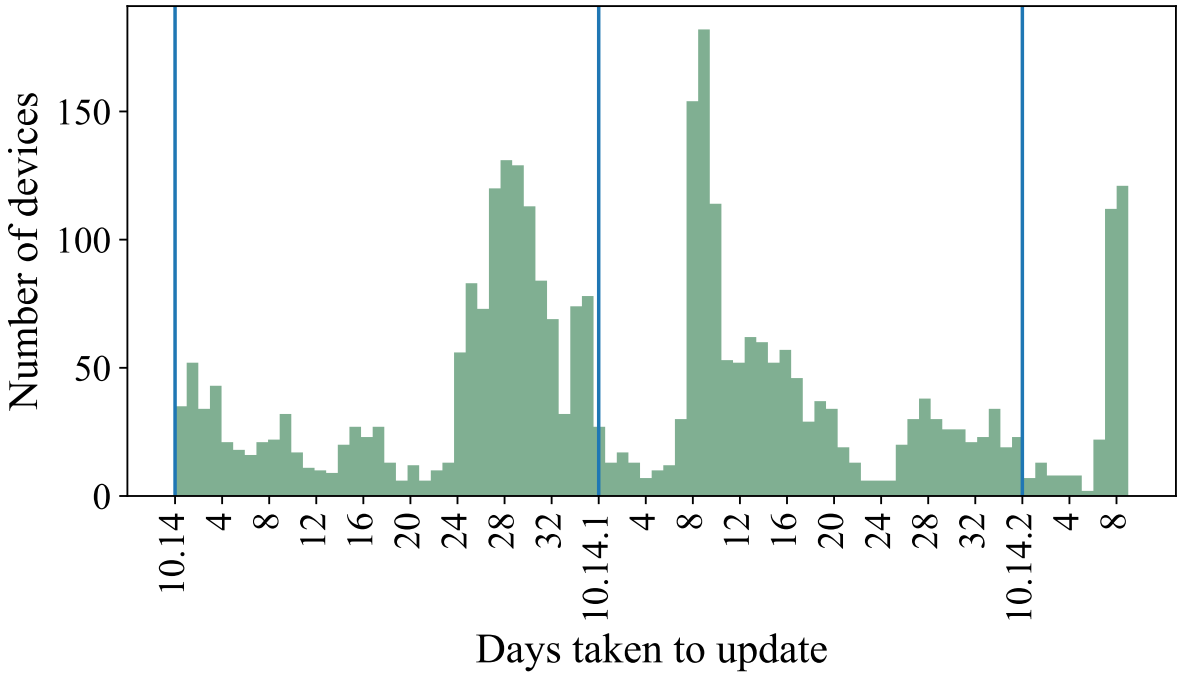

Figure 3.3. Number of days a Mac OS X device takes to update to a specific version. The version number on the x-axis denotes the day that the specified version update was published.

Table 3.2. Windows device updates deltas. We compute the average, median, P90, P95, P99, and variance of the number of days between when the update was released, and when we observe each device download the update. The devices are partitioned by those with and without a security incident.

Incident?	**# Devices**	μ	**Median**	**P90**	**P95**	**P99**	σ^2
No	5,976	2.5	0	6	15	42	59
Yes	483	2.6	0	6	14	49	62

shows the update pattern of Mac OS devices over time, anchored around the three OS updates released by Apple during our measurement period. In general, Mac OS users are relatively slow to update, anecdotally because of the interruptions and risks Mac OS updates entail.

Of these devices, 57 (2.7%) of them were compromised. Compromised devices have a mean and median update rate of 16.21 and 14 days, respectively, while their clean counterparts have a mean and median update rate of 17.96 and 16 days. However, this difference is not statistically significant according to the Mann-Whitney U test ($p = 0.13$).

Windows. For Windows we developed a signature to extract the knowledge base (KB)

number of "Other Software" updates (e.g., Adobe Flash Player, etc.).[8] Our signature detects when a device downloads the update. While we cannot verify that the update was applied, it does indicate whether the device is using the default Windows Update settings. Since it is possible to miss an update (e.g., a device may download the update while connected to a different network), we only compare devices that we see updating. We also restrict the updates considered to ones released during our measurement period since there is nothing preventing an unpatched device from joining the network.[9] We identify the update's release day using Microsoft's Update Catalog service [62].

Across devices running Windows, we see at least one update for 6,459 of them (84% of all Windows devices). Table 3.2 shows the average, median, P90, P95, P99, and variance of the number of days between when an update is downloaded and when it is released. Based upon the averages and medians, devices update with similar deltas (2.5 days and 0 days, respectively) regardless of whether they have a security incident. We confirm our hypothesis using the Mann-Whitney U test ($p = 0.052$). We also find the fraction of compromised devices that update (7.5%) to be similar in magnitude to the baseline fraction of incidents across all Windows devices (7.0%). In short, the update behavior of compromised Windows devices is little different than that of clean devices.

Web Browser

Updating the browser may be as important as updating the operating system. Browsers are also large, complex pieces of software used on a daily basis and, as with most software, these large programs have vulnerabilities. Updating is viewed as such an important process that Chrome and Firefox employ auto-updating by default [87, 28], with UI features to encourage timely updating.

As such, we explore the relationship between compromised and clean devices and browser updating behaviors. Similar to the Mac OS devices, we are able to detect the current browser

[8] An example update is https://support.microsoft.com/en-us/help/4462930

[9] We exclude updates released multiple times with the same KB number.

Table 3.3. Number of days between when an update is published and when devices update. Compromised devices update faster than their clean counterparts.

Browser	Mean, Median, # (Cmp)	Mean, Median, # (Cln)
Chrome	14.4, 15.0 (421)	15.4, 15.0 (7883)
Firefox	5.64, 3.00 (24)	9.65, 5.00 (424)

version number from the User Agent string of a device. Since browser vendors publish the dates when they make updates available,[10] we can check whether the browser on a device is out of date each time we see the device on the network. Across the measurement period, we then calculate how quickly devices update. Also similarly to the Mac OS analysis, we exclude devices that are absent from the network for more than three days.

Moreover, we only analyze the dominant browser for each device. Many devices have User Agent strings naming different browsers. While users may use different browsers for different use cases, we identify a dominant browser to remove the noise from user applications that spoof a browser in their User Agent string. Thus, we determine which browser connects to the largest number of distinct registered domains from a device and label the device with that dominant browser. We choose unique registered domains as our metric over number of HTTP connections because there are web sites and applications that "spam" the network, making the device appear to use one browser dominantly when the natural user behavior is actually coming from a different browser.

We analyzed updates for devices that dominantly use Chrome, Edge, Firefox, and Safari. Of the total devices, 10,831 (70.8%) devices use Chrome, 719 (4.7%) devices use Edge, 561 (3.7%) devices use Firefox, and 2993 (19.6%) devices use Safari. However, only 8,304 (76.7%) of the Chrome devices, 132 (18.4%) of the Edge devices, 448 (80.0%) of the Firefox devices, and 1592 (53.2%) of the Safari devices are on the network continuously (absent for less than three days). Table 3.3 shows the browsers with statistically significant differences in update time between clean and compromised devices (Mann Whitney U: Chrome $p = 4.2 \times 10^{-4}$ and

[10] During our measurement period each popular browser had at least three major updates.

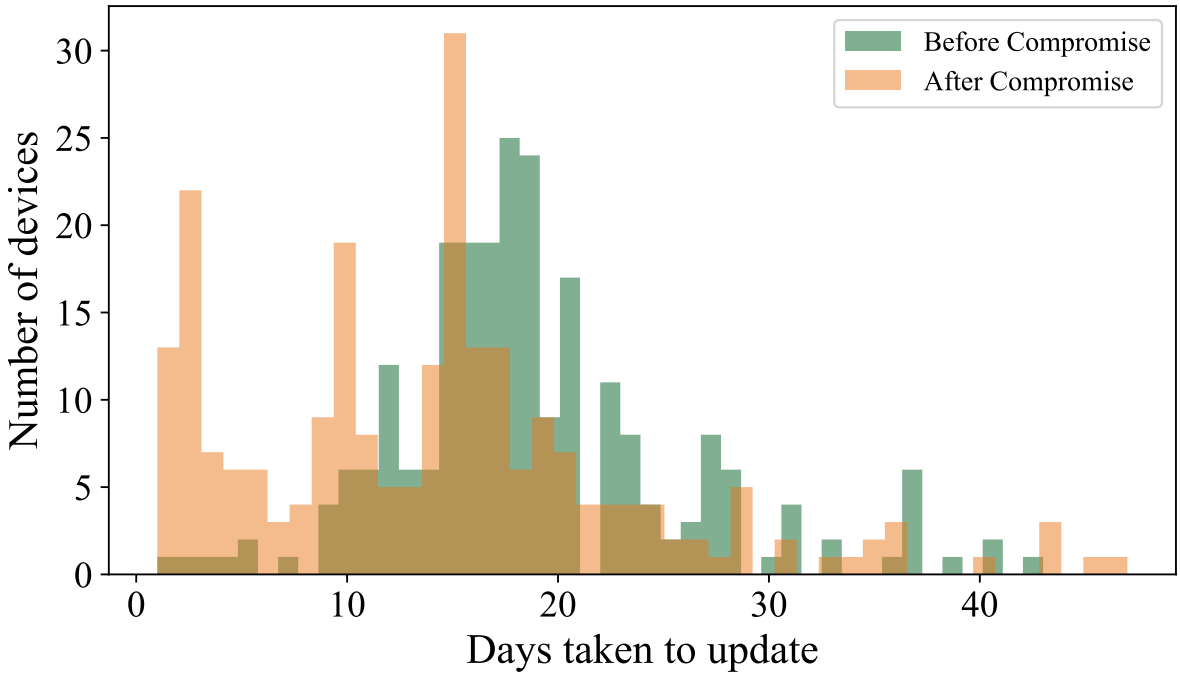

Figure 3.4. Distribution of days a device takes to update Chrome before compromise and after compromise.

Firefox $p = 0.03$).

Clean devices appear to spend more time out of date than their compromised counterparts. Examining this phenomenon in more detail, we examine the update behavior of compromised devices before and after their compromise date. We focus on devices using Chrome that have two updates spanning the compromise event (other browsers do not have a sufficiently large sample size). Figure 3.4 shows the distribution of times devices were out of date with respect to when a browser update was released for updates before and after the device was compromised. The shift in distributions illustrates that devices update faster after compromise. In more detail, devices that use Chrome have a before-compromise mean update rate of 18.9 days (18.0 median days) and an after-compromise mean update rate of 14.2 days (15.0 days median). This difference is significant, with $p = 4.8 \times 10^{-12}$ using the Wilcoxon signed-rank test.

Table 3.4. Flash Player updates on Windows devices.

Incident?	# Devices	μ	Median	P90	P95	P99	σ^2
No	1,702	4.2	1	16	20	30	53
Yes	149	3.7	1	16	21	26	47

Flash Player

The Adobe Flash player has long been associated with security risk and device compromise. The typical recommendation is to not use Flash at all, but if you do, to keep it up to date. We created a signature to detect Adobe Flash Player on Windows devices.[11] We focus on the desktop version of Flash as major browser vendors issue Flash plugin updates directly. Adobe released six updates within our measurement period, and we use Adobe's web site to identify the version and release date for each update.

Somewhat surprisingly, desktop Flash is still quite prevalent on devices. Fortunately, though, update patterns and compromise rates do not indicate that the use of Flash puts devices at greater risk of compromise. A total of 2,167 devices (28% of Windows devices) check for a Flash Player update, of which 1,851 are seen downloading an update. Table 3.4 shows the average, median, P90, P95, P99, and variance of the number of days between when an update is downloaded and when it is released. Curiously, compromised devices updated Flash slightly faster than clean devices (Mann-Whitney U test $p = 0.025$). However, the rate of compromise across devices that update Flash is 8.1%, only slightly higher than the rate across of Windows devices (7.9%) (Chi-Square $p = 0.057$). Among the 316 devices that we detect Flash Player on, but do not see updates, only 15 are compromised (4.8%). We interpret these results as a community success story. A combination of widespread awareness, aggressive updates, and focused attention have mitigated it as a significant risk factor.

We next explore why compromised devices update Flash Player more quickly. We hypothesize that a compromised device's update behavior will change after being compromised.

[11] Flash Player updates on Mac OS are downloaded over HTTPS, preventing us from crafting an effective signature.

To evaluate this claim, we compare the update patterns for compromised devices before and after becoming compromised. Out of the 149 compromised devices that update Flash, there are 60 devices (40.3%) with updates before and after their first incident. The median and average days compromised devices take to update before an incident are 6.5 and 9.9 respectively, and 0 and 1 days after becoming compromised (Wilcoxon signed-rank test $p = 1.73 \times 10^{-7}$). These results suggest that shortly after a security incident devices exhibit better Flash update hygiene.

3.5.3 Visit Reputable Web Sites

Experts recommend users to be careful in the web sites that they visit ("Visit reputable web sites" [78]), and indeed prior work has found that the category of web site users visit can be indicative of exposure to risky sites [11, 82]. We perform a similar analysis for devices that are actually compromised, and for the most part confirm that the types of sites that lead to exposure to risky sites also correlate with actual compromise.

To categorize the content devices access we use the IAB domain taxonomy (Section 3.3.4). We use the KS test with Bonferroni correction to compare the ECDFs of the fraction of distinct registered domains in each category that clean and compromised devices access, and confirm that they are statistically significant (i.e., $p < 0.001$).

Table 3.5 shows the most substantial differences between the types of content accessed, e.g., with clean devices accessing more business, advertising, and marketing content, while compromised devices accessed more gaming, hobby, uncategorized, and illegal. We note that, while previous work found that exposed devices visit more advertising domains [82], our finding of the opposite behavior can be explained by differences in methodology. The previous finding used solely HTTP requests generated by static content, while our network traces include all HTTP requests (including those generated by JavaScript) as well as HTTPS traffic.

Table 3.5. Types of content accessed more by clean or compromised devices. We show the median fraction of registered domains accessed in the category for clean (Cln.) and compromised (Cmp.) devices, and delta in median.

Clean Devices Access More			
Feature	**Cln. Median**	**Cmp. Median**	**Delta**
Business	22.36	20.14	2.22
Advertising	22.65	20.88	1.77
Marketing	12.96	11.66	1.3
Education	3.98	3.53	0.45
Content Server	6.96	6.58	0.38
Television & Video	2.18	1.89	0.29
Arts & Entertainment	2.54	2.27	0.27
Business Software	2.69	2.49	0.2
Web Design/HTML	1.39	1.24	0.15
Compromised Devices Access More			
Feature	**Cln. Median**	**Cmp. Median**	**Delta**
Computer Games	1.3	2.84	-1.54
Hobbies & Interests	2.61	3.78	-1.17
Uncategorized	26.25	26.97	-0.72
Technology	17.65	18.08	-0.43
Under Construction	5.33	5.65	-0.32
Network Security	1.43	1.65	-0.22
File Sharing	2.28	2.51	-0.23
News/Weather	2.44	2.64	-0.2
Illegal Content	0.15	0.33	-0.18
Cell Phones	0.0	0.17	-0.17
Comic Books	0.11	0.27	-0.16
Adult Content	0.36	0.51	-0.15

3.5.4 Use HTTPS

Another recommended browsing behavior is to use HTTPS when available. Of course, it is the web site itself that ultimately determines whether HTTPS can be used: if a site does not support it, users have to use HTTP. However, since prior studies on device security behavior were not able to trace HTTPS traffic, we next examine HTTPS use and network activity more generally, and then examine how it correlates with device compromise.

For each device, we count the total number of distinct fully qualified domains it contacted using HTTPS and HTTP (approximating distinct sites visited). We then consider the number of distinct FQDNs contacted just using HTTPS divided by the total (HTTPS + HTTP) as the ratio of its HTTPS use. Since a recent study of HTTPS adoption on Chrome and Firefox showed that it depends on both browser and operating system [26], we similarly categorize first by dominant browser on the device (Section 3.5.2) and then OS. Table 3.6 shows the mean and median HTTPS use across all devices, browsers, and operating systems. As a point of comparison, HTTPS use among the devices in our population is roughly consistent with the results from [26]: devices contact sites via HTTPS 78% of the time on average, and HTTPS use is lower on Windows (74–76%) compared to Mac OS (79–80%). In terms of browsers, though, in our device population Chrome does not have a distinctly higher use of HTTPS for our metric.

Table 3.6. HTTPS use among devices.

Browser	**OS**	**Mean (Median)**
Chrome	Mac OS	78.6% (79.2%)
	Linux	78.5% (79.0%)
	ChromeOS	78.1% (78.3%)
	Windows	76.2% (76.2%)
Firefox	Linux	80.8% (80.3%)
	Mac OS	80.5% (80.7%)
	Windows	78.2% (79.0%)
Safari	Mac OS	80.5% (80.7&)
Edge	Windows	73.6% (74.0%)
All Devices		77.6% (78.5%)

Turning to security outcomes, we separate the activity of devices between HTTP and HTTPS traffic and calculate their distributions for compromised and clean devices at various aggregations: number of connections to all and unique URLs (for HTTP), unique fully-qualified domain names (FQDNs), unique registered domains (RDs), and unique top-level domains (TLDs), unique /24 "subnets" (also separated into TCP and UDP traffic). To identify significant

Table 3.7. Differences in network usage for clean (Cln.) and compromised (Cmp.) devices. We use the KS test with Bonferroni correction to compare the ECDF of usage for each device type, and present the p-value along with median values for each population.

Feature	P-value	Cln. Median	Cmp. Median
Unique HTTP FQDNs	< 0.001	705	1137
Unique HTTP RDs	< 0.001	375	522
Unique HTTP TLDs	< 0.001	27	36
Unique HTTP IP URLs	< 0.001	4	57
Unique HTTPS FQDNs	< 0.001	2.5k	3.1k
Unique HTTPS RDs	< 0.001	1k	1.2k
Unique HTTPS TLDs	0.001	49.0	57.0
Unique /24 Nets.	< 0.001	3.8k	5.3k
Unique TCP /24 Nets.	< 0.001	3.6k	4.4k
Unique UDP /24 Nets.	< 0.001	20	300

differences in device behavior we use the KS test of statistical significance with Bonferroni correction. For each aggregation, Table 3.7 shows the p-value and the median values of the distributions for clean and compromised devices.

Overall, the ratio of HTTPS use is not strongly correlated with security outcomes.The connections made by compromised have similar usage of HTTPS and HTTP compared to clean devices that make similar number of connections. However, these results do show that devices that make more connections use HTTPS more than HTTP.

Across the board both kinds of devices generate more HTTPS traffic than HTTP, but that the prominent trend is simply that compromised devices generate more web traffic than clean devices. To illustrate this point in more detail, Figure 3.5 shows the distributions of average weekly device web activity for clean and compromised devices. For every device, we count the number of fully qualified domains the device visits via HTTP and HTTPS combined per week, and normalize by averaging across all weeks that the device was active. Each bar in the histogram counts the number of devices that visit a given number of FQDNs per week, with 100-domain bins. The distribution for compromised devices is clearly shifted towards visiting more sites per week (and other traffic granularities show similar behavior). We interpret this

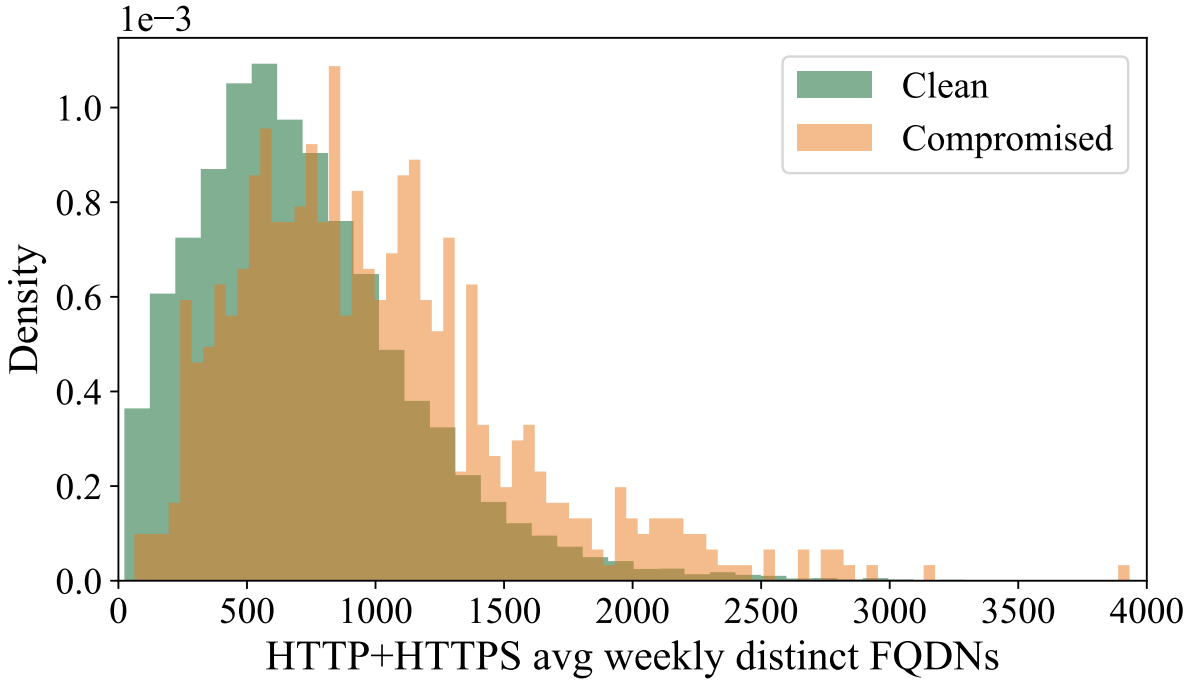

Figure 3.5. Distributions of average weekly device web activity for clean and compromised devices.

result as just reflecting that more activity correlates to higher exposure and risk (much like automobile accidents).

3.5.5 Use Antivirus

Using antivirus software is a nearly universal recommendation. In fact, residential students on our campus are nominally required to have antivirus software installed on their devices to use the network. We crafted signatures to detect network activity (e.g., updates to software or the signature database, callbacks when scanning, etc.) for over a dozen antivirus products, and Table 3.6 shows the distribution of popular products among our device population. If a device matched multiple signatures (e.g., Windows Defender and a third-party product), we counted the device in each category (hence the devices in the table sum to more than the unique device count). Avast, Windows Defender, and Avira are free, explaining their popularity among student devices.

Notably, while student devices technically need to have AV installed, regulations are not

AV Name	*# Devices*	*%*
Avast	6,704	33.2%
Windows	5,752	28.5%
McAfee	3,659	18.1%
Avira	1,837	9.1%
Norton	866	4.3%
Other	1,383	6.8%

Figure 3.6. Five most prevalent antivirus products observed, with all others aggregated as "Other".

always followed. We verified that students can still access the residential network with antivirus installed by repeatedly using a mechanism for visitors, or lying about their device type (e.g., claiming a MacBook is an iPad), and 7.5% of our devices fall into this category.

Using AV is strongly recommended to reduce risk. When focusing on Windows devices, interestingly a larger percentage (7%) of devices with antivirus are compromised compared to devices that do not have it (4%). By definition, though, most compromised devices in our population are those that were compromised by malware that antivirus did not catch.

3.5.6 Software Use

As discussed in Section 3.3.4, we extract a wide variety of features about the software used on devices observed on the network. We now explore how these software features correlate with a device being compromised. Since compromise depends on the operating system used (Windows devices are compromised more often than Mac OS devices), we also explore software features not only in the context of all devices but also individual operating systems.

For each correlated software feature, Table 3.8 shows the device population, fraction of compromised devices with the feature, and fraction of compromised devices without the feature. These results provide direct comparisons on compromise rates between devices with a particular software feature and without: e.g., devices using Tor are compromised 2–3.5$\times$ more

Table 3.8. Software features across device populations correlated with compromise. For each feature we show the number of devices with the feature, p-value from the Chi-Square test, fraction of compromised devices with and without the feature. Compromise rates: All devices 4.5%, Windows devices 7%, and Mac OS devices 1.9%.

Group	Feature	# Dev	P-value	w/ Feat.	w/o Feat.
All	Adobe AIR	826	< 0.001	10%	4%
All	P2P	2,237	< 0.001	13%	3%
All	Thunderbird	69	< 0.001	33%	4%
All	Uses Tor	321	< 0.001	12%	4%
All	Password Mgr.	434	< 0.001	8%	4%
All	Remote DNS	8,631	< 0.001	6%	2%
Win	Adobe AIR	490	< 0.001	13%	7%
Win	P2P	1,676	< 0.001	15%	5%
Win	Thunderbird	28	< 0.001	43%	7%
Win	Uses Tor	188	< 0.001	15%	7%
Win	Password Mgr.	262	0.001	12%	7%
Win	Remote DNS	5,249	< 0.001	8%	5%
Mac	Adobe AIR	336	< 0.001	6%	2%
Mac	P2P	541	< 0.001	7%	2%
Mac	Thunderbird	29	< 0.001	34%	2%
Mac	Uses Tor	123	< 0.001	7%	2%
Mac	Password Mgr	159	0.755	1%	2%
Mac	Remote DNS	3,212	< 0.001	3%	1%

often than devices that do not. To ensure that the comparisons are statistically significant, we use the Chi-Square test with Bonferroni correction since these are binary categorical features, and the very low p-values shown in Table 3.8 confirm significance.

Devices using some specific applications correlate very strongly with compromise, independent of operating system and network activity. Devices using Adobe AIR, P2P file sharing networks, Thunderbird, and Tor on average are much more likely to be compromised than devices that do not use such applications. Using these applications does indeed put devices at significantly more risk. The Thunderbird email client is particularly ironic since one reason why people use Thunderbird is because of its PGP integration [24]; yet, Thunderbird is rife with reported vulnerabilities (420 code execution vulnerabilities reported in CVE Details [17]).

Some of these software features do not directly lead to compromise, but instead indirectly reflect how attentive users are with respect to security. For instance, devices are not compromised due to using password managers or not, or whether they are kept updated, but the use of password managers does suggest that users are more security aware. We find the use of password managers to be correlated with compromise among the All and Windows device groupings. Similarly, users who explicitly configure their device to use a remote DNS server, instead of the DHCP default, reflect a certain degree of sophistication and confidence — for better or worse, considering that devices using remote DNS servers for resolution have a 1.6–3$\times$ higher rate of compromise.

3.6 Ranking Feature Importance

Our analyses so far have focused on individual security practices. As a final step, we explore the relative importance of all the features we extract using statistical modeling, as well as the relative importance of features exhibited during the hour before a device is compromised. Our goal is not to train a general security incident classifier. Rather, it is to generate a logistic model that produces interpretable results for ranking the relative importance of our features.

3.6.1 Experimental Setup

Logistic regression is a statistical technique for predicting a binary response variable using explanatory variables [38]. We set the response variable to be whether or not a device is compromised, and use all of the device features we extract from the network as the explanatory variables. We first split the data into training (50%) and test (50%), and normalize the explanatory variables to have zero mean and unit variance.

To find the important explanatory variables we use a specialized type of logistic regression called least absolute shrinkage and selection operataor (LASSO), or L1 logistic regression, since we have a high number of explanatory variables. L1 logistic regression can be regularized to correct for overfitting, thereby preventing a model from becoming too closely tied to the data that it is built from. Regularization restricts the number of explanatory variables the model

will use proportionally to how regularized the model is. The regularization parameter itself is configurable in the Scikit-learn machine learning framework we use [70].

To find the optimal regularization parameter we implement hyperparameter tuning: we build 200 models, each with a different regularization parameter, and identify the model that performs best. To identify the best model while avoiding selection bias, for each model, we perform 10-fold cross validation. We track the average area under curve (AUC) from the receiver operating characteristic (ROC) curves produced when predicting on the ten different validation data sets. We then select the regularization parameter from the model that provides the maximum average validation AUC. After identifying the optimal regularization parameter we search for multicollinearity by computing the variance inflation factor (VIF) across features used in the model, and do not find features with a VIF greater than ten [48].

To compare the importance of each feature we implement a greedy deletion algorithm [35]. Our algorithm works in the following way: We start with the N important features used to predict security incidents identified by the best model (previous paragraph). For $N-1$ feature combinations we train regularized models with hyperparameter tuning. From the resulting models, we identify the model that has the maximum AUC (when predicting on validation data), and exclude the unused feature in the next iteration of the algorithm. We exclude the unused feature since it contributes least to the overall AUC compared to the other feature combinations. We repeat this process until we have a model that uses a single feature ($N = 1$); the remaining feature contributes the most to the AUC by itself and in the presence of other features. Finally, we interpret the results in terms of the changes to the test AUC when features are added to the final model.

3.6.2 All Features

We run the greedy deletion algorithm (Section 3.6.1) multiple times with different device groupings: all devices, Windows devices, Mac OS devices, and devices with on-median more HTTP traffic. We consider devices that produce on-median more HTTP traffic based on our

Table 3.9. AUC gains from the top four features used to detect devices with security incidents. For each feature we also provide the ratio of median (continuous) or mean (categorical) values. Ratios > 1 (green) indicate that compromised devices exhibit more of the feature.

Group	Feature	Val AUC	Test AUC	Ratio
All	IAB Computer Games	+68.3%	+69.7%	2.2x
All	HTTP Reg Domains	+7.0%	+5.2%	1.6x
All	HTTP in TLD .cn	+2.3%	+3.7%	3.5x
All	Windows Antivirus	+1.9%	+1.1%	1.7x
Win	HTTP FQ Domains	+71.9%	+71.1%	1.6x
Win	IAB Computer Games	+4.2%	+2.9%	1.7x
Win	UA Str Safari	+2.2%	+2.5%	3x
Win	UA Str IE	+1.4%	+1.3%	1.1x
Mac	HTTP in TLD .cn	+76%	+76%	∞
Mac	UA Str IE	+5.3%	+4.3%	6.2x
Mac	HTTP Traffic at 2AM	+3.8%	-1.3%	0.9x
Mac	HTTP in TLD co.kr	+1.5%	+3.7%	1x
HTTP	IAB Shareware	+66.3%	+60%	∞
HTTP	UA Str IE	+7.2%	+7.9%	1.9x
HTTP	UA Str Android	+3.4%	+1.3%	2.2x
HTTP	Uses P2P	+1.0%	+2.7%	1.3x

observations in Section 3.5.4. Table 3.9 shows the top four features for each grouping, the feature's AUC contribution when predicting on validation and test data, and the ratio of the feature's median (continuous) or mean (categorical) value for compromised and clean devices. Since we select the feature combination with the highest validation AUC it is possible that adding in an extra feature will result in a small negative contribution to the test AUC (e.g., the "HTTP Traffic at 2AM" feature for Mac OS devices).

Our results indicate that behavioral features, regardless of device grouping, are most correlated with device compromise. In all cases, the first feature in each grouping relates to how much a device accesses Web content or the type of content being accessed. Having Windows antivirus products (a proxy for using Windows, which has a significantly higher compromise rate), or using P2P applications are the only two software features in the top four of any grouping. Having the IE User-Agent feature highly ranked highlights the challenge of cursory feature

Table 3.10. AUC gains for the top eight features used to detect devices with security incidents one hour before being compromised.

Feature	Val AUC	Test AUC
IAB Computer Games	+71.9%	+74.2%
IAB Web Search	+4.0%	+3.6%
IAB Illegal Content	+2.2%	+3.6%
IAB JavaScript	+1.0%	+0.1%
IAB Computer Networking	+0.7%	+0.1%
IAB Adult Content	+0.7%	+0.7%
IAB Shareware/Freeware	+0.7%	+0.4%
IAB Internet Technology	+0.5%	+1.5%

extraction. Applications can make use of embedded browsers, and examining traffic with an IE User-Agent string shows many of the detections are from the QQ chat application and Qihoo 360 security product. We also find that compromised devices, in the majority of cases (except for two features within the Mac OS grouping), exhibit more of each feature compared to clean devices.

3.6.3 One Hour Before Compromise

Lastly, we use our statistical model to examine the relative importance of security features focusing on the hour leading up to device compromise: Compared to devices that are not compromised, how are compromised devices behaving differently leading up to becoming compromised? For each compromised device, we extract their features from the hour before their first incident. To compare differences in behavior, we construct a synthetic control by taking a pseudorandom sample of clean devices. Specifically, for each compromised device we randomly select up to 300 clean devices that are (1) active in the same hour window, and (2) visit at least 50 distinct registered domains.[12]

Table 3.10 shows the most important features (relative to one another) for identifying compromised devices an hour before they are compromised. For our devices, the type of Web sites visited (Section 3.5.3) are the most distinguishing features. On-average, compromised

[12]On average compromised devices visit 50 distinct registered domains the hour before being compromised.

devices visit more Web sites in each of the eight categories in Table 3.10 than clean devices. The most popular domains our devices visit in these categories do correspond well to the category domains. For some of the very generic labels, “Computer Games” are gaming sites; “Computer Networking” include ISPs and IP geolocation services; “Internet Technology” include SSL certificate sites and registrars, etc.

3.7 Conclusion

The practice of cybersecurity implicitly relies on the assumptions that users act “securely” and that our security advice to them is well-founded. In this paper, we seek to ground both assumptions empirically – measuring both the prevalence of key security “best practices” as well as the extent to which these behaviors (and others) relate to eventual security outcomes. We believe that this kind of analysis is key to advancing security decision making from the “gut instinct” practice it is today, to one informed and improved by the collection on concrete evidence.

Acknowledgements

Chapter 3, in part, has been submitted for publication of the material as it may appear in *Proceedings of the ACM Internet Measurement Conference (IMC)*. Louis F. DeKoven, Audrey Randall, Ariana Mirian, Gautam Akiwate, Ansel Blume, Lawrence K. Saul, Aaron Schulman, Geoffrey M. Voelker, Stefan Savage, 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 4

Conclusion

Addressing compromised devices is a problem for virtually all large organizations. In most cases, large organizations must address device compromise using limited observational vantage points, differences in intervention capabilities, and evolving adversaries with varying incentives. In this dissertation, we have demonstrated multiple approaches to develop empirically-grounded systems that address device compromise within different organizations, thereby placing us on a stronger footing to devise better interventions. We presented solutions that take advantage of analytic data to determine: what can be measured under the limitations in each organization's vantage point, as well as the trade-offs across different types of intervention.

The first system detects and remediates malicious browser extensions impacting Facebook. From the perspective of an online social network browser extensions themselves are not directly accessible. Moreover, how or why a extension is installed may be unknown. We described our methodology whereby users exhibiting suspicious online behaviors are scanned (with permission) to identify the set of extensions in their browsers, and those extensions are in turn labeled based on the threat indicators they contain. Employing this methodology at Facebook over six weeks, we identified more than 1,700 new lexically distinct malicious extensions. Comparing our findings with both contemporaneous anti-malware detections (as reflected in VirusTotal) and takedowns from the Chrome Web Store, reveals a considerable detection gap in the existing abuse ecosystem. We hope that by highlighting this issue and sharing our data we can encourage

a broader and more collaborative focus on this under-addressed attack vector.

Next, we presented a system to disrupt for-profit underground services offering to artificially manipulate a user's social standing on Instagram. Unlike browser extensions the malicious software used by these services is not required to run directly on each device making the use of a malware scanner-like approach impractical. We identified techniques used by these services to evade straightforward detection, and characterized the dynamics of their customer bases using a honeypot account framework that we developed for Instagram. We found that underground services are able to attract a large clientele, and generate over $1M in monthly revenue. Lastly, we have shown through controlled experiments that blocking underground services, while effective in the short term, quickly drives adaptation and can make it difficult to amortize the cost of developing accurate abuse classification. For example, underground service quickly reacted to synchronous blocking applied to their abusive actions. Consequently, from the standpoint of protecting non-abusive users from artificial content, a more effective long-term strategy can be built on deferred interventions (e.g., removing synthetic actions after at a future point). Such approaches greatly increase the "debug time" for services seeking to reverse engineer how they are being detected and are less likely to drive the customer complaints that incentive services to pursue such adaptations.

In our final study, we developed a system that passively monitors a university's residential network to measure the prevalence of numerous security " best practices" and behaviors, and how they correlate to device compromise. Unlike the vantage point of online social networks which is constrained to social network actions, passive network traffic contains a broader set of device actions. However, this activity is encoded within network traffic, and fine-grained detail into the context of each action is commonly obscured by encryption. We described the implementation of a large-scale passive monitoring system that produces per-device models describing security practices and behaviors. Analyzing months of longitudinal data we have shown that a number of recommended security "best practices" are followed by devices, however, they are not negatively correlated with device compromise–e.g., compromised devices using

the Chrome or Firefox web browser tend to update faster than clean devices. Most positively correlated with device compromise is the types of web site devices visit–e.g., web content related to video games. Lastly, we developed a logistic model to compare the relative importance of the features we measure and device compromise. Our model shows that behavioral features such are more useful for distinguishing compromised devices. We believe that this kind of analysis is key to advancing security decision making from the "gut instinct" practice it is today, to one informed and improved by the collection on concrete evidence.

Bibliography

[1] Anupama Aggarwal and Ponnurangam Kumaraguru. What They Do in Shadows: Twitter Underground Follower Market. In *Proceedings of the 13th Conference on Privacy, Security and Trust (PST)*, Izmir, Turkey, July 2015.

[2] Apache Software Foundation. Apache Hive Website. https://hive.apache.org/, 2019.

[3] Apple. Update your iPhone, iPad, or iPod touch. https://support.apple.com/en-us/HT204204, 2018.

[4] S. Bandhakavi, S. T. King, M. Parthasarathy, and M. Winslett. Vetting Browser Extensions for Security Vulnerabilities with VEX. In *Proc. of USENIX Security*, 2010.

[5] Mihir Bellare and Phillip Rogaway. The FFX Mode of Operation for Format-Preserving Encryption. *Manuscript (standards proposal) submitted to NIST*, January 2010.

[6] Leyla Bilge, Yufei Han, and Matteo Dell'Amico. RiskTeller: Predicting the Risk of Cyber Incidents. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Dallas, Texas, USA, November 2017.

[7] Blognife. PopAds CPM Rates 2018. http://blognife.com/2017/06/22/popads-cpm-rates-2017/, 2017.

[8] Boostgram. Boostgram Web site. https://boostgram.com, 2017.

[9] Brian Krebs. Nasty Twitter Worm Outbreak. https://krebsonsecurity.com/2010/09/nasty-twitter-worm-outbreak/, 2010.

[10] Brian Krebs. Buying Battles in the War on Twitter Spam. https://krebsonsecurity.com/2013/08/buying-battles-in-the-war-on-twitter-spam/, 2013.

[11] Davide Canali, Leyla Bilge, and Davide Balzarotti. On the effectiveness of risk prediction based on users browsing behavior. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (CCS)*, Kyoto, Japan, June 2014.

[12] Yannick Carlinet, Ludovic Mé, Hervé Debar, and Yvon Gourhant. Analysis of Computer Infection Risk Factors Based on Customer Network Usage. In *2008 Second International Conference on Emerging Security Information, Systems and Technologies*, Cap Esterel, France, August 2008.

[13] N. Carlini, A. P. Felt, and D. Wagner. An Evaluation of the Google Chrome Extension Security Architecture. In *Proc. of USENIX Security*, 2012.

[14] Carrie Marshall and Cat Ellis. The best free password manager 2019. https://www.techradar.com/news/software/applications/the-best-password-manager-1325845, 2018.

[15] CERT. Ransomware. https://www.us-cert.gov/security-publications/Ransomware, 2018.

[16] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, pages 84–88, February 1981.

[17] CVE Details. Mozilla Thunderbird Vulnerability Statistics. https://www.cvedetails.com/product/3678/?q=Thunderbird, 2019.

[18] Danny Palmer. Facebook Messenger user? Watch out for fake messages rigged with malware. https://www.zdnet.com/article/facebook-messenger-user-watch-out-for-fake-messages-rigged-with-malware/, 2017.

[19] Emiliano De Cristofaro, Arik Friedman, Guillaume Jourjon, Mohamed Ali Kaafar, and M. Zubair Shafiq. Paying for Likes? Understanding Facebook Like Fraud Using Honeypots. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 129–136, Vancouver, BC, Canada, November 2014.

[20] M. Dhawan and V. Ganapathy. Analyzing Information Flow in JavaScript-based Browser Extensions. In *Proc. of ACSAC*, 2009.

[21] V. Djeric and A. Goel. Securing Script-Based Extensibility in Web Browsers. In *Proc. of USENIX Security*, 2010.

[22] DNSFilter. DNSFilter Website. https://www.dnsfilter.com/, 2019.

[23] Earning Guys. PopAds Review: A Pop-under Ad Network. http://www.earningguys.com/advertisement/popads-review/, 2017.

[24] The Enigmail Project. Enigmail — OpenPGP encryption for Thunderbird. https://www.enigmail.net/index.php/en/home, 2019.

[25] Shehroze Farooqi, Fareed Zaffar, Nektarios Leontiadis, and Zubair Shafiq. Measuring and Mitigating Oauth Access Token Abuse by Collusion Networks. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 355–368, London, UK, November 2017.

[26] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel1, and Parisa Tabriz. Measuring HTTPS Adoption on the Web. In *Proceedings of the 26th USENIX Security Symposium*, Vancouver, BC, Canada, August 2017.

[27] M. Finifter, J. Weinberger, and A. Barth. Preventing Capability Leaks in Secure JavaScript Subsets. In *Proc. of NDSS*, 2010.

[28] Firefox. How to stop Firefox from making automatic connections. https://support.mozilla.org/en-US/kb/how-stop-firefox-making-automatic-connections, 2019.

[29] Followersgratis. Followersgratis Web site. http://followersgratis.org, 2017.

[30] Alain Forget, Sarah Pearman, Jeremy Thomas, Alessandro Acquisti, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, Marian Harbach, and Rahul Telang. Do or Do Not, There Is No Try: User Engagement May Not Improve Security Outcomes. In *Procedings of the Twelfth Symposium on Usable Privacy and Security (SOUPS)*, Denver, CO, USA, June 2016.

[31] Fstoppers. Mass Planner Shut Down by Instagram: The End of the Bot Era. https://fstoppers.com/social-media/mass-planner-shut-down-instagram-end- bot-era-176654, 2017.

[32] Aaron Gember, Ashok Anand, and Aditya Akella. A Comparative Study of Handheld and Non-handheld Traffic in Campus Wi-Fi Networks. In *Proceedings of the 12th International Conference on Passive and Active Measurement*, Atlanta, GA, USA, March 2011.

[33] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, Boston, MA, USA, August 2007.

[34] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy. Verified Security for Browser Extensions. In *Proc. of IEEE S&P*, 2011.

[35] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer New York Inc., 2001.

[36] Cormac Herley. So Long, and No Thanks for the Externalities: The Rational Rejection of Security Advice by Users. In *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*, Oxford, United Kingdom, September 2009.

[37] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. FRAUDAR: Bounding Graph Fraud in the Face of Camouflage. In *Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 895–904, San Francisco, CA, USA, August 2016.

[38] David W Hosmer Jr and Stanley Lemeshow. *Applied Logistic Regression*. John Wiley & Sons, 2nd edition, 2000.

[39] Hublaagram. Hublaagram Web site. http://hublaagram.me, 2017.

[40] IAB. IAB Tech Lab Content Taxonomy. https://www.iab.com/guidelines/iab-tech-lab-content-taxonomy/, 2019.

[41] Instagram. Strengthening Our Commitment to Safety and Kindness for 800 Million. http://blog.instagram.com/post/165759350412/170926-news, 2017.

[42] Instagram. Terms of Use. https://help.instagram.com/581066165581870, 2018.

[43] Instalex. Instalex Web site. https://instalex.ru, 2017.

[44] Instalex Franchise. Instalex Franchise Web site. https://instalex.pro/franchise, 2017.

[45] Instazood. What is a Good Engagement Rate on Instagram. https://instazood.com/what-is-a-good-engagement-rate-on-instagram/, 2017.

[46] Instzood. Instzood Web site. https://instazood.com, 2017.

[47] N. Jagpal, E. Dingle, J. Gravel, P. Mavrommatis, N. Provos, M. A. Rajab, and K. Thomas. Trends and Lessons from Three Years Fighting Malicious Extensions. In *Proc. of USENIX Security*, 2015.

[48] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[49] Mobin Javed, Cormac Herley, Marcus Peinado, and Vern Paxson. Measurement and Analysis of Traffic Exchange Services. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 1–12, Tokyo, Japan, October 2015.

[50] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. Hulk: Eliciting Malicious Behavior in Browser Extensions. In *Proc. of USENIX Security*, 2014.

[51] Moazzam Khan, Zehui Bi, and John A. Copeland. Software updates as a security metric: Passive identification of update trends and effect on machine infection. In *Proceedings of IEEE Military Communications Conference (MILCOM)*, Orlando, Florida, USA, October 2012.

[52] Kirk Bauer. Logwatch. http://www.logwatch.org, 2011.

[53] Fanny Lalonde Lévesque, Jude Nsiempba, José M. Fernandez, Sonia Chiasson, and Anil Somayaji. A Clinical Study of Risk Factors Related to Malware Infections. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Berlin, Germany, November 2013.

[54] Kyumin Lee, James Caverlee, and Steve Webb. Uncovering Social Spammers: Social Honeypots + Machine Learning. In *Proceedings of the 33rd ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 435–442, Geneva, Switzerland, July 2010.

[55] Lindsey O'Donnell. New Facebook-Spread Malware Triggers Credential Theft, Cryptomining. https://www.zdnet.com/article/cryptocurrency-mining-malware-is-number-one-malware-menace-again/, 2018.

[56] L. Liu, X. Zhang, G. Yan, and S. Chen. Chrome Extensions: Threat Analysis and Countermeasures. In *Proc. of NDSS*, 2012.

[57] Yang Liu, Armin Sarabi, Jing Zhang, Parinaz Naghizadeh, Manish Karir, Michael Bailey, and Mingyan Liu. Cloudy with a Chance of Breach: Forecasting Cyber Security Incidents. In *Proceedings of the 24th USENIX Conference on Security Symposium*, Washington, DC, USA, August 2015.

[58] M. T. Louw, J. S. Lim, and V.N Venkatakrishnan. Enhancing web browser security against malware extensions. *Journal in Computer Virology*, 2008.

[59] McAfee. McAfee Labs Threats Report, 2018.

[60] Medium. Instag-RAMPAGE: the WAR on Automation. https://medium.com/@mountainbeard/instag-rampage-and-the- war-on-automation-3a7362b08112, 2017.

[61] Medium, Shane Barker. How to Become an Instagram Influencer and Start Earning Money Now. https://medium.com/swlh/how-to-become-an-instagram-influencer-and-start-earning-money-now-a8ef3169e96d, 2018.

[62] Microsoft. Microsoft Update Catalog. https://www.catalog.update.microsoft.com/Home.aspx, 2019.

[63] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 29–42, San Diego, CA, USA, October 2007.

[64] Mozilla Foundation. Public Suffix List Website. https://publicsuffix.org/, 2019.

[65] Neil J. Rubenking. The Best Antivirus Protection for 2019. https://www.pcmag.com/article2/0,2817,2372364,00.asp, 2019.

[66] New York Times. How Bots Are Inflating Instagram Egos. https://www.nytimes.com/2017/06/06/business/media/instagram-bots.html, 2017.

[67] New York Times. The Follower Factory. https://www.nytimes.com/interactive/2018/01/27/technology/social-media-bots.html, 2018.

[68] ntop. PF_RING ZC (Zero Copy) Website. https://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/, 2018.

[69] Vern Paxson. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[70] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[71] Pop Ads. PopAds Web site. https://www.popads.net, 2017.

[72] ProofPoint. ET Pro Ruleset. https://www.proofpoint.com/us/threat-insight/et-pro-ruleset, 2019.

[73] Rachel Wolfson. Cryptojacking On The Rise: WebCobra Malware Uses Victims' Computers To Mine Cryptocurrency. https://bit.ly/2vVfdZC, 2018.

[74] Redislabs. Redis Website. https://redis.io/, 2019.

[75] Elissa M. Redmiles, Sean Kross, and Michelle L. Mazurek. How I Learned to Be Secure: A Census-Representative Survey of Security Advice Sources and Behavior. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, October 2016.

[76] Elissa M. Redmiles, Sean Kross, and Michelle L. Mazurek. Where is the Digital Divide?: A Survey of Security, Privacy, and Socioeconomics. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, Denver, Colorado, USA, May 2017.

[77] Elissa M. Redmiles, Sean Kross, and Michelle L. Mazurek. How Well Do My Results Generalize? Comparing Security and Privacy Survey Results from MTurk, Web, and Telephone Samples. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy*, San Fransisco, CA, USA, May 2019.

[78] Robert Reeder, Iulia Ion, and Sunny Consolvo. 152 Simple Steps to Stay Safe Online: Security Advice for Non-tech-savvy Users. *IEEE Security and Privacy*, 15(5):55–64, June 2017.

[79] Armin Sarabi, Ziyun Zhu, Chaowei Xiao, Mingyan Liu, and Tudor Dumitras. Patch Me If You Can: A Study on the Effects of Individual User Behavior on the End-Host Vulnerability State. In *Proceedings of the 18th Passive and Active Measurement PAM*, Sydney, Australia, March 2017.

[80] Yukiko Sawaya, Mahmood Sharif, Nicolas Christin, Ayumu Kubota, Akihiro Nakarai, and Akira Yamada. Self-Confidence Trumps Knowledge: A Cross-Cultural Study of Security Behavior. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, Denver, Colorado, USA, May 2017.

[81] H. Shahriar, K. Weldemariam, T. Lutellier, and M. Zulkernine. A Model-Based Detection of Vulnerable and Malicious Browser Extensions. In *Proc. of SERE*, 2013.

[82] Mahmood Sharif, Jumpei Urakawa, Nicolas Christin, Ayumu Kubota, and Akira Yamada. Predicting Impending Exposure to Malicious Content from User Behavior. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, Toronto, Canada, October 2018.

[83] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting Spammers on Social Networks. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, Austin, TX, USA, 2010.

[84] Gianluca Stringhini, Gang Wang, Manuel Egele, Cristopher Kruegel, Giovanni Vigna, Haitao Zheng, and Ben Y. Zhao. Follow the Green: Growth and Dynamics in Twitter Follower Markets. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, pages 163–176, Barcelona, Spain, October 2013.

[85] Suricata. Suricata IDS Website. https://suricata-ids.org/, 2019.

[86] The Verge. Popular Instagram bot site Instagress has been shut down. https://www.theverge.com/2017/4/20/15374080/instagram-bot-site-instagress-dead, 2017.

[87] Update Google Chrome. Update Google Chrome. https://support.google.com/chrome/answer/95414?co=GENIE.Platform%3DDesktop&hl=en, 2019.

[88] Bimal Viswanath, M. Ahmad Bashir, Mark Crovella, Saikat Guha, Krishna P. Gummadi, Balachander Krishnamurthy, and Alan Mislove. Towards Detecting Anomalous User Behavior in Online Social Networks. In *Proceedings of the 23rd USENIX Security Symposium*, pages 223–238, San Diego, CA, USA, August 2014.

[89] Francesco Vitale, Joanna McGrenere, Aurélien Tabard, Michel Beaudouin-Lafon, and Wendy E. Mackay. High Costs and Small Benefits: A Field Study of How Users Experience Operating System Upgrades. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, Denver, Colorado, USA, May 2017.

[90] J. Wang, X. Li, X. Liu, X. Dong, J. Wang, Z. Liang, and Z. Feng. An Empirical Study of Dangerous Behaviors in Firefox Extensions. In *Proc. of ICISC*, 2012.

[91] Rick Wash. Folk Models of Home Computer Security. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, Redmond, Washington, USA, July 2010.

[92] Rick Wash and Emilee Rader. Too Much Knowledge? Security Beliefs and Protective Behaviors Among United States Internet Users. In *Proceedings of the Eleventh USENIX Conference on Usable Privacy and Security*, Ottawa, Canada, July 2015.

[93] Steve Webb, James Caverlee, and Calton Pu. Social Honeypots: Making Friends With A Spammer Near You. In *Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, August 2008.

[94] WebShrinker. IAB Categories. https://docs.webshrinker.com/v3/iab-website-categories.html#iab-categories, 2018.

[95] WebShrinker. WebShrinker Website. https://www.webshrinker.com/, 2019.

[96] M. West, A. Barth, and D. Veditz. Content Security Policy Level 3. *W3C*, 2016.

[97] The Wireshark Team. Wireshark Website. https://www.wireshark.org/, 2019.

[98] Chaowei Xiao, Armin Sarabi, Yang Liu, Bo Li, Mingyan Liu, and Tudor Dumitras. From Patching Delays to Infection Symptoms: Using Risk Profiles for an Early Discovery of Vulnerabilities Exploited in the Wild. In *Procedings of the 27th USENIX Security Symposium (USENIX Security)*, Baltimore, MD, USA, August 2018.

[99] Zeek. Zeek Protocol Analyzers Website. https://docs.zeek.org/en/stable/script-reference/proto-analyzers.html, 2019.