Malicious Browser Extensions at Scale: Bridging the Observability Gap between Web Site and Browser

Louis F. DeKoven[1], Stefan Savage[1], Geoffrey M. Voelker[1], and Nektarios Leontiadis[2]

[1]*University of California, San Diego* [2]*Facebook*

Abstract

Browser extensions enhance the user experience in a variety of ways. However, to support these expanded services, extensions are provided with elevated privileges that have made them an attractive vector for attackers seeking to exploit Internet services. Such attacks are particularly vexing for the sites being abused because there is no standard mechanism for identifying which extensions are running on a user's browser, nor is there an established mechanism for limiting the distribution of malicious extensions even when identified.

In this paper we describe an approach used at Facebook for dealing with this problem. We present a methodology whereby users exhibiting suspicious online behaviors are scanned (with permission) to identify the set of extensions in their browser, and those extensions are in turn labelled based on the threat indicators they contain. We have employed this methodology at Facebook for six weeks, identifying more than 1 700 lexically distinct malicious extensions. We use this labelling to drive user device clean-up efforts as well to report to anti-malware and browser vendors.

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) [13] 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 [3]. Unsurprisingly, this problem has evolved from one of abstract potential into a concrete threat, and today the problem of Malicious Browser Extensions (MBE) is widely understood to be real and growing [3, 7, 8, 10, 11]. 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. [7]). 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 2 provides an overview of browser extensions and related work; Section 3 outlines Facebook's approach for detecting compromised user accounts, and collecting and analyzing browser extensions; Section 4 describes the automated extension labeling system, and Section 5 evaluates it, characterizing the volume of extensions Facebook deals with and system behavior over time; Section 6 motivates the need for the new labeling system; and Section 7 concludes.

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" [9], 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 [3]. Researchers have shown similar problems exist in modern browsers [2, 5, 9, 12] and large-scale empirical measurements [8] and operational experience [7] 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 [4, 6, 9, 10] or to provide a better mechanism for vetting code in extension marketplaces [1].

Our work builds on ideas from these prior efforts. Our approach is data driven, like the work of Jagpal et al. [7] and Kapravelos et al. [8], 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.

3 Collecting browser malware

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.

[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.

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

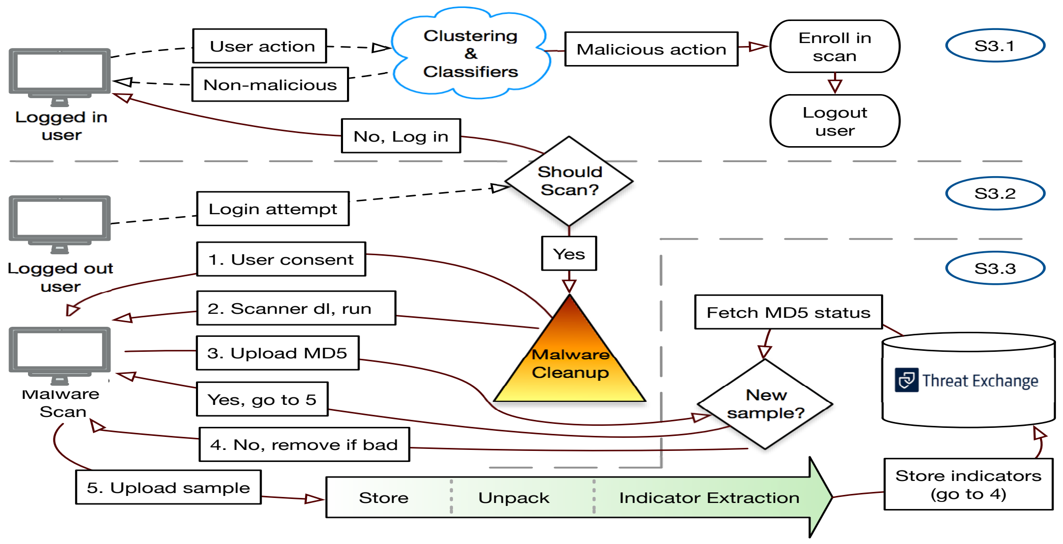


Figure 1: An overview of our system highlighting the detection, malware scanner, and static analysis steps. The dashed arrows describe normal user interaction, and solid arrows are transitions within the described system.

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.

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) 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 need an analyst to examine and take action.

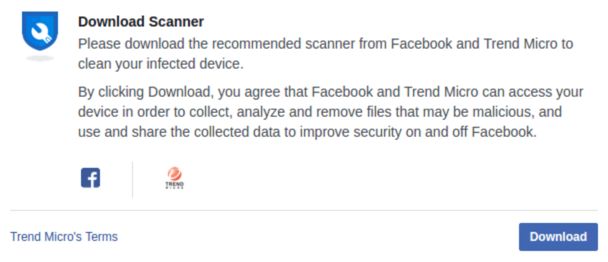
An example of the latter case would be auto-generated objectionable content being shared on Facebook (e.g. adult content) with similar characteristics, e.g. directing viewers to the same external domain that results in a drive-by malware infection. In such a case, the analyst would typically add the 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.

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 under "Malware Cleanup". Following user consent (see Figure 2), the user downloads a one-time malware scanner that runs on the potentially

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


Download Scanner

Please download the recommended scanner from Facebook and Trend Micro to clean your infected device.

By clicking Download, you agree that Facebook and Trend Micro can access your device in order to collect, analyze and remove files that may be malicious, and use and share the collected data to improve security on and off Facebook.

Trend Micro's Terms

Download

Figure 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.

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 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.

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 [7]. 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 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.

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

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.

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 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.

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 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 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.

4.1.1 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.

4.1.2 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.

4.1.3 Known false positives

Throughout the 6-week measurement period our system collected over 34*k* 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 3.2. We consider this number of false positives as small in number and of an acceptable magnitude.

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.

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`.

Figure 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.

[5] e.g. MD5 `a369ecc2e8ca5924ddf1639993ffa3aa`

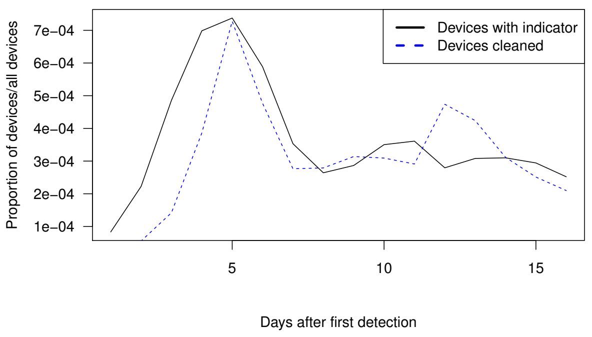


Figure 3: Daily proportion of user devices detected with a DOM-based indicator of the botnet, and the proportion of user devices that have the botnet remediated.

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

5.1 Extensions collected

Table 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 741*k* 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, 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

	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

Table 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%.

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.

5.2 Malicious extensions detected

We now consider the methodology for automated MBE labeling we presented in Section 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 (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 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 seconds after collection. However,

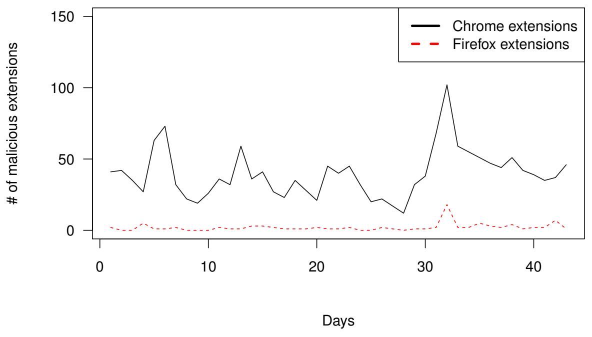


Figure 4: The number of unique extensions labeled as malicious each day of the six-week measurement period.

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 ≈ 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.

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.

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 [8] 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.

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.

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].

References

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

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

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

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

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

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

[7] 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.

[8] 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.

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

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

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

[12] 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.

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

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