Trends and Lessons from Three Years Fighting Malicious Extensions

Nav Jagpal Eric Dingle Jean-Philippe Gravel Panayiotis Mavrommatis
Niels Provos Moheeb Abu Rajab Kurt Thomas

Google

{*nav, ericdingle, jpgravel, panayiotis, niels, moheeb, kurtthomas*}*@google.com*

Abstract

In this work we expose wide-spread efforts by criminals to abuse the Chrome Web Store as a platform for distributing malicious extensions. A central component of our study is the design and implementation of WebEval, the first system that broadly identifies malicious extensions with a concrete, measurable detection rate of 96.5%. Over the last three years we detected 9,523 malicious extensions: nearly 10% of every extension submitted to the store. Despite a short window of operation—we removed 50% of malware within 25 minutes of creation— a handful of under 100 extensions escaped immediate detection and infected over 50 million Chrome users. Our results highlight that the extension abuse ecosystem is drastically different from malicious binaries: miscreants profit from *web traffic* and *user tracking* rather than email spam or banking theft.

1 Introduction

Browsers have evolved over recent years to mediate a wealth of user interactions with sensitive data. Part of this rich engagement includes extensions: add-ons that allow clients to customize their browsing experience by altering the core functionality of Chrome, Firefox, and Internet Explorer. Canonical examples include search toolbars, password managers, and ad blockers that once installed intercept webpage content through well-defined APIs to modify every page a user visits.

Criminals have responded in kind by developing malicious extensions that interpose on a victim's browsing sessions to steal information, forge authenticated requests, or otherwise tamper with page content for financial gain. Poignant malware strains include Facebook account hijackers, ad injectors, and password stealers that exist purely as *man-in-the-browser* attacks [21, 25, 37, 41]. While many of these threats have binary-based equivalents—for instance the Torpig banking trojan that injected rogue phishing forms into banking webpages or the ZeroAccess bot that tampered with page advertisements [27, 34]—extensions bridge the semantic gap between binaries and browsers, trivializing broad access to complex web interactions.

In this paper we expose wide-spread efforts by criminals to abuse the Chrome Web Store as a platform for distributing malicious extensions. Our evaluation covers roughly 100,000 unique extensions submitted to the Chrome Web Store over a three year span from January 2012–2015. Of these, we deem nearly *one in ten* to be malicious. This threat is part of a larger movement among malware authors to pollute official marketplaces provided by Chrome, Firefox, iOS, and Android with malware [7, 10, 42].

A central component of our study is the design and implementation of WebEval, the first system that broadly identifies malicious extensions with a concrete, measurable detection rate of 96.5%. We arrive at a verdict by classifying an extension's behaviors, code base, and developer reputation. In the process, we incorporate existing techniques that detect specific malware strains and suspicious extension behaviors and evaluate each of their effectiveness in comparison to our own [21, 37, 41]. WebEval also faces a unique challenge: live deployment protecting the Chrome Web Store where attackers have a strong incentive to adapt to our infrastructure. We explore the impact that evasive threats have on our overall accuracy throughout our deployment and the necessity of human experts to correct for model drift.

In total, we removed 9,523 malicious extensions from the Chrome Web Store. The most prominent threats included social network session hijackers that generated synthetic likes, friend requests, and fans; ad injectors that rewrote DOM content to laden pages with additional advertisements; and information stealers that injected rogue tracking pixels and covertly siphoned search keywords. Despite a short window of operation—we re-

ported 50% of malware within 25 minutes of creation—a handful of under 100 malicious extensions distributed via binary payloads were able to infect nearly 50 million users before removal. We distill these observations into a number of key challenges facing app marketplaces that extend beyond just the Chrome Web Store.

In summary, we frame our contributions as follows:

- We present a comprehensive view of how malicious extensions in the Chrome Web Store have evolved and monetized victims over the last three years.
- We detail the design and implementation of our security framework that combines dynamic analysis, static analysis, and reputation tracking to detect 96.5% of all known malicious extensions.
- We highlight the importance of human experts in operating any large-scale, live deployment of a security scanner to address evasive malware strains.
- We explore the virulent impact of malicious extensions that garner over 50 million installs; the single largest threat infecting 10.7 million Chrome users.

2 Background

We provide a brief background on how users obtain extensions and the control extensions have over the Chrome browser that simplify malware development.

2.1 Chrome Web Store

The Chrome Web Store is the central repository of all Chrome extensions. While initially the store was an optional ecosystem, rampant abuse outside of the store lead to Chrome locking down all Windows machines in May 2014 [13]. With this policy decision, Chrome now automatically blocks all extensions not present in the store from installation.

The Chrome Web Store relies on built in protections against malware that subject every extension to an abuse review. This approach is not unique to Chrome: Firefox, iOS, and Android all rely on application reviews to protect their user base from malware [1, 14, 17]. Malicious extensions detected during preliminary review are never exposed to the public. In the event the Chrome Web Store retroactively detects a malicious extension, the store can take down the offending code and signal for all Chrome clients to expunge the extension [16]. This defense layer provides a homogeneous enforcement policy for all Chrome users compared to the heterogeneous security environments of their desktop systems that may have no recourse against malicious extensions.[1]

[1]Chrome extensions are not supported on Android or other mobile platforms. As such, we limit our discussion of malicious extensions to desktop environments.

2.2 Chrome Extension Architecture

Developers author extensions much like websites using a combination of JavaScript, CSS, and HTML. Unlike websites, extensions are exempt from same origin protections and are afforded a range of Chrome and document-level controls that allow customizing how users interact with the Internet.

Permissions: Extensions may interact with privileged Chrome resources such as tabs, cookies, and network traffic through a Chrome-specific API. Chrome mediates these sensitive capabilities through a coarsely defined permission model where a permission consists of a resource (e.g., cookies) and a scope (e.g., *https://mail.google.com*) [4]. When a developer authors an extension, she lists all desired permissions in a static manifest. As discussed by Carlini *et al.*, this design favors "benign but buggy" extensions where the author adheres to a principle of least privilege [9]. The model provides no protection against malicious extensions beyond explicitly signaling broad capabilities (e.g., intercepting all network traffic).

Background Page & Content Scripts: Chrome loads an extension's core logic into a long running process called a background page. This privileged process obtains access to all of the Chrome API resources specified in the extension's permission manifest. To prevent permission re-delegation attacks, Chrome isolates background pages from all other extensions and web sites. Chrome eases this isolation by allowing extensions to register content scripts that run directly in the context of a web page as though part of the same origin (and thus with access to all of the origin's DOM content, DOM methods, and session cookies). Background pages communicate with content scripts through a thin message passing layer provided by Chrome. As with the Chrome API, extensions must specify content scripts and the targeted domains in an extension's manifest.

3 System Overview

We develop our system called WebEval to protect the Chrome Web Store from malicious extensions. The challenge is messy and fraught with evasive malware strains that adapt to our detection techniques. We rely on a blend of automated systems and human experts who work in conjunction to identify threats and correct for failures surfaced by user reports of abuse. Before diving into detailed system operations, we highlight the design principles that guided our development and offer a birds eye view of the entire architecture's operation.

3.1 Design Goals

At its heart, WebEval is designed to return a verdict for whether an extension is malicious. If the extension is pending publication, the Chrome Web Store should *block* the extension from release. Previously published extensions must be *taken down* and uninstalled from all affected Chrome instances. Arriving at a malware verdict is constrained by multiple requirements:

1. *Minimize malware installs*. Our foremost goal with WebEval is to minimize the number of users exposed to malicious extensions. However, near-zero false positives are imperative as Chrome expunges an extension's entire user base if we return a malware verdict incorrectly. We design our system such that human experts vet every verdict prior to actioning.

2. *Simplify human verification*. Whenever possible, our system should be fully automated to minimize the time required from human experts to confirm an extension is malicious.

3. *Time-constrained*. Our system embargoes extensions from public release until we reach a verdict. Its critical that we return a decision within one hour. Relatedly, our system must scale to the throughput of newly submitted items to the Chrome Web Store and weekly re-evaluated extensions that we estimate at roughly 19,000 reviews/day.

4. *Comprehensible, historical reports.* Any automated reports produced by our system must be comprehensible to human analysts, including machine learning verdicts. Similarly, all reports should contain some annotation to allow a historical perspective on the evolution of malicious extensions.

5. *Tolerant to feature drift*. Finally, our system must keep pace with the evasive nature of malware and adaptations in monetization strategies. This includes allowing experts to easily deploy new rules to capture emerging threats that are then automatically incorporated into long-running detection modules.

3.2 System Flow

WebEval is a living system that has evolved over the last three years in response to threats facing the Chrome Web Store. We describe our current pipeline for classifying an extension as malicious in Figure 1. The system consists of four stages: (❶) a scheduler that submits extensions for evaluation; (❷) our extension execution framework that captures behavioral signals; (❸) an annotation phase that incorporates content similarity, domain reputation, and anti-virus signatures; and finally (❹) scoring where manually curated rules, an automated classifier, and human experts reach a verdict for whether an extension is malicious.

Scheduler: We feed every extension uploaded to the Chrome Web Store, either new or updated, into our system and analyze it within one hour of submission. In total, we analyzed 99,818 Chrome extensions submitted over the course of January 2012–January 2015. This set includes extensions that were blocked prior to public release.[2] Furthermore, we have access to each revision of the extension's code base: over 472,978 unique variants (measured by SHA1 sums). Each revision triggers a re-scan in addition to a weekly re-scan aimed at extensions that fetch dynamic, remote resources that can become malicious.

Evaluation: We subject every extension to an evaluation phase that extracts behavioral signals for classification. This includes a reputation scan of the publisher, static analysis of the extension's code base, and dynamic analysis that emulates common tasks performed in Chrome: querying search engines, visiting social media, and browsing popular news sites. We store all raw features for posterity, totaling over 45 TB. Our philosophy is to retain everything (even packet contents) in order to enable offline analysis in the event an extension becomes defunct due to dead or broken remotely fetched resources. This storage simultaneously enables tracking trends in malware behavior over time and retroactively applying new malware signatures. We present the full details of our evaluation framework in Section 4.

Annotation: We practice a defense in depth strategy that incorporates domain blacklists, anti-virus engines, and content similarity that contextualizes an extension's behaviors against the larger ecosystem of malicious developers and extensions. We include these signals as annotations to an extension's evaluation in the event our own behavioral suites fail to surface any malicious logic. We present the annotation process in greater detail at the end of Section 4.

Scoring: The final step of WebEval returns a verdict for whether to expunge a malicious extension. We use a combination of manually curated rules and a logistic regression classifier re-trained daily over all previously detected malicious extensions to generate a score. A human expert then confirms our automated verdict before passing our decision on to the Chrome Web Store to take action. We present our technique for training, regulariza-

[2] We note that any extensions blocked prior to release are absent from the previous work by Kapravelos *et al.* [21] that studied malicious extensions found in the Chrome Web Store.

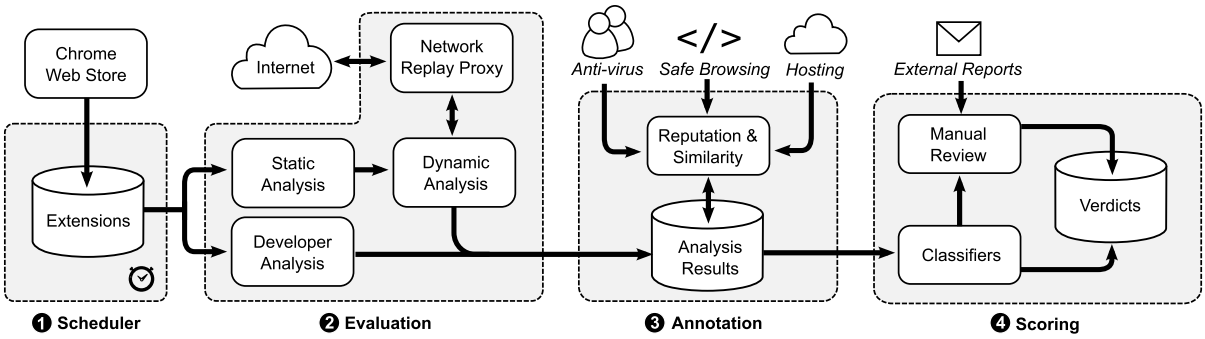

Figure 1: Our pipeline for detecting malicious extensions. WebEval's architecture consists of a scheduler, extension execution framework, an annotator that incorporates third-party security intelligence, and finally scoring where we return a malware verdict.

tion, and manual decision rules in Section 5. We discuss our approach for obtaining labeled data and evaluating our system's accuracy later in Section 6.

4 Evaluating Extensions

WebEval's core automation systems generate a report that surfaces signals about an extension's code base, the impact it has a user's browsing experience, and who developed the extension. With a few exceptions, none of these signals in isolation indicate outright malice: we leave it up to our classifier and human experts to determine which combinations of features clearly distinguish malware.

4.1 Static Analysis

Apart from remotely fetched resources, all of an extension's HTML, CSS, JavaScript, and manifest are self-contained and available to the Chrome Web Store upon submission. We scan through these components to identify potential threats.

Permissions & Content Scripts: We enumerate all an extension's permissions, content scripts, and contexts. Permissions in particular offer some indication of an extension's capabilities such as intercepting and modifying traffic (*proxy, webRequest*), triggering on a page load (*tabs*), introspecting on all cookies (*cookies*), and uninstalling or disabling other extensions (*management*). As part of this process we also identify broad contexts (e.g., <all_urls>, *https://**) that allow an extension to interact with every page.

Code Obfuscation: We scan for the presence of three types of code obfuscation: minification, encoding, and packing. We build on the detection strategy of Kaplan *et al.* that identifies common character substrings found in obfuscated vs. unobfuscated code [20]. Instead of detecting individual characters, we develop a set of regular expressions that identifies boilerplate initialization tied to the most prominent packers (e.g., *jsmini.com*, *jscompress.com*, and */packer/*). We employ a similar approach for detecting long encoded character strings. Finally, we detect minification by measuring the distance between a prettified version of an extension's JavaScript against the original supplied by the developer.

Files and Directory Structure: We extract the file names and directory structure of an extension as well as text shingles of the contents of every file. We rely on these features for detecting near-duplicate extensions (discussed in Section 4.4) as well as identifying commonly imported libraries and malicious files.

4.2 Dynamic Analysis

We collect the majority of our malware signals by black-box testing each extension with a barrage of behavioral suites that simulate common browsing experiences as well as custom tailored detection modules that trigger malicious logic. While more exhaustive approaches such as symbolic JavaScript execution exist [32], in practice we obtain sufficient enough behavioral coverage to reach accurate malware verdicts as discussed in Section 6.

Sandbox Environment: Our testing environment consists of a Windows virtual machine outfitted with two logging components: (1) a *system monitor* that captures low-level environment changes such as Windows settings, Chrome settings, and file creation; and (2) an *in-browser activity logger* that interposes on and logs all DOM events and Chrome API calls. This activity logger is natively built into Chrome explicitly for monitoring the behavior of extensions [28]. We note that Chrome isolates extensions from this logging infrastructure. Extensions cannot tamper with our results unless they compromise Chrome itself.

We supplement our monitoring infrastructure by routing all network traffic through a *network logging proxy*. This proxy also serves as a replay cache. For each test

suite we develop, we first record the network events produced by Chrome absent any extension installed. During dynamic evaluation we replay any network events from the cache that match. If a cache miss occurs, we route requests out of our sandbox to the Internet. This proxy allows us to minimize page dynamism, guarantee that test suites are consistent across executions absent new dynamic behavior, and reduce overall network round trip time. Similarly, we can easily flag network requests produced by our test actions (e.g., a click or fetching ad content) versus those produced by an extension.

The output of our dynamic analysis is a list of all network requests, DOM operations, and Chrome API calls made by an extension. Each of these include the page the event occurred on as well as the remote target of XHR requests and injected scripts. We supply all of these as raw features to our classifier as well as to human experts who can generate manual rules that capture sequences of events tied to known malware strains.

Behavioral Suites: The event driven nature of extensions requires that we replay realistic browsing scenarios to trigger malware. Our system allows human experts to record complex interactions (e.g., clicks, text entry, etc) with webpages that we then replay against every extension to detect malicious behaviors. These simulations, called behavioral suites, cover querying *google.com* with multiple searches; logging into *facebook.com* via a test account and viewing the account's news feed; shopping on *amazon.com* and *walmart.com*; and lastly browsing popular media sites including *nytimes.com* and *youtube.com*. As new threats arise, analysts can easily deploy new behavioral suites to trigger a malicious extension's logic.

Generic Suites: Our replay suites are by no means exhaustive; we rely on a generic set of test suites to simulate a wider variety of browser events. These tests are duplicates of the techniques previously discussed by Kapravelos *et al.* for Hulk [21] and include simulating network requests to popular news, video, shopping, and banking sites to trigger an extension's *webRequest* handler as well as using HoneyPages that create dummy elements on the fly to satisfy JavaScript requests from extensions.

Malicious Logic Suites: We supplement our browsing actions by explicitly testing an extension's logic against known threats: uninstalling other extensions (e.g., antivirus, Facebook malware remover); preventing uninstallation by terminating or redirecting tabs opening *chrome://extensions*; and stripping or modifying Content Security Policy headers. We explicitly flag each of these activities in addition to the log signals produced throughout the extension's evaluation.

4.3 Developer Analysis

The closed-garden nature of the Chrome Web Store enables tracking fine-grained reputation about developers and the extensions they author. We monitor where developers log in from, the email domain they use to register, the age of the developer account, and the total number of extensions authored thus far. These signals help us detect fake developer accounts that miscreants register via commonly abused email providers and proxies, staples of abusive account creation [39]. We note that newly registered developers must pay a nominal one-time fee of $5 that increases the overhead of churning out fake accounts [15].

In the event a malicious extension escapes initial detection, we also incorporate signals generated from users interacting with the Chrome Web Store. These includes the number of installs an extension receives, the number of users who have rated the extension, and the average rating. Our intuition is that highly used extensions that never receive any feedback are suspicious as are extensions that receive many low ratings.

4.4 Annotation

In the event the signals we collect during evaluation are insufficient, we rely on a defense in depth strategy that incorporates intelligence from the broader security community. In particular, we scan all of the files included in an extension with multiple anti-virus engines similar to VirusTotal.[3] If any single anti-virus vendor reports a file as malicious we flag the file in our report. We extend a similar strategy to all of the outgoing network requests produced by an extension where we scan the domains contacted against Google Safe Browsing and a collection of domain blacklists.

We also evaluate an extension in the context of all previously scanned extensions. We take the text shingles of an extension's code base computed during static analysis and identify near-duplicate extensions that share 80% of the same code. This approach allows us to detect extension developers that routinely re-upload previously detected malicious extensions. We extend this clustering logic to group extensions based on common embedded strings such as Google Analytics UIDs, Facebook App IDs, and Amazon Affiliate IDs. Finally, for extensions that evade initial detection and are released to the Chrome Web Store, we cluster the extensions based on the referrer of all incoming install requests to identify common websites involved in social engineering. We surface these clusters to human experts along with the ratio of known malware in each cluster.

[3] Due to licensing agreements, we are unable to disclose which anti-virus software we scan with.

5 Scoring Extensions

We reach a concrete verdict of an extension's malice by flagging any extension caught by our automated classifier or manually constructed heuristics. A human expert then verifies our decision and removes the offending extension from the Chrome Web Store if appropriate.

5.1 Automated Detection

Our automated detection uses a proprietary implementation of an online gradient descent logistic regression with L1 regularization to reduce the size of our feature space [6]. We believe similar accuracy can be achieved in a distributed fashion with the open source machine learning libraries provided by Spark [33]. We train a model daily over all previously scanned extensions with labeled training data originating from human experts (discussed shortly in Section 6).

Our feature set for classification consist of a collection of over 20 million signals. For each extension we construct a sparse string feature vector that contains every requested permission, the contexts the extension operates on, whether obfuscation was present, and a string representation of all of the extension's file names and directory structure. From dynamic analysis we include a feature for every DOM operation, Chrome API call, XHR request, remotely contacted domain, and a bit for whether the extension uninstalled a security related extension, prevented uninstallation, or modified CSP headers. From the developer analysis we include the email domain, last login geolocation, and a discretized bucket of the developer account's age.

We exclude annotation signals from learning; they are only used by human experts for manually curating rules and analyzing clusters of badness. We also exclude text shingles both to limit our feature space and retain meaningful signals. Our philosophy is that any input to the classifier should have a direct translation to an activity that analysts can recognize rather than loosely contextualized text blobs.

As part of the learning stage, we assign each feature a weight which we optimize using a gradient descent on a logistic regression model. In particular, we use L1 regularization to reduce our feature set to roughly 1,000 of the most impactful features. These features become decision rules, which we use to classify new extensions. Because human reviewers cannot look at every single extensions in the Web Store, we have variable confidence in the malware or benign labels assigned to training instances. To compensate for this, we multiply the gradient descent learning rate with a correction factor that is proportional to an approximate confidence level. Every known malware items gets a correction factor of 1.0 due to prior vetting by a human expert. On the other hand, the learning rate for benign items is scaled down by the following factor:

$$f = \frac{min(\frac{P}{P_t}, 1.0) + min(\frac{A}{A_t}, 1.0)}{2}$$

We represent the popularity of an extension P as the number of existing installs and the age of an extension A as the number of days since the extension was published. P_t and A_t represent thresholds above which we omit any penalty. This correction factor captures the risk that a new extensions with no user base is malicious and yet to be identified, while seasoned extensions with tens of thousands of users are likely benign.

For the sake of tuning the learning pipeline, we use 5-folds cross validation to confirm we do not overfit the model. The final model we use in production is trained on 100% of the data available. For the purposes of our study, we evaluate our model based on its accuracy the next day rather than relying on a holdout golden dataset.

5.2 Manual Rules

We supplement our automated detection with manually curated rules generated by human experts that address many of the most prominent threats facing the Chrome Web Store (discussed later in Section 7). While these rules are fall backs in the event our automated classifier fails, they are immensely helpful in contextualizing the monetization strategy of malicious extensions that we track over time. We note that all extensions surfaced by these rules are still subject to expert verification.

Facebook Hijacking: Initial reports of malicious extensions hijacking a victim's Facebook account to post status updates, send chat messages, befriend users, or "like" content without consent first emerged in 2012 and have persisted ever since [29]. We detect these extensions by scanning network logs produced during dynamic evaluation for outgoing network POSTs to resources (e.g., *ajax/follow/follow_profile.php*) that may indicate unauthorized account behavior.

Ad Injection: Ad injection extensions insert or replace web advertisements. We identify this behavior by comparing the origin of inserted DOM elements and injected scripts against a list of known advertisers derived from third party ad block software, previous reports on ad injection affiliate programs [37, 41], and domains surfaced during manual review. We also scan for DOM operations that replace existing advertisements on any of the pages visited during our behavioral suites where we know ad positions *a priori*.

Search Leakage: Search leakage broadly refers to any extension that funnels search queries to third parties, typically for modifying search results, injecting advertisements, or tracking user interests. We detect search leakage by scanning outgoing network requests to determine whether they contain the same keywords our behavioral suite supplies to *google.com*. This module may potentially miss term leakage in the event of encrypted or obfuscated network parameters.

User Tracking: We rely on a heuristic to detect user tracking that involves scanning all DOM operations for the insertion of 0×0, 1×1, or hidden image during dynamic analysis. We consider any such operation a likely indicator of inserting a tracking pixel.

6 Evaluation

We evaluate WebEval under a live deployment and track daily accuracy as vetted by human experts. As part of our analysis we offer insights into the most important features for classification and the role of human experts in correcting for evasive strains.

6.1 Dataset

Our evaluation dataset consists of 99,818 extensions scored by WebEval between January 2012–2015. Human experts provided our ground truth labels. Due to the possibility of delayed detection we continue to update labels one month after the cut off for our dataset. In total, experts identified 9,523 malicious extensions (9.4% of all extensions created during the same window). For the purposes of our evaluation, we define WebEval's verdict as a *false positive* if WebEval returned a malware label that was either rejected as incorrect by human experts or later refuted by the extension's developer and overturned upon secondary review. Similarly, we define a *false negative* as any extensions surfaced by human experts or external reports despite our system returning a benign verdict. We likely underestimate false negatives as some threats are bound to escape both automated and external review.

6.2 Overall Accuracy

We measure the precision and recall of WebEval as a function of all scored extensions over the last three years. In total, our machine learning pipeline and manually curated rule sets surfaced 93.3% of all known malicious extensions to human experts (recall). Of the extension's that WebEval flagged as potentially malicious, human experts agreed 73.7% of the time (precision). If we restrict our calculation to the last year, WebEval had a recall of 96.5% and a precision of 81%. We find that accuracy is a living process that we detail in Figure 2.

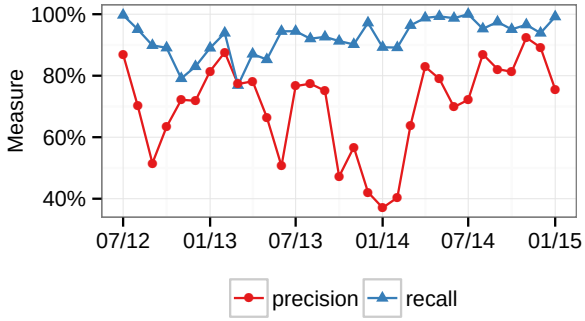

Figure 2: Monthly precision and recall of all scoring systems in aggregate from 2012–2015.

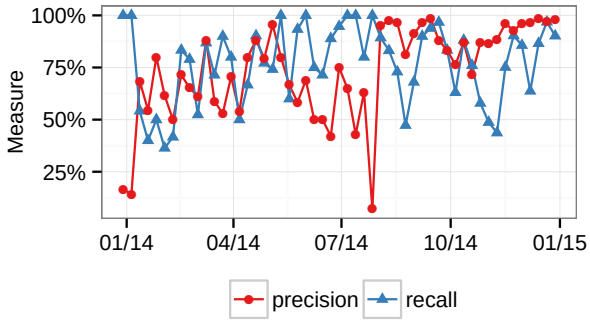

Figure 3: Weekly precision and recall of logistic regression classifier from 2014–2015.

We experience drops in recall when new threats emerge which we subsequently recover from via updated rules and daily retraining of our classifier with new samples. We consistently value recall over precision: we would rather burden human experts with more reviews rather than expose users to malicious extensions. Nevertheless, our precision is reasonable enough as to not force human experts to review every extension in our dataset.

6.3 Automated Classifier Accuracy

Ideally WebEval can run in a purely automated fashion without curated rules or expert verification. We evaluate this possibility by calculating the precision and recall of our logistic model, shown in Figure 3. Over the last year our classifier surfaced 77% of known threats. Human experts agreed with our model's verdict 86% of the time. Overall performance has steadily increased over time with the addition of new features, an increasing training corpus, and increasingly frequent model retraining. Accuracy of the classifier during the final two weeks of our evaluation boasted 98% precision and 91% recall—on par with human experts. However, new threats always require human intervention as indicated by consistent drops in recall throughout time: while the model can quickly recover with daily retraining, we maintain that

Requested Permission	Precision	Recall
tabs	12%	84%
webRequest	23%	39%
webRequestBlocking	22%	27%
notifications	14%	27%
contextMenus	15%	26%
storage	9%	25%
webNavigation	21%	19%
cookies	10%	14%
unlimitedStorage	14%	13%
idle	27%	10%

Table 1: Top 10 permissions requested in extension manifest.

Chrome API	Precision	Recall
runtime.onInstalled	12%	79%
tabs.onUpdated	29%	61%
runtime.connect	21%	50%
extension.getURL	25%	34%
tabs.executeScript	47%	31%
tabs.query	31%	27%
runtime.onConnect	46%	25%
tabs.get	43%	24%
browserAction.setBadgeText	28%	23%
browserAction.setBadgeBackgroundCol...	39%	21%

Table 2: Top 10 Chrome API calls performed during dynamic execution.

DOM Operation	Precision	Recall
eval	10%	76%
Window.navigator	19%	59%
XMLHttpRequest.onreadystatechange	31%	56%
XMLHttpRequest.open	21%	53%
Document.createElement	20%	47%
Window.setTimeout	18%	46%
Node.appendChild	20%	45%
HTMLElement.onload	25%	30%
HTMLScriptElement.src	51%	25%
Window.location	23%	12%

Table 3: Top 10 DOM operations performed during dynamic execution.

Behavioral Signal	Precision	Recall
XHR Request	30%	52%
Code Obfucsation	21%	25%
Script Injected	50%	19%
HTTP 400 Error	41%	9%
Modifies CSP Headers	86%	2%
Uninstalls Extension	96%	0.5%
Prevents Uninstallation	100%	0.1%

Table 4: Precision and recall of individual behavioral signatures.

experts must always be part of our pipeline to minimize both false positives and false negatives. This is an immediate consequence of a centralized market for extensions where there are limited external sources of labeled training data. In contrast, email and telephony spam systems can rely on honeypots and informed users to readily generate representative daily training data. While our human throughput currently scales to the size of the Chrome Web Store, larger ecosystems face a significant challenge for sustainable accuracy.

6.4 Relevance of Individual Signals

WebEval is an amalgam of behavioral signals where no single feature captures the majority of malicious extensions. We examine assumptions we had of certain behaviors, whether they are unique to malware, and which signals are the most important to classification.

Requested Permissions: We list the most popular permissions used by malware and benign extensions in Table 1. These permissions include allowing an extension to trigger when Chrome creates a new tab (84% of all malware) or when Chrome generates a network request (39%). While these behaviors appear fundamental to malware they are equally prevalent in benign applications. This observation captures a significant limitation of the current Chrome permission model as applied towards security judgments: coarse permissions required by all extensions provide no indication that an extension is malicious. Similarly, 93% of all malicious extensions request to interact with every URL as do 57% of all other extensions. These broad contexts make it difficult to determine the pages an extension interacts with, further complicating dynamic analysis.

Chrome API Calls & DOM Operations: We find the strongest features for detecting malware originate from a mixture of Chrome API calls and DOM operations. We provide a list of the most common operations in Table 2 and Table 3. The majority of malware (and benign extensions) rely on injecting scripts, generating XHR requests, and adding new DOM elements that target newly created tabs. What distinguishes the two are the aggregate set of events triggered as well as the domains of remote resources loaded into a page (e.g., injected scripts or content). Our model effectively learns which resources are commonly fetched by malware in addition to common strategies for tampering with pages.

Malicious Logic: Recent work by Kapravelos *et al.* proposed a number of behavioral flags they deemed "suspicious" for extensions. We evaluate the effectiveness of

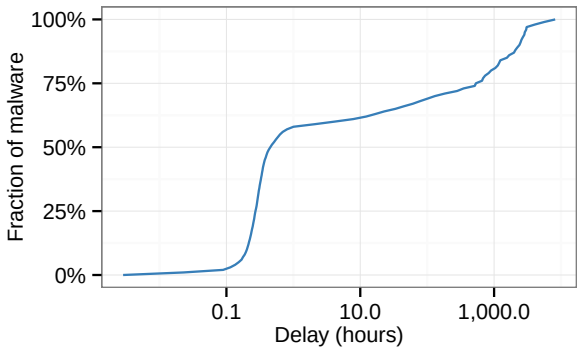

Figure 4: CDF of the delay before catching a malicious extension after it is first submitted to the Chrome Web Store. We catch malicious extensions within a median of 25 minutes.

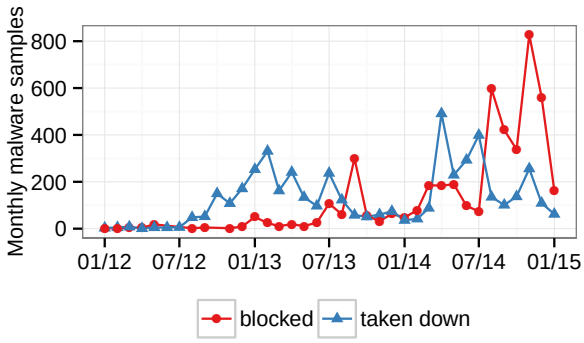

Figure 5: Actions taken against malicious extensions in the Chrome Web Store over time. Our systems are becoming increasingly proactive at blocking malware rather than reactive.

these signals in Table 4. We find that behaviors such as modifying CSP headers, uninstalling extensions, or preventing uninstallation are exclusive to malware, though rare. Contrastingly, Hulk's decision to surface extensions that produce network request errors or inject scripts would overly burden human experts due to low precision. These signals still have value, but they must be combined with the other features collected by our system in order to generate a precise verdict.

6.5 Detection Latency

A critical metric of WebEval's performance is our vulnerability window: the time between when a developer submits a malicious extension to the Chrome Web Store until its detection. This metric represents a worst case scenario where we assume an extension is malicious from its onset rather than after an update or a remote resource begins including malicious functions. Over the last year it took a median of 25 minutes before we flagged an extension as malicious—within the one hour window an extension is embargoed from public access. However, this delay has a long tail as shown in Figure 4. We catch 70% of malicious extensions within 5 days and 90% within 3 months. During this period, users are exposed to malicious content, the impact of which we evaluate in Section 7. Over time, our verdicts have become increasingly proactive rather than reactive as shown in Figure 5. Blocked extensions never reach the public, while extensions taken down by the Chrome Web Store leave users vulnerable for a short period. As we discuss shortly, proactive blocking has a substantial impact on reducing the number of known victims exposed to malware.

6.6 Manual Review Effort

WebEval relies heavily on human experts to validate the verdicts of automated classification and manual rules to guarantee high precision. In the last year, we surfaced 10,120 suspicious extensions for review, entailing a total of 464 hours of analysis—an average of 2.75 minutes per extension. This process is simplified by access to all of WebEval's dynamic and static analysis and concrete training features as previously discussed in Section 4 and Section 5. We recognize that manual review by experts represents a scarce resource that is challenging to scale. Consequently, we continuously look for ways to improve automated verdicts to achieve a precision on par with human experts.

7 Trends in Malicious Extensions

Consistently high recall over the last three years allows us to provide a retrospective on how malicious extensions have evolved over time. This includes the monetization vectors used, the breadth of users impacted, and the developers responsible.

7.1 Abuse Vectors

Despite hundreds of new monthly malicious extensions, we find the strategies for abusing Chrome users have remained largely constant. Figure 6 shows a breakdown of abuse strategies of extensions per month where a manually curated label is available;[4] we categorize extensions flagged by automated systems that provide no context on abuse vectors as "other". Noticeably absent from the top threats are banking trojans, password theft, and email spam. While these are all within the realm of a malicious extension's capabilities—and have cropped up from time to time—such threats are dwarfed by Facebook hijacking, ad injection, and information theft.

[4]Labels are not guaranteed to be unique; an extension can simultaneously hijack Facebook credentials, inject ads, and insert tracking pixels.

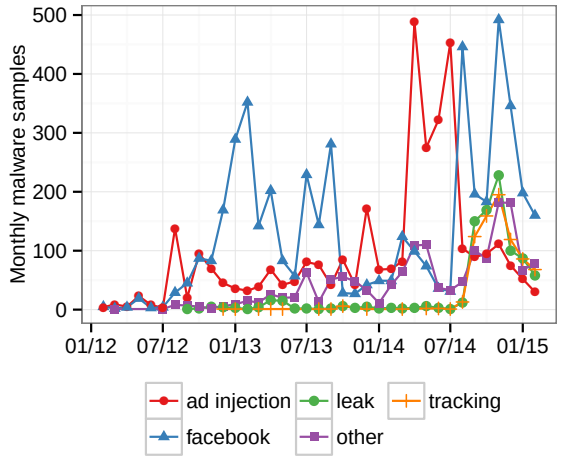

Figure 6: Malware varietals detected each month from 2012–2015.

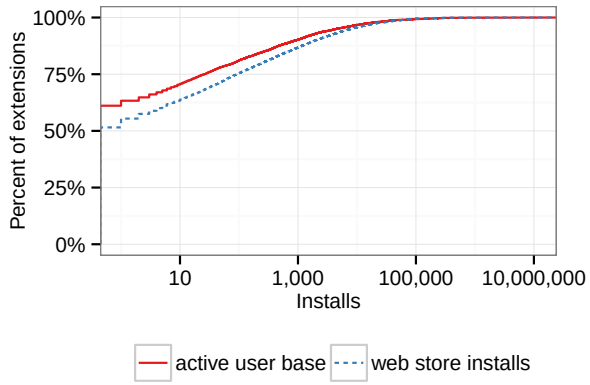

Figure 7: CDF of installs broken down by those originating from the Chrome Web Store and an extension's active pings that capture both store-based and sideloaded installs.

Facebook Hijacking: Our findings show that Facebook malware remains a persistent threat with over 4,809 variants in the last three years. These malicious extensions purport to offer enhancements such as removing the Facebook Timeline, adding a "dislike" button, or changing the theme of the Facebook interface. The hook has evolved over time. The latest rendition tricks users into installing an extension masquerading as a video codec required to view a video posted by a friend. Once installed, the extension hijacks the victim's credentials to post status updates that propagate the malware. How the extensions monetize Facebook accounts is not entirely clear, but appears to involve inflating likes, fans, and friend counts much like previously studied fake engagement contagions on Twitter [36, 38].

Ad Injection: Ad injection is the second most prevalent threat in the Chrome Web Store comprising 3,496 extensions. These extensions rely on content scripts that run on every page that allow the extension to scan DOMs for common banners to replace with rogue advertisements or simply insert new ads into pages. We note that ad injection is not expressly prohibited by the Chrome Web Store: the extensions flagged also performed some other malicious behavior or violated one of the store's policies as determined by a human expert.

Other Variants: In recent months we have witnessed a larger variety of abuse vectors. In depth investigations of a sample of these extensions reveal malware tampering with bitcoin wallets, injecting into banking sessions for Brazilian institutions, and modifying Amazon affiliate URLs. While we lack manual rules for these specific abuse vectors, we are nevertheless able to catch them via our classifier.

7.2 Installs

Malicious extensions obtain installs in one of two fashions: (1) via binaries that modify Chrome's user profile to sideload extensions,[5] or (2) via social engineering where miscreants direct users to the Chrome Web Store or prompt users with an install dialogue on a third-party site. We measure both approaches using two metrics. We define an extension's *active user base* as the total number of Chrome clients who ping the Chrome Web Store with update requests (sent by all extensions, including sideloaded extensions). This value changes each day as users install or uninstall extensions, so we select the all-time maximum. We define an extension's *web store installs* as the total number of install dialogues Chrome clients initiate with the Chrome Web Store. We note that a third option exists for miscreants to obtain installs: paying an existing, legitimate extension developer to hand over their app. In practice, we found only 6 malicious extensions (0.06%) that involved an ownership transfer.

Evidence of Side Loading: We provide a breakdown of both install metrics in Figure 7. We find that 51% of malicious extensions never received any active user base or Web Store installs due to early detection. Evidence of sideloading is relatively rare: only 290 extensions had a larger active user base than Web Store installs. However, these extensions were immensely popular with over 43.5 million combined active users. In contrast, all malicious extensions combined received 29.6 million installs via the Chrome Web Store. As such, it would appear that binary distribution of malicious extensions contributed substantially to user infections. This allows malware authors to rely on the same distribution models of the past (e.g., drive-by downloads [30], exploit packs [18], pay-per-install [8]) while tapping into the extension API as a means for simplifying exploitation.

[5]The extension still must be in the Chrome Web Store due to the lockdown policy discussed previously in Section 2

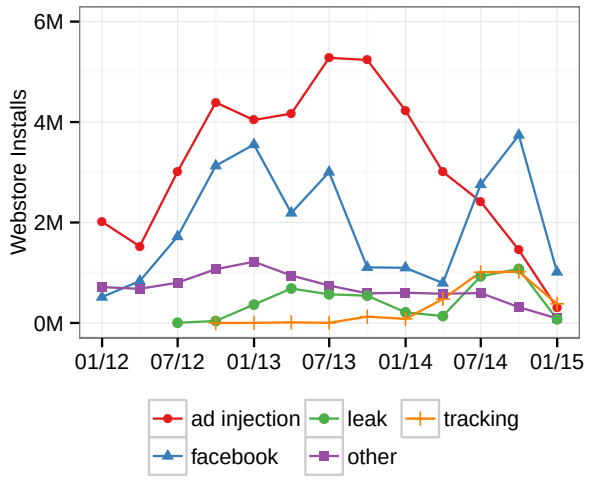

Figure 8: Malware installations via the Chrome Web Store for the past three years broken down by abuse vector. This excludes binaries sideloading extensions.

Country	Infected Users	Popularity
United States	2,375,363	8%
Brazil	1,982,570	7%
Mexico	1,942,288	6%
Colombia	1,634,933	5%
Turkey	1,569,949	5%
Argentina	1,525,364	5%
India	1,475,228	5%
Russia	1,244,932	4%
Peru	955,695	3%
Vietnam	806,625	3%

Table 5: Top 10 regions impacted by malicious extensions downloaded via the Chrome Web Store.

Equally problematic, installs follow a long tail distribution. We find that 64 extensions (1% of all malware) attained an aggregate 46.6 million active users, 83% of all installations. The top two most popular threats were ad injectors and search hijackers that each garnered over 10 million active users. Miscreants distributed each extension solely via binaries flagged as malware by Google Safe Browsing. Our results emphasize that seemingly small false negative detection rates can have substantial negative impact on Chrome users. This drastically differs from email and telephony spam where an incorrectly labeled message typically impacts only a single user—not millions.

Popular Social Engineering Campaigns: Focusing exclusively on installs mediated by the Chrome Web Store, we investigate which abuse vectors achieved the most new installs per month and the country of origin of installs. Figure 8 tracks the rise and fall of various monetization strategies over time. Despite a short window

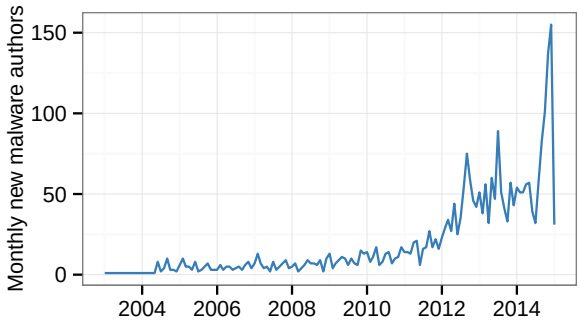

Figure 9: Registration time of malware authors. Most authors rely on accounts created in the last three years.

before catching malicious extensions we still find that social engineering campaigns enticed millions of new users each month to install Facebook malware, ad injection software, and information stealers. The downward trend in recent months is the result of proactive blocking rather than retroactive takedowns that expose users to malware for a short window. We find no single country is disproportionately represented as the source of installs, as shown in Table 5. Our results highlight the global scale and negative impact that malicious extensions have on users and the need for greater research attention to the problem.

7.3 Malicious Developers

We identify 2,339 malicious extension developers throughout the course of our study. While 50% of developers authored their malicious extension within 3 to 4 months of registering, there is a long tail of potentially compromised accounts used to interact with the Chrome Web Store as shown in Figure 9. We find miscreants access 31% of developer accounts from IPs within Turkey followed in popularity by range of other countries detailed in Table 6. Many of the countries with the highest infection counts were also prominent locations for malicious developers indicating threats were likely localized. Re-use of malicious developer accounts was fairly limited: 50% of accounts authored fewer than 2 malicious extensions while 90% authored fewer than 10.

8 Discussion

With multiple years spent fighting malicious extensions, we reflect on some of the lessons we have learned, limitations of our approach, and potential technical and policy improvements that can help prevent malicious extensions in the future.

Country	Developers	Popularity
Turkey	552	31%
United States	164	9%
Dominican Republic	126	7%
Brazil	106	6%
Vietnam	83	4%
Russia	60	3%
Germany	43	2%
Peru	43	2%
India	43	2%
Israel	39	2%

Table 6: Top 10 login geolocations of malicious developers.

8.1 Lessons Learned

When we first set out to identify malicious extensions our expectation was to find banking trojans and password stealers that duplicated the strategies pioneered by Zeus and SpyEye. In practice, the abusive extension ecosystem is drastically different from malicious binaries. Monetization hinges on direct or indirect relationships with syndicated search partners and ad injection affiliate programs, some of which earn millions of dollars from infected users [37]. Miscreants derive wealth from *traffic* and *user targeting* rather than the computing resources or privileged access mediated via the browser. It may simply be that the authors of malicious binaries have little incentive (or external pressure) to change, leaving extensions to a distinct set of actors. This uncertainty is a strong motivation for exploring the extension ecosystem further.

A second lesson is the importance of equipping an abuse prevention team with the tools necessary to rapidly respond to new, unforeseen threats. As we have shown, even momentary lapses in protection have drastic consequences on Chrome users. This is especially true in the case of social engineering campaigns like those used to distribute malicious Facebook extensions that spread exponentially. We argue that evaluating a detection system purely on precision and recall is not effective when the ultimate goal is to protect users from malware installations. Instead, we must weigh false negatives by their consequences—the number of victims exposed to malware. In this light, our system has continuously improved over the last three years.

In the long term we believe the Chrome Web Store must extricate itself from the current fire-fighting approach to malicious extensions and outright disrupt the malicious actors involved. This reflects a nascent strategy within the research community to pursue criminal relationships such as those underpinning spammed pharmaceuticals [26]. However, to arrive at this point we must first lay a foundation for how to study the extension ecosystem. As the research community develops the necessary understanding this abuse space—and in particular the ad and search relationships involved—there must be a system to both protect users as well as generate longitudinal data on abuse strategies and their support infrastructure. WebEval satisfies both of these requirements.

8.2 Role of Policy

Research primarily considers technical solutions to abuse, but we argue that policy decisions prove equally effective at protecting users. When Chrome first released extensions there was no requirement of developers uploading their code to the Chrome Web Store. This enabled malicious developers to side-load extensions via binaries and left Chrome users with little room for discovering the installation or recourse. The subsequent Chrome lockdown forced all malicious extensions to at least be surfaced to the Chrome Web Store and created a homogeneous enforcement policy for all Chrome users. While binaries can still side-load extensions in the Chrome Web Store, WebEval now incorporates signals to detect organic versus silent installs.

It is worth noting the Chrome lockdown policy has some limitations. Anecdotally, we have observed binaries distributing payloads that overwrite the local content of legitimate extensions previously installed by a user. Because only the legitimate extension is in the store, WebEval cannot offer any protection. Chrome has since responded to this threat by introducing extension content verification, but this is just a single stage in an increasing arms race.

8.3 Limitations

Dynamic analysis and security crawlers consistently run the risk of overlooking malicious behaviors due to cloaking [2, 23, 31]. Extension analysis is equally vulnerable. Potential threats include malware delaying execution until after WebEval's evaluation; supplying benign versions of remotely fetched JavaScript until after evaluation; or malware developers fingerprinting our evaluation environment and IP addresses. A separate issue is code coverage: our behavioral suites are not guaranteed to trigger all of an extension's logic during evaluation. Worse, we face an intractably broad threat surface that we must test as the majority of malware requests access to every page a user visits. While symbolic execution systems exist for Javascript [32], they rely on fuzzing that is not guaranteed to trigger malicious behavior due to the implicit event-driven nature of extensions where activation requires a specific sequence of listeners to fire. Solutions to these challenges remain elusive; we currently rely on human experts and abuse reports to surface false negatives so we can adapt our detection framework.

8.4 Improving Detection

Fundamentally improving WebEval (and by proxy other security scanners) requires we break from evaluating extensions in a sandboxed environment vulnerable to cloaking and instead move to *in situ* monitoring of Chrome users. This strategy, previously considered by researchers to improve drive-by download detection [35], applies equally to malicious extensions. However, such a move creates a new challenge of balancing early infections of clients, user privacy, and anonymous but feature-rich reporting of an extension's behaviors with enough details to detect malice.

Furthermore, while we can retrain our a model of malicious extensions to incorporate client logs, the process would be immensely aided by the cooperation of website developers who label DOM resources as sensitive. We should take these labels as hints, not facts, to account for overzealous developers who label every DOM element as sensitive in an effort to dissuade extension modifications, even when desired by users. We believe this combined approach strikes the best balance between Chrome's current philosophy of allowing users to alter their browsing experience in any way with the necessity of early detection of malicious modifications.

9 Related Work

Security Sandboxes & Malware Detection: WebEval borrows heavily from a history of malware analysis sandboxes that capture system calls and network traffic. Examples include Anubis [5], CWSandbox [40], and GQ [24] among a breadth other architectures [12]. However, malicious extensions pose a unique set of challenges that limit the effectiveness of these sandboxes without modification. Unlike standalone applications, Chrome extensions run in the context of a webpage making it harder for traditional system-wide malware monitoring techniques to isolate malware activity from that of the browser. Our system manages to achieve this isolation by comparing extension activity to baseline activity captured while the extension was not running as well as by tapping natively into Chrome's JavaScript and API modules.

The closest system to our own is Hulk which captures in-browser activity logs [21]. Unlike Hulk, our system goes beyond identifying suspicious behaviors to return a concrete verdict of malice. This is imperative as the signals proposed by Hulk are insufficient at detecting most malicious extensions as we showed in Section 6. Research has also explored competing strategies such as information flow tracking in JavaScript with tainted inputs [11] or tracking common API calls made by Browser Helper Objects installed by adware [22]. These techniques influence our design but only capture a subset of the malicious extensions we identify.

Buggy & Malicious Extensions: Most research into browser extensions has focused on their security and permission model in light of the possible vulnerabilities [3, 4, 9, 19]. Only recently has research shifted towards the threat of outright malicious extensions. This includes re-imagining application-based attacks as man-in-the-browser threats [25]; examining the role of extensions in the ad injection ecosystem [37, 41]; and characterizing malicious extensions found in the Chrome Web Store [21]. Our observations agree with many of these former studies. We expand upon these works by offering a complete perspective of how malicious extension monetization techniques have evolved over the last three years and the techniques malware developers use to distribute extensions.

10 Conclusion

In this work we exposed wide-spread efforts by criminals to abuse the Chrome Web Store as a platform for distributing malicious extensions. As part of our study, we presented the design and implementation of a framework that automatically classifies an extension's behaviors, code base, and author reputation to surface malware. Due to our live deployment, this system cannot run in a fully automated fashion: we required regular inputs from human experts to correct for false negatives surfaced via Chrome user reports and manual investigations. Our unique combination of automated and human systems yielded a framework that identified 96.5% of all known malware submitted to the Chrome Web Store between January 2012–2015.

In total, we detected 9,523 malicious extensions that hijacked social networking sessions to generate synthetic likes, friend requests, and fans; ad injectors and affiliate fraudsters that rewrote DOM content to laden pages with additional advertisements; and information stealers that injected rogue tracking pixels and covertly siphoned search keywords. Despite a short window of operation—we disabled 50% of malware within 25 minutes of creation—a handful of under 100 malicious extensions were able to infect over 50 million users before removal. Our results highlight key challenges of protecting app marketplaces that are broadly applicable beyond the Chrome Web Store.

References

[1] Apple. App Review. `https://developer.apple.com/app-store/review/`, 2015.

[2] Davide Balzarotti, Marco Cova, Christoph Karlberger, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Efficient de-

tection of split personalities in malware. In *Proceedings of the Network and Distributed System Security Conference*, 2010.

[3] Sruthi Bandhakavi, Samuel T King, Parthasarathy Madhusudan, and Marianne Winslett. Vex: Vetting browser extensions for security vulnerabilities. In *Proceedings of the USENIX Security Symposium*, 2010.

[4] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting browsers from extension vulnerabilities. In *Proceedings of the Network and Distributed System Security Conference*, 2010.

[5] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *Proceedings of the Network and Distributed System Security Conference*, 2009.

[6] Jeremy Bem, Georges R Harik, Joshua L Levenberg, Noam Shazeer, and Simon Tong. Large scale machine learning systems and methods, 2007. US Patent 7,222,127.

[7] Christina Bonnington. First instance of ios app store malware detected, removed. `http://www.wired.com/2012/07/first-ios-malware-found/`, 2012.

[8] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring pay-per-install: The commoditization of malware distribution. In *Proceedings of the USENIX Security Symposium*, 2011.

[9] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. An evaluation of the google chrome extension security architecture. In *Proceedings of the USENIX Security Symposium*, 2012.

[10] Lucian Constantin. Malicious browser extensions pose a serious threat and defenses are lacking. `http://www.pcworld.com/article/2049540/malicious-browser-extensions-pose-a-serious-threat-and-defenses-are-lacking.html`, 2014.

[11] Mohan Dhawan and Vinod Ganapathy. Analyzing information flow in javascript-based browser extensions. In *Proceedings of the Annual Computer Security Applications Conference*, 2009.

[12] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys*, 2012.

[13] Erik Kay. Protecting Chrome users from malicious extensions. `http://chrome.blogspot.com/2014/05/protecting-chrome-users-from-malicious.html`, 2014.

[14] Firefox. Review Process. `https://addons.mozilla.org/en-US/developers/docs/policies/reviews`, 2015.

[15] Google. Developer registration fee. `https://support.google.com/chrome_webstore/answer/187591?hl=en`, 2015.

[16] Google. Google Chrome Web Store Developer Agreement. `https://developer.chrome.com/webstore/terms`, 2015.

[17] Google. Google Play Developer Program Policies. `https://play.google.com/about/developer-content-policy.html`, 2015.

[18] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, et al. Manufacturing compromise: the emergence of exploit-as-a-service. In *Proceedings of the Conference on Computer and Communications Security*, 2012.

[19] Arjun Guha, Matthew Fredrikson, Benjamin Livshits, and Nikhil Swamy. Verified security for browser extensions. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2011.

[20] Scott Kaplan, Benjamin Livshits, Benjamin Zorn, Christian Siefert, and Charlie Curtsinger. " nofus: Automatically detecting"+ string. fromcharcode (32)+" obfuscated". tolowercase ()+" javascript code. In *Technical Report, Microsoft*, 2011.

[21] Alexandros Kapravelos, Chris Grier, Neha Chachra, Chris Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *Proceedings of the USENIX Security Symposium*, 2014.

[22] Engin Kirda, Christopher Kruegel, Greg Banks, Giovanni Vigna, and Richard Kemmerer. Behavior-based spyware detection. In *Proceedings of the USENIX Security Symposium*, 2006.

[23] Clemens Kolbitsch, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Rozzle: De-cloaking internet malware. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.

[24] Christian Kreibich, Nicholas Weaver, Chris Kanich, Weidong Cui, and Vern Paxson. Gq: Practical containment for measuring modern malware systems. In *Proceedings of the ACM SIGCOM Internet Measurement Conference*, 2011.

[25] Lei Liu, Xinwen Zhang, Guanhua Yan, and Songqing Chen. Chrome extensions: Threat analysis and countermeasures. In *Proceedings of the Network and Distributed System Security Conference*, 2012.

[26] Damon McCoy, Hitesh Dharmdasani, Christian Kreibich, Geoffrey M. Voelker, and Stefan Savage. Priceless: The role of payments in abuse-advertised goods. In *Proceedings of the Conference on Computer and Communications Security*, 2012.

[27] Paul Pearce, Vacha Dave, Chris Grier, Kirill Levchenko, Saikat Guha, Damon McCoy, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Characterizing large-scale click fraud in zeroaccess. In *Proceedings of the Conference on Computer and Communications Security*, 2014.

[28] Adrienne Porter Felt. See what your apps & extensions have been up to. `http://blog.chromium.org/2014/06/see-what-your-apps-extensions-have-been.html`, 2015.

[29] Emil Protalinski. Malicious Chrome extensions hijack Facebook accounts. `http://www.zdnet.com/article/malicious-chrome-extensions-hijack-facebook-accounts/`, 2012.

[30] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iFRAMEs point to us. In *Proceedings of the USENIX Security Symposium*, 2008.

[31] M Rajab, Lucas Ballard, Nav Jagpal, Panayiotis Mavrommatis, Daisuke Nojiri, Niels Provos, and Ludwig Schmidt. Trends in circumventing web-malware detection. In *Google Technical Report*, 2011.

[32] Prateek Saxena, Devdatta Akhawe, Steve Hanna, Feng Mao, Stephen McCamant, and Dawn Song. A symbolic execution framework for JavaScript. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2010.

[33] Spark. Machine learning library (mllib) programming guide. `http://spark.apache.org/docs/1.4.0/mllib-guide.html`, 2015.

[34] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the Conference on Computer and Communications Security*, 2009.

[35] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the Conference on Computer and Communications Security*, 2013.

[36] Gianluca Stringhini, Gang Wang, Manuel Egele, Christopher Kruegel, Giovanni Vigna, Haitao Zheng, and Ben Y Zhao. Follow the green: Growth and dynamics in twitter follower markets. In *Proceedings of the ACM SIGCOM Internet Measurement Conference*, 2013.

[37] Kurt Thomas, Elie Bursztein, Chris Grier, Grant Ho, Nav Jagpal, Alexandros Kapravelos, Damon McCoy, Antonio Nappa, Vern Paxson, Paul Pearce, Niels Provos, and Moheeb Abu Rajab. Ad injection at scale: Assessing deceptive advertisement modifications. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2015.

[38] Kurt Thomas, Frank Li, Chris Grier, and Vern Paxson. Consequences of connectivity: Characterizing account hijacking on twitter. In *Proceedings of the Conference on Computer and Communications Security*, 2014.

[39] Kurt Thomas, Damon McCoy, Chris Grier, Alek Kolcz, and Vern Paxson. Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse. In *Proceedings of the USENIX Security Symposium*, 2013.

[40] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2007.

[41] Xinyu Xing, Wei Meng, Udi Weinsberg, Anmol Sheth, Byoungyoung Lee, Wenke Lee, and Roberto Perdisci. Unraveling the relationship between ad-injecting browser extensions and malvertising. In *Proceedings of the International Conference on the World Wide Web*, 2015.

[42] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proceedings of the Network and Distributed System Security Conference*, 2012.