Include Me Out: In-Browser Detection of Malicious Third-Party Content Inclusions

Sajjad Arshad, Amin Kharraz, and William Robertson
Northeastern University, Boston, USA
{arshad,mkharraz,wkr}@ccs.neu.edu

Abstract. Modern websites include various types of third-party content such as JavaScript, images, stylesheets, and Flash objects in order to create interactive user interfaces. In addition to explicit inclusion of third-party content by website publishers, ISPs and browser extensions are hijacking web browsing sessions with increasing frequency to inject third-party content (e.g., ads). However, third-party content can also introduce security risks to users of these websites, unbeknownst to both website operators and users. Because of the often highly dynamic nature of these inclusions as well as the use of advanced cloaking techniques in contemporary malware, it is exceedingly difficult to preemptively recognize and block inclusions of malicious third-party content before it has the chance to attack the user's system.

In this paper, we propose a novel approach to achieving the goal of preemptive blocking of malicious third-party content inclusion through an analysis of inclusion sequences on the Web. We implemented our approach, called Excision, as a set of modifications to the Chromium browser that protects users from malicious inclusions while web pages load. Our analysis suggests that by adopting our in-browser approach, users can avoid a significant portion of malicious third-party content on the Web. Our evaluation shows that Excision effectively identifies malicious content while introducing a low false positive rate. Our experiments also demonstrate that our approach does not negatively impact a user's browsing experience when browsing popular websites drawn from the Alexa Top 500.

Keywords: Web security, Malvertising, Machine learning

1 Introduction

Linking to third-party content has been one of the defining features of the World Wide Web since its inception, and this feature remains strongly evident today. For instance, recent research [28] reveals that more than 93% of the most popular websites include JavaScript from external sources. Developers typically include third-party content for convenience and performance – e.g., many JavaScript libraries are hosted on fast content delivery networks (CDNs) and are likely to already be cached by users – or to integrate with advertising networks, analytics frameworks, and social media. Third-party content inclusion has also been used

by entities other than the website publishers themselves. For example, ad injection has been adopted by ISPs and browser extension authors as a prominent technique for monetization [25].

However, the inherent feature of content-sharing on the Web is also an Achilles heel when it comes to security. Advertising networks, as one example, have emerged as an important vector for adversaries to distribute attacks to a wide audience [21, 22, 29, 36, 43]. Moreover, users are more susceptible to *malvertising* in the presence of ad injection [17, 38, 42]. In general, linking to third-party content is essentially an assertion of trust that the content is benign. This assertion can be violated in several ways, however, due to the dynamic nature of the Web. Since website operators cannot control external content, they cannot know *a priori* what links will resolve to in the future. The compromise of linked content or pure malfeasance on the part of third parties can easily violate these trust assumptions. This is only exacerbated by the transitive nature of trust on the Web, where requests for content can be forwarded beyond the first, directly observable origin to unknown parties.

While the same origin policy (SOP) enforces a modicum of origin-based separation between code and data from different principals, developers have clamored for more flexible sharing models provided by, e.g., Content Security Policy (CSP) [7], Cross-Origin Resource Sharing (CORS) [6], and postMessage-based cross-frame communication. These newer standards permit greater flexibility in performing cross-origin inclusions, and each come with associated mechanisms for restricting communication to trusted origins. However, recent work has shown that these standards are difficult to apply securely in practice [34, 40], and do not necessarily address the challenges of trusting remote inclusions on the dynamic Web. In addition to the inapplicability of some approaches such as CSP, third parties can leverage their power to bypass these security mechanisms. For example, ISPs and browser extensions are able to tamper with HTTP traffic to modify or remove CSP rules in HTTP responses [17, 38].

In this paper, we propose an in-browser approach called Excision to automatically detect and block malicious third-party content inclusions as web pages are loaded into the user's browser or during the execution of browser extensions. Our approach does not rely on examination of the content of the resources; rather, it relies on analyzing the sequence of inclusions that leads to the resolution and loading of a terminal remote resource. Unlike prior work [22], Excision resolves *inclusion sequences* through instrumentation of the browser itself, an approach that provides a high-fidelity view of the third-party inclusion process as well as the ability to interdict content loading in real-time. This precise view also renders ineffective common obfuscation techniques used by attackers to evade detection. Obfuscation causes the detection rate of these approaches to degrade significantly since obfuscated third-party inclusions cannot be traced using existing techniques [22]. Furthermore, the in-browser property of our system allows users to browse websites with a higher confidence since malicious third-party content is prevented from being included while the web page is loading.

We implemented EXCISION as a set of modifications to the Chromium browser, and evaluated its effectiveness by analyzing the Alexa Top 200K over a period of 11 months. Our evaluation demonstrates that EXCISION achieves a 93.39% detection rate, a false positive rate of 0.59%, and low performance overhead. We also performed a usability test of our research prototype, which shows that EXCISION does not detract from the user's browsing experience while automatically protecting the user from the vast majority of malicious content on the Web. The detection results suggest that EXCISION could be used as a complementary system to other techniques such as CSP.

The main contributions of this paper are as follows:

- We present a novel in-browser approach called EXCISION that automatically detects and blocks malicious third-party content before it can attack the user's browser. The approach leverages a high-fidelity in-browser vantage point that allows it to construct a precise inclusion sequence for every third-party resource.
- We describe a prototype of EXCISION for the Chromium browser that can effectively prevent inclusions of malicious content.
- We evaluate the effectiveness and performance of our prototype, and show that it is able to automatically detect and block malicious third-party content inclusions in the wild – including malicious resources not previously identified by popular malware blacklists – without a significant impact on browser performance.
- We evaluate the usability of our prototype and show that most users did not notice any significant quality impact on their browsing experience.

2 Problem Statement

In the following, we first discuss the threats posed by third-party content and then motivate our work.

2.1 Threats

While the inclusion of third-party content provides convenience for web developers and allows for integration into advertising distribution, analytics, and social media networks, it can potentially introduce a set of serious security threats for users. For instance, advertising networks and social media have been and continue to be abused as a vector for injection of malware. Website operators, or publishers, have little control over this content aside from blind trust or security through isolation. Attacks distributed through these vectors – in the absence of isolation – execute with the same privileges as all other JavaScript within the security context of the enclosing DOM. In general, malicious code could launch drive-by downloads [10], redirect visitors to phishing sites, generate fraudulent clicks on advertisements [22], or steal user information [16].

Moreover, ad injection has become a new source of income for ISPs and browser extension authors [25]. ISPs inject advertisements into web pages by

tampering with their users' HTTP traffic [9], and browser extension authors have recently started to inject or replace ads in web pages to monetize their work. Ad injection negatively impacts both website publishers and users by diverting revenue from publishers and exposing users to malvertising [38, 42]. In addition to ad injection, malicious browser extensions can also pose significant risks to users due to the special privileges they have [19].

2.2 Motivation

Publishers can try to isolate untrusted third-party content using iframes (perhaps enhanced with HTML5 sandboxing features), language-based sandboxing, or policy enforcement [1, 12, 15, 23, 24]. However, these approaches are not commonly used in practice; some degrade the quality of ads (from the advertiser's perspective), while others are non-trivial to deploy. Publishers could attempt to use Content Security Policy (CSP) [7] to define and enforce access control lists for remote inclusions in the browser. However, due to the dynamic nature of the web, this approach (and similar access control policy-based techniques) has problems. Recent studies [34, 40] indicate that CSP is difficult to apply in practice. A major reason for this is the unpredictability of the origins of inclusions for third-party resources, which complicates the construction of a correct, yet tight, policy.

For example, when websites integrate third-party advertisements, multiple origins can be contacted in order to deliver an ad to the user's browser. This is often due to the practice of re-selling ad space (a process known as ad syndication) or through real-time ad auctions. Either of these approaches can result in ads being delivered through a series of JavaScript code inclusions [35]. Additionally, the growing number of browser extensions makes it a non-trivial task for website operators to enumerate the set of benign origins from which browser extensions might include a resource. As an example, for theverge.com website, the number of unique included domains over a period of 11 months increases roughly linearly; clearly, constructing an explicit whitelist of domains is a challenging task.

Even if website publishers can keep pace with origin diversity over time with a comprehensive list of CSP rules, ISPs and browser extensions are able to tamper with in-transit HTTP traffic and modify CSP rules sent by the websites. In addition, in browsers such as Chrome, the web page's CSP does not apply to extension scripts executed in the page's context [2]; hence, extensions are able to include arbitrary third-party resources into the web page.

Given the challenges described above, we believe that existing techniques such as CSP can be evaded and, hence, there is a need for an automatic approach to protect users from malicious third-party content. We do not necessarily advocate such an approach in isolation, however. Instead, we envision this approach as a complementary defense that can be layered with other techniques in order to improve the safety of the Web.

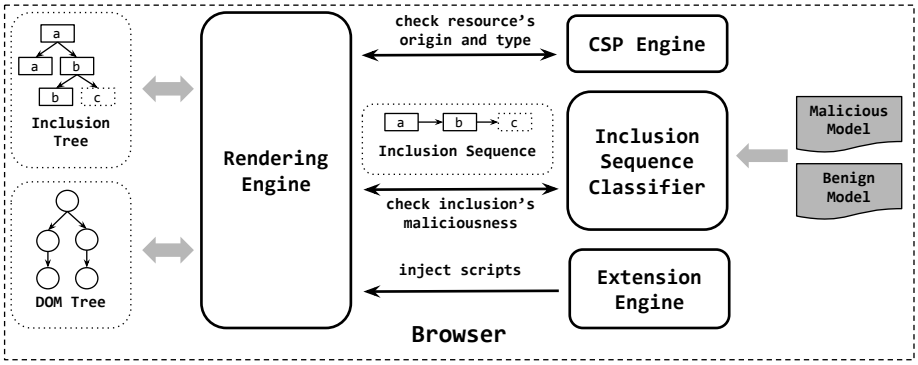


Fig. 1: An overview of Excision.

3 Excision

In this section, we describe Excision, our approach for detecting and blocking the inclusion of malicious third-party content in real-time. An overview of our system is shown in Figure 1. Excision operates by extracting resource *inclusion trees* from within the browser. The inclusion tree precisely records the inclusion relationships between different resources in a web page. When the user requests a web page, the browser retrieves the corresponding HTML document and passes it to the rendering engine. The rendering engine incrementally constructs an inclusion tree for the DOM and begins extracting external resources such as scripts and frames as it reaches new HTML tags. For inclusion of a new resource, the rendering engine consults the CSP engine and the *inclusion sequence classifier* in order to decide whether to include the resource. If the resource's origin and type are whitelisted in the CSP rules, the rendering engine includes the resource without consulting the inclusion sequence classifier and continues parsing the rest of the HTML document. Otherwise, it extracts the *inclusion sequence* (path through the page's inclusion tree) for the resource and forwards this to the inclusion sequence classifier. Using pre-learned models, the classifier returns a decision about the malice of the resource to the rendering engine. Finally, the rendering engine discards the resource if it was identified as malicious. The same process occurs for resources that are included dynamically during the execution of extension scripts after they are injected into the page.

3.1 Inclusion Trees and Sequences

A website can include resources in an HTML document from any origin so long as the inclusion respects the same origin policy, its standard exceptions, or any additional policies due to the use of CSP, CORS, or other access control framework. A first approximation to understanding the inclusions of third-party content for a given web page is to process its DOM tree [41] while the page loads. However,

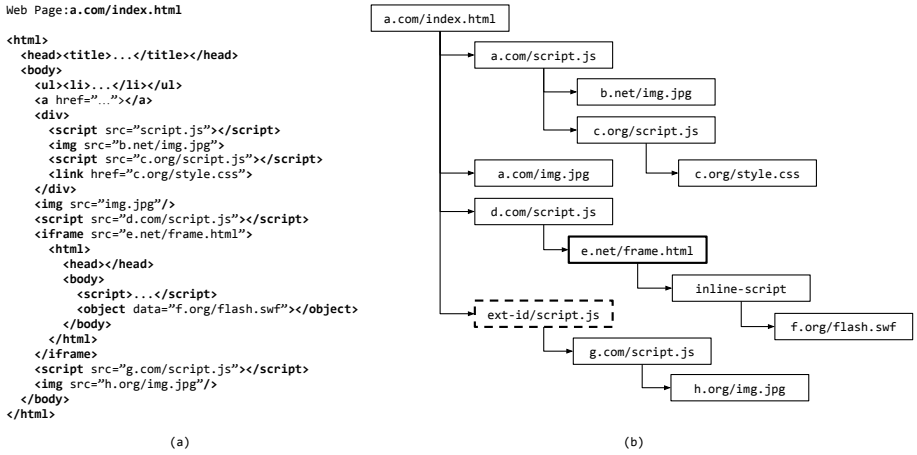


Fig. 2: (a) DOM Tree, and (b) Inclusion Tree

direct use of a web page's DOM tree is unsatisfactory because the DOM does not in fact reliably record the inclusion relationships between resources referenced by a page. This follows from the ability for JavaScript to manipulate the DOM at run-time using the DOM API.

Instead, in this work we define an *inclusion tree* abstraction extracted directly from the browser's resource loading code. Unlike a DOM tree, the inclusion tree represents how different resources are included in a web page that is invariant with respect to run-time DOM updates. It also discards irrelevant portions of the DOM tree that do not reference remote content. For each resource in the inclusion tree, there is an *inclusion sequence* that begins with the root resource (i.e., the URL of the web page) and terminates with the corresponding resource. Furthermore, browser extensions can also manipulate the web page by injecting and executing JavaScript code in the page's context. Hence, the injected JavaScript is considered a direct child of the root node in the inclusion tree. An example of a DOM tree and its corresponding inclusion tree is shown in Figure 2. As shown in Figure 2b, `f.org/flash.swf` has been dynamically added by an `inline script` to the DOM tree, and its corresponding inclusion sequence has a length of 4 since we remove the inline resources from inclusion sequence. Moreover, `ext-id/script.js` is injected by an extension as the direct child of the root resource. This script then included `g.com/script.js`, which in turn included `h.org/img.jpg`.

3.2 Inclusion Sequence Classification

Given an inclusion sequence, Excision must classify it as benign or malicious based on features extracted from the sequence. The task of the *inclusion sequence classifier* is to assign a class label from the set {benign,malicious} to a given sequence based on previously learned models from a labeled data set. In our

definition, a malicious sequence is one that starts from the root URL of a web page and terminates in a URL that delivers malicious content. For classification, we used hidden Markov models (HMM) [31]. Models are comprised of states, each of which holds transitions to other states based on a probability distribution. Each state can probabilistically emit a symbol from an alphabet. There are other sequence classification techniques such as Naïve Bayes [20], but we used an HMM for our classifier because we also want to model the inter-dependencies between the resources that compose an inclusion sequence.

In the training phase, the system learns two HMMs from a training set of labeled sequences, one for the benign class and one for the malicious class. We estimated the HMM parameters by employing the Baum-Welch algorithm which finds the maximum likelihood estimate of these parameters based on the set of observed sequences. In our system, we empirically selected 20 for the number of states that are fully connected to each other. In the subsequent detection phase, we compute the likelihood of a new sequence given the trained models using the forward-backward algorithm and assign the sequence to the class with the highest likelihood. Training hidden Markov models is computationally expensive. However, computing the likelihood of a sequence is instead very efficient, which makes it a suitable method for real-time classification [31].

4 Classification Features

Let $r_0 \rightarrow r_1 \rightarrow \cdots \rightarrow r_n$ be an inclusion sequence as described above. Feature extraction begins by converting the inclusion sequence into sequences of feature vectors. After analyzing the inclusion trees of several thousand benign and malicious websites for a period of 11 months, we identified 12 feature types from three categories. For each feature type, we compute two different features: individual and relative features. An individual feature value is only dependent on the current resource, but a relative feature value is dependent on the current resource and its preceding (or parent) resources. Consequently, we have 24 features for each resource in an inclusion sequence. Individual features can have categorical or continuous values. All continuous feature values are normalized on $[0, 1]$ and their values are discretized. In the case of continuous individual features, the relative feature values are computed by comparing the individual value of the resource to its parent's individual value. The result of the comparison is less, equal, or more. We use the value none for the root resource. To capture the high-level relationships between different inclusions, we only consider the host part of the URL for feature calculation.

4.1 DNS-based Features

The first feature category that we consider is based on DNS properties of the resource host.

Top-Level Domain. For this feature, we measure the types of TLDs from which a resource is included and how it changes along the inclusion sequence. For

Value	Example
`none`	IPs, Extensions
`gen`	*.com, *.org
`gen-subdomain`	*.us.com
`cc`	*.us, *.de, *.cn
`cc-subdomain`	*.co.uk, *.com.cn
`cc-int`	*.xn--p1ai (ru)
`other`	*.biz, *.info

Table 1: Individual TLD values

Value	Example
`none`	root resource
`{got,lost}-tld`	Ext. → *.de, *.us → IP
`gen-to-{cc,other}`	*.org → {*.de, *.info}
`cc-to-{gen,other}`	*.uk → {*.com, *.biz}
`other-to-{gen,cc}`	*.info → {*.net, *.uk}
`same-{gen,cc,other}`	*.com → *.com
`diff-{gen,cc,other}`	*.info → *.biz

Table 2: Relative TLD values.

Value	Example
`ipv6`	2607:f0d0::::4
`ipv4-private`	192.168.0.1
`ipv4-public`	4.2.2.4
`extension`	Ext. Scripts
`dns-sld`	google.com
`dns-sld-sub`	www.google.com
`dns-non-sld`	abc.dyndns.org
`dns-non-sld-sub`	a.b.dyndns.org

Table 3: Individual type values

Value	Example
`none`	root resource
`same-site`	w.google.com → ad.google.com
`same-sld`	1.dyndns.org → 2.dyndns.org
`same-company`	ad.google.com → www.google.de
`same-eff-tld`	bbc.co.uk → london.co.uk
`same-tld`	bbc.co.uk → london.uk
`different`	google.com → facebook.net

Table 4: Relative type values.

every resource in an inclusion sequence, we assign one of the values in Table 1 as an individual feature. For the relative feature, we consider the changes that occur between the top-level domain of the preceding resource and the resource itself. Table 2 shows 15 different values of the relative TLD feature.

Type. This feature identifies the types of resource hosts and their changes along the inclusion sequence. Possible values of individual and relative features are shown in Table 3 and Table 4 respectively.

Level. A domain name consists of a set of labels separated by dots. We say a domain name with n labels is in level $n-1$. For example, `www.google.com` is in level 2. For IP addresses and extension scripts, we consider their level to be 1. For a given host, we compute the individual feature by dividing the level by a maximum value of 126.

Alexa Ranking. We also consider the ranking of a resource's domain in the Alexa Top 1M websites. To compute the normalized ranking as an individual feature, we divide the ranking of the domain by one million. For IP addresses, extensions, and domains that are not in the top 1M, we use the value `none`.

4.2 String-based Features

We observed that malicious domain names often make liberal use of digits and hyphens in combination with alphabetical characters. So, in this feature category, we characterize the string properties of resource hosts. For IP addresses and extension scripts, we assign the value 1 for individual features.

Non-Alphabetic Characters. For this feature, we compute the individual feature value by dividing the number of non-alphabetical characters over the length of domain.

Unique Characters. We also measure the number of unique characters that are used in a domain. The individual feature is the number of unique characters in the domain divided by the maximum number of unique characters in the domain name, which is 38 (26 alphabetics, 10 digits, hyphen, and dot).

Character Frequency. For this feature, we simply measure how often a single character is seen in a domain. To compute an individual feature value, we calculate the frequency of each character in the domain and then divide the average of these frequencies by the length of the domain to normalize the value.

Length. In this feature, we measure the length of the domain divided by the maximum length of a domain, which is 253.

Entropy. In practice, benign domains are typically intended to be memorable to users. This is often not a concern for attackers, as evidenced by the use of domain generation algorithms [8]. Consequently, we employ Shannon entropy to measure the randomness of domains in the inclusion sequence. We calculate normalized entropy as the absolute Shannon entropy divided by the maximum entropy for the domain name.

4.3 Role-based Features

We observed that identifying the role of resources in the inclusion sequences can be helpful in detecting malicious resources. For example, recent work [29] reveals that attackers misuse ad networks as well as URL shortening services for malicious intent. So far, we consider three roles for a resource: *i)* ad-network, *ii)* content delivery network (CDN), and *iii)* URL shortening service.

In total, we have three features in this category, as each host can simultaneously perform multiple roles. Both individual and relative features in this category have binary values. For the individual feature, the value is Yes if the host has the role, and No otherwise. For the relative feature, we assign a value Yes if at least one of the preceding hosts have the corresponding role, and No otherwise. For extension scripts, we assign the value No for all of the features. To assign the roles, we compiled a list of common hosts related to these roles that contains 5,767 ad-networks, 48 CDNs, and 461 URL shortening services.

5 Implementation

In this section, we discuss our prototype implementation of Excision for detecting and blocking malicious third-party content inclusions. We implemented Excision as a set of modifications to the Chromium browser[1]. In order to implement our system, we needed to modify Blink and the Chromium extension engine to enable Excision to detect and block inclusions of malicious content in an online and automatic fashion while the web page is loading. The entire set of modifications consists of less than 1,000 lines of C++ and several lines of JavaScript.

[1] While our implementation could be adopted as-is by any browser vendors that use WebKit-derived engines, the design presented here is highly likely to be portable to other browsers.

5.1 Enhancements to Blink

Blink is primarily responsible for parsing HTML documents, managing script execution, and fetching resources from the network. Consequently, it is ideally suited for constructing the inclusion tree for a web page, as well as blocking the inclusion of malicious content.

Tracking Resource Inclusion. Static resource inclusions that are hard-coded by publishers inside the page's HTML are added to the inclusion tree as the direct children of the root node. For dynamic inclusions (e.g., via the createElement() and write() DOM API functions), the system must find the script resource responsible for the resource inclusion. To monitor dynamic resource inclusions, the system tracks the start and termination of script execution. Any resources that are included in this interval will be considered as the children of that script resource in the inclusion tree.

Handling Events and Timers. Events and timers are widely used by web developers to respond to user interactions (e.g., clicking on an element) or schedule execution of code after some time has elapsed. To capture the creation and firing of events and timers, the system tracks the registration of callback functions for the corresponding APIs.

5.2 Enhancements to the Chromium Extension Engine

The Chromium extension engine handles the loading, management, and execution of extensions. To access the page's DOM, the extension injects and executes *content scripts* in the page's context which are regular JavaScript programs.

Tracking Content Scripts Injection and Execution. Content scripts are usually injected into web pages either via the extension's manifest file using the content_scripts field or at runtime via the executeScript API. Either way, content scripts are considered direct children of the root node in the inclusion tree. Therefore, in order to track the inclusion of resources as a result of content script execution, the extension engine was modified to track the injection and execution of content scripts.

Handling Callback Functions. Like any other JavaScript program, content scripts rely heavily on callback functions. For instance, onMessage and sendMessage are used by content scripts to exchange messages with their background pages. To track the execution of callback functions, two JavaScript files were modified in the extension engine which are responsible for invocation and management of callback functions.

6 Evaluation

In this section, we evaluate the security benefits, performance, and usability of the Excision prototype. We describe the data sets we used to train and evaluate the system, and then present the results of the experiments.

Item	Website Crawl	Extension Crawl
Websites Crawled	234,529	20
Unavailable Websites	7,412	0
Unique Inclusion Trees	47,789,268	35,004
Unique Inclusion Sequences	27,261,945	61,489
Unique URLs	546,649,590	72,064
Unique Hosts	1,368,021	1,144
Unique Sites	459,615	749
Unique SLDs	419,119	723
Unique Companies	384,820	719
Unique Effective TLDs	1,115	21
Unique TLDs	404	21
Unique IPs	9,755	3

Table 5: Summary of crawling statistics.

Dataset	No. of Inclusion Sequences		No. of Terminal hosts	
	Web. Crawl	Ext. Crawl	Web. Crawl	Ext. Crawl
Benign	3,706,451	7,372	35,044	250
Malicious	25,153	19	1,226	2

Table 6: Data sets used in the evaluation.

6.1 Data Collection

To collect inclusion sequences, we performed two separate crawls for websites and extensions. The summary of crawling statistics are presented in Table 5.

Website Crawl. We built a crawler based on an instrumented version of PhantomJS [3], a scriptable open source browser based on WebKit, and crawled the home pages of the Alexa Top 200K. We performed our data collection from June 20th, 2014 to May 11th, 2015. The crawl was parallelized by deploying 50 crawler instances on five virtual machines, each of which crawled a fixed subset of the Alexa Top 200K websites. To ensure that visited websites did not store any data on the clients, the crawler ran a fresh instance of PhantomJS for each visit. Once all crawlers finished crawling the list of websites, the process was restarted from the beginning. To thwart cloaking techniques [18] utilized by attackers, the crawlers presented a user agent for IE 6.0 on Windows and employed Tor to send HTTP requests from different source IP addresses. We also address JavaScript-based browser fingerprinting by modifying the internal implementation of the `navigator` object to return a fake value for the `appCodeName`, `appName`, `appVersion`, `platform`, `product`, `userAgent`, and `vendor` attributes.

Extension Crawl. To collect inclusion sequences related to extensions, we used 292 Chrome extensions reported in prior work [42] that injected ads into web pages. Since ad-injecting extensions mostly target shopping websites (e.g., Amazon), we chose the Alexa Top 20 shopping websites for crawling to trigger ad injection by those 292 extensions. We built a crawler by instrumenting Chromium 43 and collected data for a period of one week from June 16th to June 22nd, 2015. The system loaded every extension and then visited the home

pages of the Alexa Top 20 shopping websites using Selenium WebDriver [4]. This process was repeated after crawling the entire set of extensions. In addition, our crawler triggered all the events and timers registered by content scripts.

6.2 Building Labeled Datasets

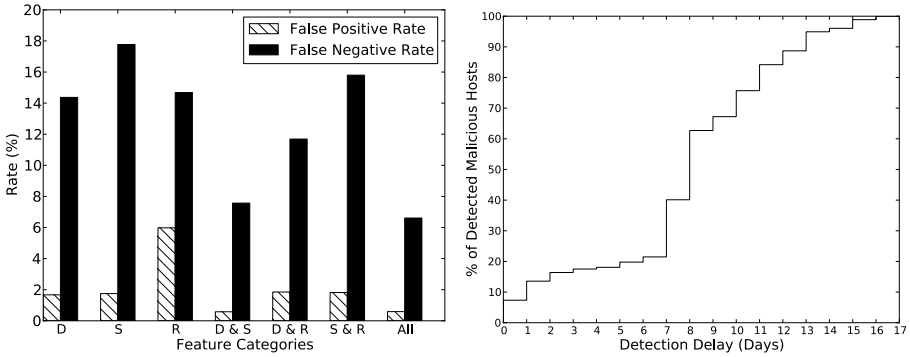
To classify a given inclusion sequence as benign or malicious, we trained two hidden Markov models for benign and malicious inclusion sequences from our data set. We labeled collected inclusion sequences as either benign or malicious using VirusTotal [5]. VirusTotal's URL scanning service aggregates reports of malicious URLs from most prominent URL scanners such as Google Safe Browsing [13] and the Malware Domain List. The malicious data set contains all inclusion sequences where the last included resource's host is reported malicious by at least three out of the 62 URL scanners in VirusTotal. On the other hand, the benign data set only contains inclusion sequences that do not contain any host in the entire sequence that is reported as malicious by any URL scanner in VirusTotal. To build benign data set, we considered reputable domains such as well-known search engines and advertising networks as benign regardless of whether they are reported as malicious by any URL scanner in VirusTotal. Table 6 summarizes the data sets. The unique number of inclusion sequences and terminal hosts are shown separately for the website and extension data sets. The terminal hosts column is the number of unique hosts that terminate inclusion sequences.

6.3 Detection Results

To evaluate the accuracy of our classifier, we used 10-fold cross-validation, in which we first partitioned each data set into 10 equal-sized folds, trained the models on nine folds, and then validated the resulting models with the remaining fold. The process was repeated for each fold and, at the end, we calculated the average false positive rate and false negative rate. When splitting the data set into training and testing sets, we made sure that inclusion sequences with different lengths were present in both. We also ensured that both sets contained extension-related inclusion sequences.

The results show that our classifier achieved a false positive rate of 0.59% and false negative rate of 6.61% (detection rate of 93.39%). Most of the false positives are due to inclusion sequences that do not appear too often in the training sets. Hence, users are unlikely to experience many false positives in a real browsing environment (as will be shown in our usability analysis in Section 6.6).

To quantify the contribution of different feature categories to the classification, we trained classifiers using different combinations of feature categories and compared the results. Figure 3a shows the false positive rate and false negative rate of every combination with a 10-fold cross-validation training scheme. According to Figure 3a, the best false positive and false negative rates were obtained using a combination of all feature categories.



(a) Effectiveness of features for classification (D = DNS, S = String, R = Role).

(b) Early detection results.

Fig. 3: Feature category contributions and early detection results.

6.4 Comparison with URL Scanners

To evaluate the ability of our system in detecting unreported suspicious hosts, we ran our classifier on inclusion sequences collected from June 1st until July 14th, 2015. We compared our detection results with reports from URL scanners in VirusTotal and detected 89 new suspicious hosts. We believe that these hosts are in fact dedicated malicious hosts that play the role of redirectors and manage malicious traffic flows as described in prior work [21]. These hosts did not deliver malicious resources themselves, but they consistently included resources from other hosts that were flagged as malicious by URL scanners. Out of 89 suspicious domains, nearly 44% were recently registered in 2015, and more than 23% no longer resolve to an IP address.

Furthermore, we detected 177 hosts that were later reported by URL scanners after some delay. Figure 3b shows the early detection results of our system. A significant number of these hosts were not reported until some time had passed after Excision initially identified them. For instance, nearly 78% of the malicious hosts were not reported by any URL scanner during the first week.

6.5 Performance

To assess the performance of Excision, we used Selenium WebDriver to automatically visit the Alexa Top 1K with both original and modified Chromium browsers. In order to measure our prototype performance with a realistic set of extensions, we installed five of the most popular extensions in the Chrome Web Store: Adblock Plus, Google Translate, Google Dictionary, Evernote Web Clipper, and Tampermonkey.

For each browser, we visited the home pages of the entire list of websites and recorded the total elapsed time. Due to the dynamic nature of ads and their influence on page load time, we repeated the experiment 10 times and measured

the average elapsed time. On average, the elapsed times were 3,065 and 3,438 seconds for the original and modified browsers respectively. Therefore, EXCISION incurred a 12.2% overhead on browsing time on average, which corresponds to a noticeable overhead that is nevertheless acceptable for many users (see Section 6.6). To measure the overhead incurred by EXCISION on browser startup time, we launched the modified browser 10 times and measured the average browser launch time. EXCISION caused a 3.2 seconds delay on browser startup time, which is ameliorated by the fact that this is a one-time performance hit.

6.6 Usability

We conducted an experiment to evaluate the impact of EXCISION on the user's browsing experience. We conducted the study on 10 students that self-reported as expert Internet users. We provided each participant with a list of 50 websites that were selected randomly from the Alexa Top 500 and then asked them to visit at least three levels down in each website. Participants were asked to report the number of visited pages and the list of domains reported as malicious by our system. In addition, participants were asked to record the number of errors they encountered while they browsed the websites. Errors were considered to occur when the browser crashed, the appearance of a web page was corrupted, or page load times were abnormally long. Furthermore, in order to ensure that benign extensions were not prevented from executing as expected in the presence of our system, the browser was configured to load the five popular extensions listed in Section 6.5 and participants were asked to report any problem while using the extensions.

The results of the study show that out of 5,129 web pages visited by the participants, only 83 errors were encountered and the majority of web pages loaded correctly. Most of these errors happened due to relatively high load times. In addition, none of the participants reported any broken extensions. Furthermore, 31 malicious inclusions were reported by our tool that were automatically processed (without manual examination, for privacy reasons) using VirusTotal. Based on the results, we believe that our proof-of-concept prototype is compatible with frequently used websites and extensions, and can be improved through further engineering to work completely free of errors.

Ethics. In designing the usability experiment, we made a conscious effort to avoid collecting personal or sensitive information. In particular, we restricted the kinds of information we asked users to report to incidence counts for each of the categories of information, except for malicious URLs that were reported by our tool. Malicious URLs were automatically submitted to VirusTotal to obtain a malice classification before being discarded, and were not viewed by us or manually inspected. In addition, the participants were asked to avoid browsing websites requiring a login or involving sensitive subject matter.

7 Related Work

Third-party Content Isolation. Several recent research projects [14, 37, 39] attempted to improve the security of browsers by isolating browser components in order to minimize data sharing among software components. The main issue with these approaches is that they do not perform any isolation between JavaScript loaded from different domains and web applications, letting untrusted scripts access the main web application's code and data. Efforts such as AdJail [23] attempt to protect privacy by isolating ads into an iframe-based sandbox. However, this approach restricts contextual targeting advertisement in which ad scripts need to have access to host page content.

Detecting Malicious Domains. There are multiple approaches to automatically detecting malicious web domains. Madtracer [22] has been proposed to automatically capture malvertising cases. But, this system is not as precise as our approach in identifying the causal relationships among different domains. EXPOSURE [8] employs passive DNS analysis techniques to detect malicious domains. SpiderWeb [36] is also a system that is able to detect malicious web pages by crowd-sourcing redirection chains. Segugio [32] tracks new malware-control domain names in very large ISP networks. WebWitness [27] automatically traces back malware download paths to understand attack trends. While these techniques can be used to automatically detect malicious websites and update blacklists, they are not online systems and may not be effectively used to detect malicious third-party inclusions since users expect a certain level of performance while browsing the Web.

Another effective detection approach is to produce blacklists of malicious sites by scanning the Internet that can be efficiently checked by the browser (e.g., Google Safe Browsing [13]). Blacklist construction requires extensive infrastructure to continuously scan the Internet and bypass cloaking and general malware evasion attempts in order to reliably identify malware distribution sites, phishing pages, and other Web malice. As our evaluation in Section 6 demonstrates, these blacklists sometimes lag the introduction of malicious sites on the Internet, or fail to find these malicious sites. However, they are nevertheless effective, and we view the approach we propose as a complementary technique to established blacklist generation and enforcement techniques.

Policy Enforcement. Another approach is to search and restrict third-party code included in web applications [12, 15, 24]. For example, ADsafe [1] removes dangerous JavaScript features (e.g., `eval`), enforcing a whitelist of allowed JavaScript functionality considered safe. It is also possible to protect against malicious JavaScript ads by enforcing policies at runtime [30, 33]. For example, Meyerovich et al. [26] introduce a client-side framework that allows web applications to enforce fine-grained security policies for DOM elements. AdSentry [11] provides a shadow JavaScript engine that runs untrusted ad scripts in a sandboxed environment.

8 Conclusion

In this paper, we presented EXCISION, an in-browser system to automatically detect and block malicious third-party content inclusions before they can attack the user's browser. Our system is complementary to other defensive approaches such as CSP and Google Safe Browsing, and is implemented as a set of modifications to the Chromium browser. EXCISION does not perform any blacklisting to detect malicious third-party inclusions. Rather, it incrementally constructs an inclusion tree for a given web page and automatically prevents loading malicious resources by classifying their inclusion sequences using a set of pre-built models.

Our evaluation over an 11 month crawl of the Alexa Top 200K demonstrates that the prototype implementation of EXCISION detects a significant number of malicious third-party content in the wild. In particular, the system achieved a 93.39% detection rate with a false positive rate of 0.59%. EXCISION was also able to detect previously unknown malicious inclusions. We also evaluated the performance and usability of EXCISION when browsing popular websites, and show that the approach is capable of improving the security of users on the Web by detecting 31 malicious inclusions during a user study without significantly degrading the user experience.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1409738.

References

1. ADsafe. `http://www.adsafe.org/`.
2. CSP in Content Scripts. `https://developer.chrome.com/extensions/contentSecurityPolicy#interactions`.
3. PhantomJS. `http://phantomjs.org/`.
4. Selenium: Web Browser Automation. `http://www.seleniumhq.org/`.
5. VirtusTotal. `https://www.virustotal.com/`.
6. Cross-Origin Resource Sharing (CORS). `http://www.w3.org/TR/cors/`, 2014.
7. Content Security Policy 1.1. `https://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html`, 2015.
8. Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *Network and Distributed System Security Symposium (NDSS)*, 2011.
9. Devin Coldewey. Marriott puts an end to shady ad injection service. `http://techcrunch.com/2012/04/09/marriott-puts-an-end-to-shady-ad-injection-service/`, 2012.
10. Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *International World Wide Web Conference (WWW)*, 2010.

11. Xinshu Dong, Minh Tran, Zhenkai Liang, and Xuxian Jiang. AdSentry: Comprehensive and flexible confinement of JavaScript-based advertisements. In *Annual Computer Security Applications Conference (ACSAC)*, 2011.
12. Matthew Finifter, Joel Weinberger, and Adam Barth. Preventing capability leaks in secure JavaScript subsets. In *Network and Distributed System Security Symposium (NDSS)*, 2010.
13. Google, Inc. Google Safe Browsing API. `https://developers.google.com/safe-browsing/`, 2015.
14. Chris Grier, Shuo Tang, and Samuel T. King. Secure web browsing with the OP web browser. In *IEEE Symposium on Security and Privacy (Oakland)*, 2008.
15. Salvatore Guarnieri and Benjamin Livshits. GATEKEEPER: Mostly static enforcement of security and reliability policies for JavaScript code. In *USENIX Security Symposium*, 2009.
16. Lin-Shung Huang, Zack Weinberg, Chris Evans, and Collin Jackson. Protecting browsers from cross-origin CSS attacks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2010.
17. Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. Trends and lessons from three years fighting malicious extensions. In *USENIX Security Symposium*, 2015.
18. John P. John, Fang Yu, Yinglian Xie, Arvind Krishnamurthy, and Martin Abadi. deSEO: Combating search-result poisoning. In *USENIX Security Symposium*, 2011.
19. Alexandros Kapravelos, Chris Grier, Neha Chachra, Chris Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *USENIX Security Symposium*, 2014.
20. David D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *European Conference on Machine Learning (ECML)*, 1998.
21. Zhou Li, Sumayah Alrwais, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures. In *IEEE Symposium on Security and Privacy (Oakland)*, 2013.
22. Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing your enemy: Understanding and detecting malicious web advertising. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.
23. Mike Ter Louw, Karthik Thotta Ganesh, and V.N. Venkatakrishnan. AdJail: Practical enforcement of confidentiality and integrity policies on web advertisements. In *USENIX Security Symposium*, 2010.
24. Sergio Maffeis and Ankur Taly. Language-based isolation of untrusted JavaScript. In *IEEE Computer Security Foundations Symposium (CSF)*, 2009.
25. Ginny Marvin. Google study exposes "tangled web" of companies profiting from ad injection. `http://marketingland.com/ad-injector-study-google-127738`, 2015.
26. Leo A. Meyerovich and Benjamin Livshits. ConScript: Specifying and enforcing fine-grained security policies for JavaScript in the browser. In *IEEE Symposium on Security and Privacy (Oakland)*, 2010.
27. Terry Nelms, Roberto Perdisci, Manos Antonakakis, and Mustaque Ahamad. WebWitness: Investigating, categorizing, and mitigating malware download paths. In *USENIX Security Symposium*, 2015.
28. Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, , and Giovanni Vigna. You are what you include: Large-scale evaluation of remote JavaScript inclusions. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

29. Nick Nikiforakis, Federico Maggi, Gianluca Stringhini, M Rafique, Wouter Joosen, Christopher Kruegel, Frank Piessens, Giovanni Vigna, and Stefano Zanero. Stranger danger: Exploring the ecosystem of ad-based URL shortening services. In *International World Wide Web Conference (WWW)*, 2014.
30. Phu H. Phung, David Sands, and Andrey Chudnov. Lightweight self-protecting JavaScript. In *ACM Symposium on Information, Computer, and Communications Security (ASIACCS)*, 2009.
31. Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.
32. Babak Rahbarinia, Roberto Perdisci, and Manos Antonakakis. Segugio: Efficient behavior-based tracking of new malware-control domains in large isp networks. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2015.
33. Charles Reis, John Dunagan, Helen J. Wang, Opher Dubrovsky, and Saher Esmeir. BrowserShield: Vulnerability-driven filtering of dynamic HTML. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.
34. Sooel Son and Vitaly Shmatikov. The postman always rings twice: Attacking and defending postMessage in HTML5 websites. In *Network and Distributed System Security Symposium (NDSS)*, 2013.
35. Brett Stone-Gross, Ryan Stevens, Richard Kemmerer, Christopher Kruegel, Giovanni Vigna, and Apostolis Zarras. Understanding fraudulent activities in online ad exchanges. In *Internet Measurement Conference (IMC)*, 2011.
36. Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.
37. Shuo Tang, Haohui Mai, and Samuel T. King. Trust and protection in the Illinois browser operating system. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.
38. 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 *IEEE Symposium on Security and Privacy (Oakland)*, 2015.
39. Helen J. Wang, Chris Grier, Alexander Moshchuk, Samuel T. King, Piali Choudhury, and Herman Venter. The multi-principal OS construction of the Gazelle web browser. In *USENIX Security Symposium*, 2009.
40. Michael Weissbacher, Tobias Lauinger, and William Robertson. Why is CSP failing? trends and challenges in CSP adoption. In *International Conference on Recent Advances in Intrusion Detection (RAID)*, 2014.
41. World Wide Web Consortium (W3C). What is the document object model? `http://www.w3.org/TR/DOM-Level-2-Core/introduction.html`.
42. Xinyu Xing, Wei Meng, Udi Weinsberg, Anmol Sheth, Byoungyoung Lee, Roberto Perdisci, and Wenke Lee. Unraveling the relationship between ad-injecting browser extensions and malvertising. In *International World Wide Web Conference (WWW)*, 2015.
43. Apostolis Zarras, Alexandros Kapravelos, Gianluca Stringhini, Thorsten Holz, Christopher Kruegel, and Giovanni Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *Proceedings of the Internet Measurement Conference (IMC)*, 2014.