A Large-scale Analysis of Content Modification by Open HTTP Proxies

Giorgos Tsirantonakis,[*] Panagiotis Ilia,[*] Sotiris Ioannidis,[*] Elias Athanasopoulos,[†] Michalis Polychronakis[‡]

[*] FORTH, Greece
{tsirant, pilia, sotiris}@ics.forth.gr

[†] University of Cyprus, Cyprus
eliasathan@cs.ucy.ac.cy

[‡] Stony Brook University, USA
mikepo@stonybrook.edu

***Abstract*—Open HTTP proxies offer a quick and convenient solution for routing web traffic towards a destination. In contrast to more elaborate relaying systems, such as anonymity networks or VPN services, users can freely connect to an open HTTP proxy without the need to install any special software. Therefore, open HTTP proxies are an attractive option for bypassing IP-based filters and geo-location restrictions, circumventing content blocking and censorship, and in general, hiding the client's IP address when accessing a web server. Nevertheless, the consequences of routing traffic through an untrusted third party can be severe, while the operating incentives of the thousands of publicly available HTTP proxies are questionable.**

In this paper, we present the results of a large-scale analysis of open HTTP proxies, focusing on determining the extent to which user traffic is manipulated while being relayed. We have designed a methodology for detecting proxies that, instead of passively relaying traffic, actively modify the relayed content. Beyond simple detection, our framework is capable of macroscopically attributing certain traffic modifications at the network level to well-defined malicious actions, such as ad injection, user fingerprinting, and redirection to malware landing pages.

We have applied our methodology on a large set of publicly available HTTP proxies, which we monitored for a period of two months, and identified that 38% of them perform some form of content modification. The majority of these proxies can be considered benign, as they do not perform any harmful content modification. However, 5.15% of the tested proxies were found to perform modification or injection that can be considered as malicious or unwanted. Specifically, 47% of the malicious proxies injected ads, 39% injected code for collecting user information that can be used for tracking and fingerprinting, and 12% attempted to redirect the user to pages that contain malware.

Our study reveals the true incentives of many of the publicly available web proxies. Our findings raise several concerns, as we uncover multiple cases where users can be severely affected by connecting to an open proxy. As a step towards protecting users against unwanted content modification, we built a service that leverages our methodology to automatically collect and probe public proxies, and generates a list of safe proxies that do not perform any content modification, on a daily basis.

NDSS '18, 18-21 February 2018, San Diego, CA, USA
Copyright 2018 Internet Society, ISBN 1-1891562-49-5
http://dx.doi.org/10.14722/ndss.2018.23244

I. Introduction

Internet users often place their trust on systems that are not under their control. From a security and privacy perspective, a particularly critical class of such systems is HTTP proxies that act as "stepping stones" between web clients and servers. By relaying their traffic through a proxy, users can access content and services that are otherwise blocked due to geographical restrictions, content filtering policies, or censorship; and to some extent preserve their anonymity, by hiding the originating IP address from the final destination (although still exposing it to the proxy).

Once the user traffic reaches the proxy towards its actual destination, unless end-to-end encryption is used, a rogue or compromised proxy can tamper with the transmitted content or snoop for sensitive user data [27]. However, the problem is not alleviated even when end-to-end encryption is used, as man-in-the-middle attacks are still possible using fake or even valid—obtained through compromised CAs or generated by powerful adversaries—certificates, or SSL-stripping attacks [20]. The potential harm due to network traffic interception attacks can be severe, ranging from mere annoyance and inconvenience to system compromise and theft of private information.

There has been a remarkable effort during the last years by the community and some key organizations to make the web more secure. The emergence of Certificate Authorities that offer free TLS certificates such as Let's Encrypt,[1] and the services launched by cloud providers such as CloudFlare[2] and Amazon[3] that enable certificate management without much hassle, have resulted in increasing adoption of HTTPS. Only recently the encrypted web traffic has surpassed the unencrypted traffic [11], but there is still a long distance to cover.

However, as a large volume of web content is still transmitted unencrypted over HTTP, rogue web proxy operators can monetize their traffic by altering the relayed content to inject ads and affiliate links, prompt users to download spyware and other unwanted software, or mount phishing attacks [16]. Even more deviously, instead of placing additional ads that may annoy users, miscreants can replace *existing* ads in the page with their own ads. This can be as simple as replacing a website's ad network identifier with the attacker's own affiliate identifier, essentially stealing the revenue of the original

[1] https://letsencrypt.org
[2] https://www.cloudflare.com/ssl/
[3] https://aws.amazon.com/certificate-manager/

website (i.e., publisher). A more severe form of network traffic manipulation is the inclusion of malicious JavaScript code for mounting XSS, CSRF, or DDoS attacks [18], the injection of exploits against the browser or other client-side software [28], [33], or the infection of downloaded executables [24], which can all result in full system compromise. In a proof-of-concept implementation, Chema and Fernandez [7] deployed a malicious proxy that modifies the originally requested JavaScript files to dynamically fetch and execute malicious code. This allowed them not only to collect cookies and user sensitive information, but also to take control of the infected hosts and build a *botnet*.

The proliferation and widespread use of web proxies necessitates an approach to detect, understand and measure the extent of content modification by rogue web proxies. The ease of setting up a free online proxy (e.g., on a cloud-hosted virtual machine) and registering it on the numerous "proxy list" websites, raises the question of whether miscreants employ these tactics to attract and gain access to user traffic, and then monetize it or cause further harm. Sporadic evidence so far has shown that this is indeed happening in various types of network relays, including VPN servers and anonymity network relays [1], [3], [6], [16], [24], [35], [36], but the extent of the problem in the front of web proxies remains unknown.

To understand and measure the extent of content modification by rogue HTTP proxies, in this work, we have designed a methodology for detecting and analyzing content alteration and code injection attempts. Specifically, we have built a framework that regularly collects HTTP proxies from several "proxy list" websites, and tests them using a novel technique based on decoy websites (dubbed *honeysites*) under our control. Furthermore, we have implemented a content modification detection approach that operates at the level of a page's DOM tree, which can detect even slight object modifications. To facilitate the analysis of content modification incidents, we have implemented a clustering technique for grouping together cases of content modification that follow similar patterns.

Instead of visiting real websites with and without a proxy and comparing the difference in the retrieved content, the use of honeysites allows us to avoid any false positive issues due to highly dynamic content (e.g., news updates, rotating ads, and time-related information), or content localization (e.g., due to the proxy's different location than the client). However, we decided to also include one real website in our experiments (*http://bbc.com*), even though we employ honeysites of different complexity and content diversity. This allows us to verify that the modifications observed in honeysites also occur in real websites, and to investigate whether any sophisticated proxies can distinguish real websites from honeysites and alter their behavior accordingly to avoid being detected.

In this study, we monitored 15 public "proxy list" websites for a period of two months, and systematically collected all the proxies they offered. This process resulted in 65,871 unique HTTP proxies. By sending multiple probes to each proxy we found that 16,427 never appeared to be alive, and that, interestingly, only 19,473 succeeded in fetching at least one of the requested testing websites. Our results suggest that 5.15% of the tested proxies perform some form of modification that can be clearly considered as malicious. The observed modifications included the injection of extra (or the modification of existing) ads, the inclusion of tracking and fingerprinting libraries, and the collection of data from social networking services on which the user is already authenticated. Besides that, we also discovered more severe and sophisticated instances of malicious behavior, such as SSL stripping and redirection to servers that have been reported to host malware.

By identifying and analyzing multiple cases of content injection and modification, this study provides insights about the behavior of rogue web proxies and reveals many patterns that exist between these modifications. This enabled us to build a web service for assessing web proxies on a daily basis, and making publicly available a list of proxies that did not perform any modification during our tests.

In summary, our work makes the following contributions:

- We present an approach for the detection of unwanted or malicious content modification by rogue HTTP proxies, based on the observation of discrepancies in the retrieval of content through the tested proxies. Our technique uses decoy websites, dubbed *honeysites*, that have been designed with a different degree of complexity and external content dependencies, to allow differentiating between different types of content injection or alteration.
- We have performed a large-scale measurement study of open HTTP proxies retrieved from public "proxy list" websites, and used our technique to assess the extent of malicious content modification by rogue proxies. Our findings suggest that 5.15% of the tested proxies engaged in some form of malicious content modification.
- We provide an in-depth analysis of indicative content modification cases that we observed, which involved the injection of ads, tracking and fingerprinting code, private information collection, and malware distribution.
- We have implemented a service that automatically collects and tests HTTP proxies on a daily basis, for generating and publishing an up-to-date list of proxies that do not perform any content modification or injection. The service is available at *http://proxyscan.ics.forth.gr*.

II. Background and Related Work

A. HTTP Proxies

Web proxies are one of the most widely used types of network relays mainly due to their ease of use, especially for users that simply want to bypass filtering restrictions or access content that is not available in their country. Many web proxy websites allow users to conveniently type in the URL of the page they want to visit, which is then rendered directly, usually within a frame. Legacy HTTP or SOCKS proxies require users to manually configure their browser to explicitly use the proxy by entering the IP address and port of the proxy into the browser's settings window. Many browser extensions facilitate this process, offering the convenience of single-click buttons for toggling a configured proxy on and off.

Both types of web proxies are widely used, as it is evident from the numerous websites dedicated to providing up-to-date lists of working web proxies around the world. Besides the URL or the pair of IP address and port of each proxy, these lists typically provide information about the geographic location of the proxy (important for users who want to access content available only in certain countries), its supported protocols, the last time the proxy was checked for responsiveness, and even statistics about its uptime and access latency.

It is common for a proxy list website to provide information about hundreds or even thousands of recently checked proxies. Considering that there are numerous independent proxy list websites, we estimate the number of publicly accessible web proxies to be in the order of tens of thousands. This conservative estimation is based on the number of proxies listed in just the top ten proxy list websites returned by a Google search for "HTTP proxy list," and is indeed, in line with the numbers reported in our study.

B. Detecting Content Alteration and Fraudulent Ad Traffic

A large body of prior work has focused on security issues related to the ad serving ecosystem. Li et al. [29] shed light to fraudulent activities in online ad exchanges. In the front of click fraud, Dave et al. [9] proposed a methodology for measuring the extent of click-spam, and proactively detecting different simultaneous click-spam attacks. To counter click fraud, Hamed Haddadi [14] proposed the concept of "Bluf ads," which are ads meant to be detected and clicked only by automated crawlers or click-fraud workers.

Malvertising is also an important threat that has received attention, mostly focusing on methods to detect and block malicious ads. Li et al. [19] performed a large-scale study of malicious ads, and built MadTracer, a defense that automatically generates detection rules and uses them to block malicious ads. Ford et al. [12] studied the particular class of malware delivery that leverages malicious Flash advertisements to infect web site visitors, and developed an automate Flash content analysis technique for the detection of malicious behavior.

Once malware has infected a user system, it often uses ad injection to monetize the victim's web browsing activity. In a large-scale study of ad injection on infected machines, Thomas et al. [30] followed a similar approach to ours, by inspecting a page's DOM tree, to identify the injection of malicious content. The actual identification was based on a whitelist of legitimate content, similar in spirit to content secure policy (CSP) used by modern browsers for whitelisting JavaScript code. This approach has the limitation that if the injected code involves domains that are included in the whitelist (e.g., when replacing existing ads with malicious ads from the same ad network), then the injection will go unnoticed. Our content modification detection approach does not rely on a whitelist, and thus can identify any addition or alteration of an element in the DOM tree, even if it involves the same ad network. Their results indicated that 5% of unique IP addresses accessing Google were infected with malware that injected ads. Operation Ghost Click, one of the largest cybercriminal takedown efforts in history, took down an ad fraud infrastructure that affected 4 million users infected by the DNS changer malware, and made its owners 14 million USD over a period of four years [2].

Many efforts have focused on designing solutions to detect and prevent content modification. Vratonjic et al. [32] anticipated the problem of in-line ad replacement or modification through man-in-the-middle attacks, and proposed a collaborative technique to preserve the integrity of ads—our findings highlight the need for such schemes, and the necessity of preserving network traffic integrity in general. Reis et al. [26] proposed "web tripwires," a method that allows publishers to include a verification check in their page that compares the DOM a user received with the DOM the publisher sent. Arshad et al. [4] developed OriginTracer, a tool that notifies the user about which party is responsible for any modifications in a loaded page's content by tracking DOM element alterations (e.g., due to browser extensions). The same authors have developed Excision [5], an in-browser mechanism that relies on an enhanced version of the DOM tree to keep track of the relationships between the resources of a page in order to block malicious third-party content.

Although individual websites can detect to a certain extent modifications to their client-rendered content using the above approaches, these mechanisms are not widely adopted. In contrast, our efforts focus on the detection of content modification by scanning publicly accessible HTTP proxies to uncover various forms of traffic modification.

C. HTTP Proxy Studies

As discussed in the above, recent research has shown that ad injection and hijacking has been impacting tens of millions of users worldwide through malicious browser extensions and malware infections [2], [30]. Therefore, it is natural to expect miscreants to employ similar monetization strategies through the deployment of rogue network relays. Sporadic evidence so far has shown that traffic interception by rogue network relays is indeed happening [1], [3], [6], [16], [24], [35], [36].

O'Neill et al. [23] developed a probing tool, which was deployed through Google AdWords campaigns, and measured the prevalence of TLS proxies. They found that one in 250 TLS connections are TLS-proxied, and identified over 3,600 cases of malware intercepting a TLS communication. Holz et al. [17] developed the Crossbear system to discover TLS man-in-the-middle attacks by tracking IP routes and comparing certificates with distributed nodes they have deployed across the Internet. Carnavalet et al. [10] uncovered security vulnerabilities in TLS proxies used by antivirus and parental control applications, allowing attackers to mount man-in-the-middle attacks. Weaver et al. [34] leveraged the ICSI Netalyzr client base to measure the prevalence of HTTP proxies, and found that 14% of the tested clients made use of a web proxy. Also, they detected and classified several kinds of proxies, and observed cases of content modification due to client-side security software (e.g., antivirus and firewall products) or server-side compression and transcoding. Chung et al. [8] used a paid proxy service whereby users can route their traffic via other users to uncover content manipulation in end-to-end connections.

In contrast to the above studies, our research focuses on publicly available HTTP proxies that users knowingly employ (for various reasons, such as to access otherwise blocked content, seek protection when using untrusted networks, and preserve their anonymity), and which are available via numerous "free proxy list" websites. Furthermore, our methodology is different since we focus on detecting modifications in the DOM tree of the web page served to the user. Besides some limited evidence that open HTTP proxies are indeed involved in suspicious activity [15], [16], our effort is the first to perform a large-scale, systematic analysis of the extent and nature of content modification performed by open HTTP proxies.

III. Methodology

In order to understand the behavior of open HTTP proxies and measure the extent of content modification in the relayed user traffic, we have designed a methodology for systematically collecting web proxies from several proxy list websites and testing them using two *honeysites* under our control. We have also implemented a content modification detection technique that compares the DOM tree of the web page fetched by a proxy to its static template. Furthermore, in order to facilitate analysis and assessment of the behavior of content modifying proxies, we have implemented a two-level clustering technique that groups together similar content modification incidents.

As malicious behavior, in this work, we consider content modifications and injections that (i) enable the proxy provider to have monetary gain (ii) affect user privacy and (iii) can lead to browser/system compromise. More specifically, we consider as malicious the injection of ads, the injection of JavaScript code for cryptocurrency mining, the replacement of existing ads, and cases where the rogue proxy replaces the website's ad network identifier with its own affiliate identifier for stealing the website's revenue. Furthermore, we consider as malicious any attempt by a proxy to collect sensitive user information and information regarding the user's system (i.e., OS, browser) that can be used for user profiling and tracking, to set and read cookies, harvest credentials etc. In addition to the above, we consider SSL-stripping, redirection to fake/phishing websites and the delivery of malware and exploits. It should be noted though that we do not consider as malicious the removal of ads by "privacy preserving" proxies, as this modification does not negatively impact the user but aims to protect user privacy.

A. *Collecting a Set of Proxies*

To gather a representative set of proxies, we first collected a set of reliable proxy list websites, and then collected the proxies listed on each website. We began with a Google search query for "HTTP proxy list" (in April 2017), and collected the first 50 results returned. We followed this approach, as a typical user will most likely use a search engine to find websites offering proxies. Among these 50 websites, there were some that offer their lists only with a paid subscription, which we left out. Also, we identified a few cases of almost identical proxy list websites, which are actually managed by the same entity. For each such case, we decided to only use the one website that offered the most proxies, as the great majority of the other websites' proxies were also included in the considered one. After the above, we ended up with 15 different popular proxy list websites that included a representative set of the public proxies available to users at the time of our experiment.

We managed to automatically crawl 10 of the above sites, while we manually collected the proxies from the other five. The crawler was visiting all the pages of each one of these websites on a daily basis, for a period of two months (mid April – mid June, 2017), collecting all available proxies and updating our dataset with any previously unseen ones. The other five proxy list websites employed various techniques (e.g., user registration and CAPTCHAs) to prevent automatic crawling, and thus we manually exported all the proxies they provided in each of our visit. We performed the manual collection once every 10 days, for the duration of our experiment.

Lastly, in order to draw comparisons between public proxies and subscription-based ones, we purchased a one-month subscription in one such proxy list website, and collected every five days all the proxies it offered. Our efforts, both automated and manual, resulted in 65,871 unique proxies.

B. *Probing and Testing the Proxies*

As mentioned, our crawling module visits a set of proxy list websites and updates the set of proxies to be tested by our framework on a daily basis. As the number of proxies in our dataset increases every day, and as we intend to test them on a daily basis for three different websites, we need a scalable architecture that can support multiple parallel tests.

Through some preliminary experimentation we found that many proxies tend to be slow, mainly due to the high load they have, and that sometimes they appear as non-responsive even though they are alive. Although this behavior is expected to a certain degree, as it can be explained by their public nature, it is important for our system to succeed in testing them. Thus, we decided to attempt multiple times throughout a day to test each proxy, and to set a rather high timeout interval (180 seconds).

However, the multiple attempts to test a proxy, in combination with the high timeout interval, impose a significant overhead on the system, which becomes critical as the number of proxies increases. A first step towards avoiding unnecessary latency overheads, is to identify which proxies are non-responsive for a long time (i.e., not alive) and avoid attempting to test them at that time. To accomplish that, we have designed a module that sends a few TCP probes to each proxy in our dataset on a regular basis, in order to identify which of them are alive and responding to incoming connections.

Our framework has been designed to send such probes to all the proxies in our dataset almost every hour, 22 times per day. However, for simplicity we decided to suspend probing and testing during the crawling phase, and to resume it after the dataset is updated. If a proxy is found to be alive by at least one probe, it is included in the set of proxies to be tested that day. In this way, we avoid testing proxies that continuously do not respond to our probes, but attempt multiple times to test proxies that appear to be alive, but possibly are temporarily unavailable. This approach does not implicate that eventually all these proxies will be found to work properly, but it allows us to identify long-term non responsive proxies and focus our efforts on the ones that appeared to be alive recently.

For testing a proxy, our system uses Selenium to configure and launch an instance of the Firefox browser, and requests the three testing websites in parallel, by opening three different tabs. If the testing websites are fetched and rendered correctly, the framework saves the downloaded web pages for content modification detection, as it is discussed in the following. If a timeout occurs and the requested websites have not been fetched, the proxy is added to a queue in order to be tested again later. In that way we focus our attempts on re-testing proxies that did not succeed in fetching the test websites.

In order to achieve scalability and manage testing a large number of proxies, our framework divides the set of proxies to be tested into multiple distinct subsets and assigns a subset to a different machine. Each machine is responsible to send probes

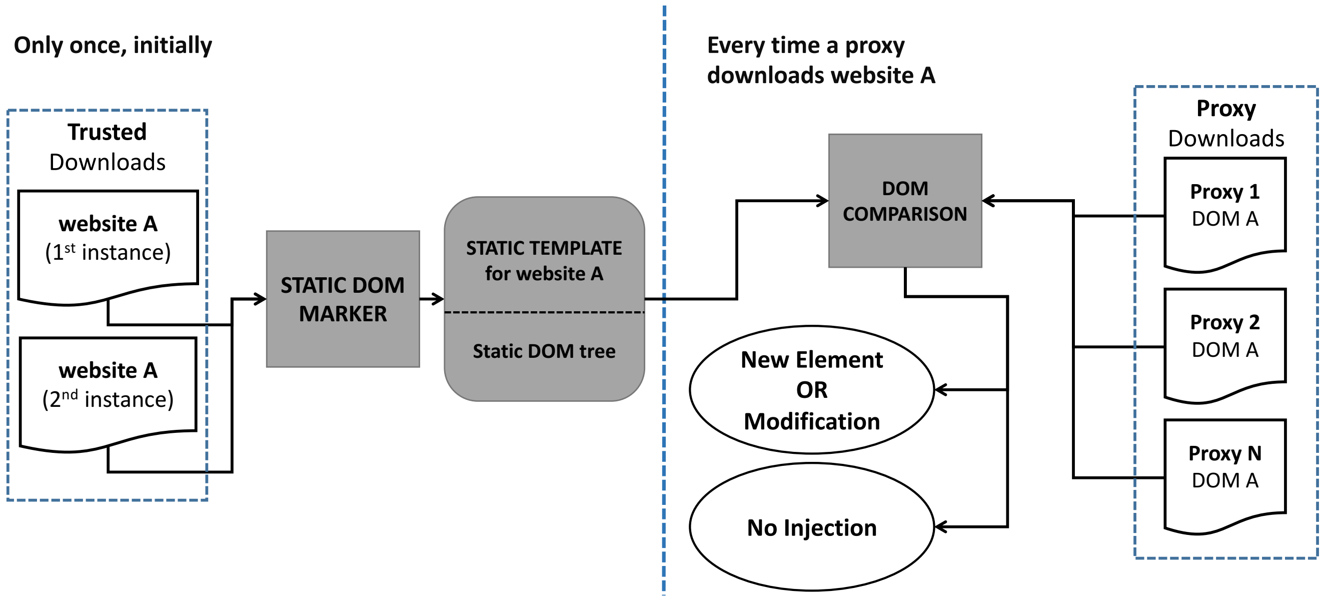

Fig. 1. The methodology followed for detecting content modifications in websites fetched by a proxy. Initially, our framework downloads the target website (or honeysite) multiple times without a proxy, and compares the DOM tree of these downloads, to create a static template of the web page (left). For detecting content modification, it fetches the website through a proxy and compares its DOM tree with the previously generated static template (right).

to the proxies in its subset, for generating its list of proxies to be tested, and starts testing them by launching in parallel up to 60 instances of Firefox. As mentioned previously, each browser instance corresponds to a specific proxy, and requests for retrieving the three testing websites are sent in parallel.

For the purposes of this study, our experimental setup was comprised of two commodity machines. This setup could test up to almost 58 thousand proxies per day, and additional machines could be easily included if needed, as the framework was designed to scale. Since the number of alive proxies in a single day never surpassed 22 thousands in our experiment, as it will be discussed (see Figure 3), we were able to test them multiple times per day, without needing additional resources.

C. Use of Honeysites

Conceptually, a simple approach for detecting content modification by rogue network relays is to visit a given public web page twice, once connecting directly and once through the relay, and comparing the fetched content. In practice, however, this approach is not very effective for two main reasons. First, parts of the page may change due to inherently dynamic content, such as news updates, time-related information, rotating ads, and various forms of content personalization. Second, the geographic distance between the client and the proxy may also result in differences due to content localization, as many websites switch to pages of different language (and possibly region-specific content) based on the visitor's origin. On the other hand, selecting only simple static pages for probing, may miss rogue proxies that focus only on specific types of content that can be monetized, i.e., ads that already exist in the page.

The complexity of having to distinguish between legitimate content modification due to the inherent dynamic behavior of a fetched web page, and any kind of content modification performed by the web proxy itself, makes this simple approach prone to false positives. For this reason, we decided not to rely on visiting real web pages, but to primarily focus on an alternative approach that relies on decoy websites under our control. Specifically, we deploy two *honeysites* of different levels of complexity and dependence on (controllable) third-party content (e.g., ads).

The reason for using two honeysites of different complexity and content, is that we do not only want to detect content modification events, but also to examine whether the behavior of the tested proxies changes according to the type of content relayed through them (e.g., detect whether a proxy modifies only JavaScript files). The main difference between our honeysites lies in the degree of dynamic content generated by the server, and on the third party resources they use. The first honeysite (h_1) consists of a simple, completely static web page. The second one (h_2) is a website created in WordPress,[4] which contains some JavaScript elements and resources, an *iframe* that loads some external content, and three *fake ads*.

The *fake* ads included in h_2 were created based on actual source code provided by three popular ad networks: Google AdSense,[5] Media.net[6] and BuySellAds.[7] We did not actually register for displaying ads from these networks, but just used the example ad inclusion code snippets they provide and set the publisher's ID to an invalid, non-existent value. This ensures that the included ads will not be rendered, as the requests for fetching them will typically fail. What is important, however, is that these fake ads look (from the perspective of the proxy) identical and indistinguishable from legitimate ones. Having a honeysite with embedded ads (h_2) allows us to differentiate between proxies that perform unconditional con-

[4] https://wordpress.com
[5] https://www.google.com/adsense
[6] http://www.media.net
[7] https://www.buysellads.com

tent modification, from proxies that inject content depending on the presence of ad networks in the fetched web page (e.g., replacing the existing legitimate publisher's ID with the attacker's registered ID, injecting additional ads, or completely replacing the honeysite's fake ads).

For testing a given proxy, we automatically browse to the honeysites through the proxy and compare the retrieved content with the original content that was served by our web server. A benefit of this approach is that, by having control of both ends (i.e., client and server), we can precisely identify whether any part of the page was modified, or whether new additional content was injected. However, we cannot preclude the possibility of malicious proxies that do not inject or modify content on every web page they fetch, but somehow determine which downloads to alter, in order to avoid being detected.

To prevent any potentially sophisticated malicious proxies from suspecting the true purpose of our honeysites, and thus avoid exhibiting any inappropriate behavior, as an initial measure we decided to use Amazon's S3 cloud service for hosting the honeysites, instead of using any other infrastructure (e.g., servers hosted in our academic institution's network). Towards that direction, we also decided to visit a popular real web page, alongside with the honeysites we test regularly, in order to detect and further investigate the behavior of any such proxies.

D. Detecting Content Modification

Detecting content modification by a proxy for the static honeysite h_1 is straightforward, given that its "ground truth" content never changes. The dynamic content of h_2 however, makes the detection of content modifications more challenging, as we have to distinguish between changes due to the dynamic content itself and changes due to potential content injection or modification by the tested proxy. To achieve that, we initially download the dynamic honeysite multiple times from a trusted computer (without using a proxy) and compare its content in order to generate a static *template*. Specifically, our approach constructs and traverses the DOM tree of the trusted downloads, and compares the DOM elements to identify which elements remain the same between different downloads. By comparing the DOM elements and their values, we are able to identify and *mark* the static and dynamic elements of the honeysite, and therefore to create a static version of the honeysite that can be used as a *template*.

After creating the static template of each honeysite, we use an approach similar to the process followed for generating the templates themselves, in order to compare the DOM tree of the websites fetched by proxy to the corresponding template. An overview of our methodology is presented in Figure 1. This approach can effectively identify content modification in most cases, as the malicious proxies typically inject new rogue elements or try to modify existing ones. Our system is able to detect changes in the DOM tree when new rogue elements are inserted, and when the value of the elements that are marked as static in the template changes. A limitation, however, of this approach is that it cannot detect changes in the value of dynamic elements, as these values legitimately change between different downloads.

We can overcome this limitation by carefully selecting the types of dynamic content to be included in our honeysite, so that it is generated only in a predictable way. It is noted that our approach takes into consideration the position of each dynamic element during the generation of the template, and thus, during the comparison between the proxy fetched web page with its template, our system expects to identify specific dynamic elements at particular positions in the page's DOM tree. Furthermore, by specifically including dynamic content that is generated in a predictable way, we are able to identify any non-anticipated changes in this content. That is, by carefully choosing dynamic content that is fetched from a particular source, we expect to observe dynamic content from that source.The only case of modifications that can go unnoticed by our system is when a malicious proxy alters a dynamic element in the expected way (e.g., replacing an existing ad with a different ad, from the exact ad network).

E. Clustering Content Modification Incidents

After some initial experimentation using the methodology described previously, we observed a very large number of proxies that modify the content of the honeysites they fetch, either by injecting new elements or by modifying existing ones. In particular, our methodology revealed DOM tree modifications in thousands of cases from both honeysites. To facilitate the analysis of all the content modification incidents detected, we designed a two-level *content modification clustering* approach for identifying injections that follow similar patterns and grouping them together.

At first, we traverse the DOM tree of each download and we identify the position and type of all the injected elements. Our initial clustering approach categorizes these downloads according to the difference between the expected DOM element (at a specific position of the DOM tree according to the template), and the occurring element. As an example, all the cases where an *iframe* element is expected at a particular position of the template's DOM tree, but a script element occurs at that particular position in the DOM tree of the downloaded honeysite, end up in the same cluster.

After that, for implementing a more fine-grained clustering we compare the DOM tree of every instance in each group (i.e., first-level cluster) with all the other modification instances in that particular group. If the DOM tree of two such instances is exactly the same, which means that the same elements appear in the same order and have the exact same content, we add them in the same second-level cluster. It should be noted that during this process we do not take into account the dynamic elements of the DOM tree, which legitimately change in the trusted downloads, but as discussed previously, we use of form of *whitelisting* to detect changes in this elements. Our two-level clustering approach minimizes the effort needed for manually analyzing the observed content modification incidents, as we only need to manually inspect a few downloads from each cluster to understand that particular proxy behavior.

The manual effort can be further reduced by comparing a sequence of DOM tree elements between the different clusters. By keeping track of the DOM tree elements and their position (and their order) in the template of the honeysite, and comparing them with the sequences of elements in every other cluster, it becomes easy to identify proxies that do not inject any new rogue elements, but instead, modify the testing websites by removing content (e.g., remove existing ads).

TABLE I. PROXIES COLLECTED AND TESTED IN OUR EXPERIMENT

Tested Proxies			Content Modifying Proxies	
Total	Alive	Working	Total	Malicious
65,871	49,444	19,473	**7,441 (38.21%)**	**1,004 (5.15%)**

Additionally to the manual inspection of representative content modification cases from each cluster, we employed a simple form of dynamic analysis to gain better understanding of the functionalities performed by the injected code. To that end, we configured the Firefox browser with the Firebug plugin, and rendered a few downloads from each cluster, while monitoring all the outgoing requests and fetched resources. By executing the injected code in a controlled environment and inspecting all the HTTP requests, we were able to detect and analyze additional JavaScript files that are fetched dynamically, and to understand the functionalities of obfuscated code.

Our analysis revealed cases of malicious proxies that set tracking pixels and cookies, inject ads, and perform browser fingerprinting, among others. Also, we were able to determine the nature of the ads shown to the user, and whether these ads are inappropriate or intrusive (e.g., scareware). In that way, we ensure that all the proxies we characterized as malicious, indeed performed content injection or modification that can significantly affect, or even worse, threaten the user. A detailed analysis of our findings is presented in Section V.

IV. ANALYSIS OF PROXY CHARACTERISTICS

As mentioned, we collected a set of proxies by crawling 10 popular proxy list websites on a daily basis, in an automated way, and by systematically visiting five more proxy list websites, and a subscription-based one, and manually exporting all the proxies they offered. This process resulted in the collection of 65,871 unique HTTP proxies in total.

According to our methodology, once the crawling phase is completed, our system starts sending a few TCP probes every hour to each proxy in the updated dataset, for determining which of them are alive and accept incoming connections. We do follow this probing technique in order to identify (temporarily) non responding proxies and thus avoid attempting to test them. When a previously non-responding proxy responds to our probes, it is immediately included in the set of proxies to be tested on that day. While the number of proxies that appeared to be alive each day of the experiment was not high, we identified that in total 49,444 proxies responded to some of our probes during the period of our experiment (see Table I).

Interestingly, from the 49,444 proxies that appeared at some point to be alive, after responding to some of the probes, only 19,473 (38.38%) succeeded in fetching the requested testing websites when tested (we refer to them as "properly working" proxies). As discussed in the following, many of the proxies we characterize as "non-working" provided information that indicate their type of failure. We observed that most of these proxies failed to fetch the testing websites due to network and DNS errors. On the contrary, a large fraction of the non-working proxies never fetched any of the requested websites or provided any information about their type of failure, or in general, any information indicating that they are still alive,

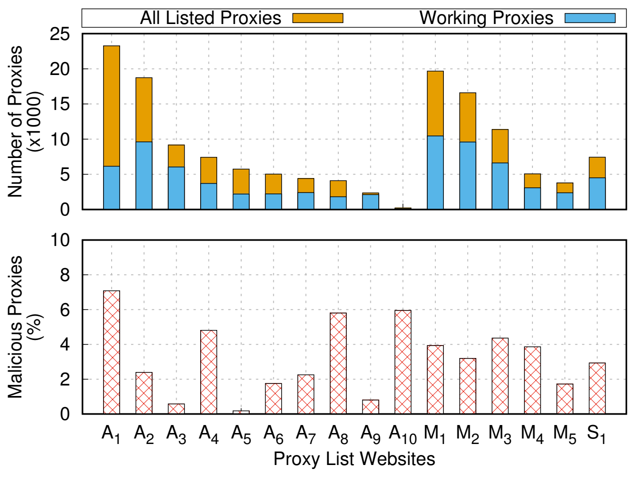

Fig. 2. Total number of proxies collected from each proxy list website during the two-month period of the study, through automated daily crawling (A_1–A_{10}) and manual collection (M_1–M_5), as well as the proxies collected from a subscription-based website (S_1). Additionally, information regarding the number of proxies that successfully fetched the requested websites, and percentage of proxies found to perform malicious content modification.

after responding to some of our probes. However, this is not surprising, as we decided to collect all the proxies that appear in the crawled proxy list websites, and not only those having good statistics such as high responsiveness.

By following the previously described methodology for detecting content modification, we observed that 7,441 proxies (38.21% of the properly working ones) altered the content of the retrieved web page in some way. As the HTML DOM tree of the fetched pages may be modified by a proxy for various legitimate reasons, the observation of a modification event does not necessarily mean that the tested proxy is malicious, or that its actions can negatively impact a user. The main type of content alteration we observed was due to "privacy-protecting" proxies that block trackers and ads that exist in the retrieved page. We also observed proxies that inject elements or iframes that are empty, or modified the DOM of the page in such a way that cannot be classified as malicious. After employing our clustering method and manually inspecting all the clusters we identified in total 1,004 (5.15%) proxies that performed some form of malicious or unwanted modifications. A detailed analysis of the malicious proxies is presented in Section V.

A. *All Proxies Collected and Tested*

The total number of proxies we collected from each proxy list website, as well as the number of the properly working ones is presented in Figure 2. During the collection process we noticed that most of the sites ask for a money fee to generate proxy lists in a more usable format, and on occasion claim to provide even more proxies to paying users. Also, it became apparent that some websites offer far less proxies than they claim to have, and less surprisingly, some of these proxies are listed in multiple websites. Indicatively, the collection process resulted to 144,349 proxies in total, from all the websites, which correspond to only 65,871 unique proxies in our set.

1) Public proxy list websites: As presented in Figure 2, the websites that impose some restrictions with regards to

TABLE II. COMPARISON BETWEEN PROXIES COLLECTED FROM PUBLIC PROXY LIST WEBSITES AND A SUBSCRIPTION-BASED WEBSITE

	Category of Proxy Lists	Total	Working	Malicious
1.	Free Publicly Available	58,435	14,973 (**25.62%**)	872 (**5.82%**)
2.	Only Subscription Based	3,278	990 (**30.20%**)	61 (**6.16%**)
3.	Appeared in Both Categories	4,158	3,510 (**84.41%**)	71 (**2.02%**)

automatic crawling, from which we collected proxies manually (i.e., M_1–M_5), have a higher fraction of properly working proxies, compared to the automatically crawled websites (i.e., A_1–A_{10}). Our tests suggest that 53% to 63% of the proxies in these websites succeeded in fetching the requested web pages, and that around 2% to 4% of the proxies in these websites are malicious. On the other hand, for the automatically crawled proxy list websites, we observe lower percentages of working proxies, in general, and proxy list websites such as A_3, A_5 and A_9 (i.e., *https://hidemy.name*, *https://www.us-proxy.org* and *http://www.idcloak.com*) that have very few malicious proxies (i.e., less than 1%), while others reach 6% and 7%.

Two notable cases of proxy list websites that interestingly present contrary behavior are A_1 and A_9. The former listed and provided more proxies than any other visited website (23,262), but had the lowest percentage of working proxies (26.4%), and the highest percentage of malicious ones (7.08%). On the other hand, A_9 offered only 2,354 proxy during the two month period of our experiment (i.e., second lowest) but impressively, 88.9% of them were found to be working properly, and less than 1% were found to be malicious. These observations lead to the assumption that the former website focuses on having a large volume of proxies, possibly by collecting and listing proxies that appear in other websites, while the latter makes more efforts to only provide highly reliable proxies.

2) Subscription-based proxy service: For being able to draw comparisons between free public web proxies and proxies acquired from a paid service, and for detecting if they exhibit significantly different behaviors, we bought a one-month subscription from a proxy list website that claims to own over 50 thousand proxies. As we gained access to the site's proxy lists for a month, we visited it six times in total, every five days, and collected all the proxies offered (S_1 in Figure 2). We discovered that, in reality, this website provided only about two thousand proxies each time it was visited, and claimed that only these proxies were available at the time of visit. In total, we collected only 7,436 proxies from the subscription-based website, and interestingly, only 3,278 of them were not already included in our set.

In Table II we present statistics for (1) the proxies collected from public proxy list websites (2) the proxies offered only by the subscription-based website and (3) the proxies appeared in lists of both the public websites and the subscription-based one. As shown in Table II, the percentages of properly working proxies in (1) and (2) are very similar, 25.62% and 30.20% respectively. After testing and assessing the proxies in our dataset, we identified that these two categories have a similar percentage of malicious proxies, i.e., 5.82% and 6.16%. However, interestingly, we identified that the proxies found in both groups have a significantly higher working rate (84.41%) and the smallest malicious rate (2.02%). Furthermore, regarding the malicious proxies detected in the free and subscription-based websites, we did not identify any significant differences in the modifications they perform, and in general, in the behavior they exhibit. Specifically, for the proxies collected from the subscription-based website, we identified modifications in accordance to all the different high-level classes of malicious behavior that are presented in Section V.

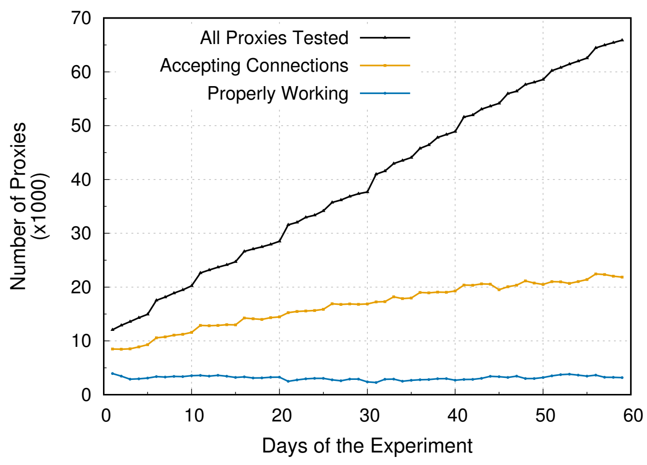

Fig. 3. Total number of proxies in our set, once the daily collection process is completed, and number of proxies found to be alive and properly working on each day of the experiment.

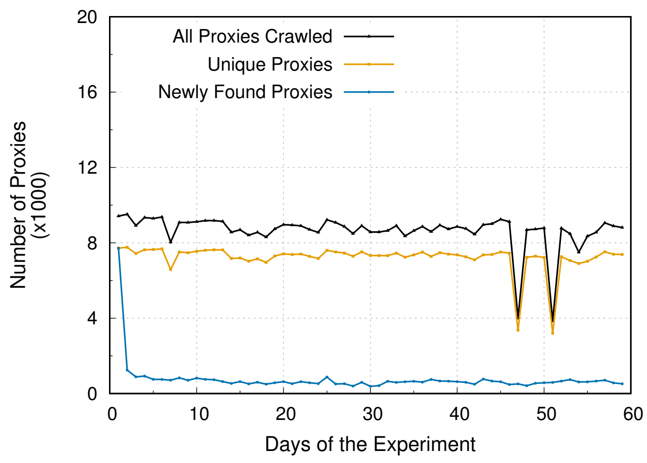

Fig. 4. Number of proxies our crawler automatically collected from the 10 proxy list websites each day of the experiment (i.e., A_1–A_{10} in Figure 2).

3) Daily statistics of proxies: In order to understand the dynamics of the proxy collection process, and how our probing mechanism affects the scalability and performance of the system, in Figure 3 we present the total number of proxies in our dataset after the daily collection process is completed and the set of proxies is updated. In this figure, we can spot the contribution of the manual collection process, as a small increase (i.e., a noticeable step) is observed for the sixth, eleventh day etc. In addition, Figure 3 presents the number of proxies that had been found to accept incoming connections (i.e., alive), each single day of the experiment, as well as the number of the properly working proxies(i.e., retrieved the requested websites). An interesting observation is that, while

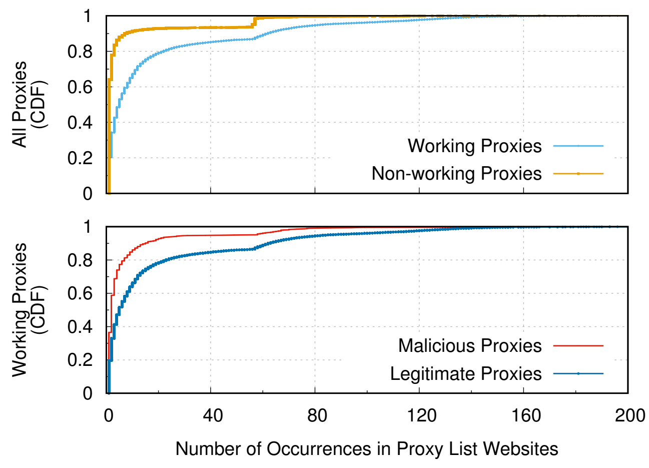

Fig. 5. Cumulative fraction of proxies in regards to the number of times each proxy occurs in the crawled proxy list websites.

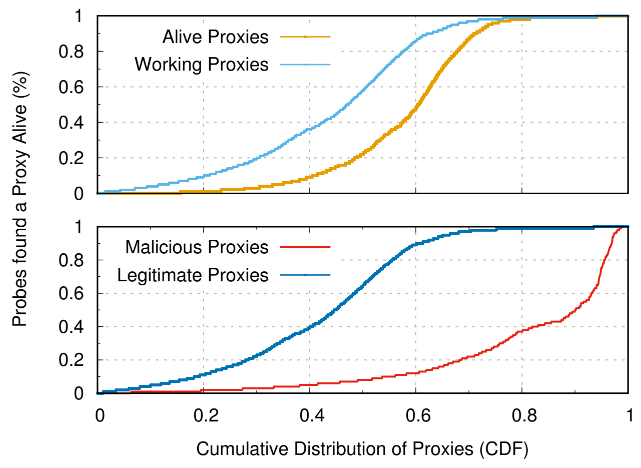

Fig. 6. Cumulative fraction of proxies found to be alive by our probes.

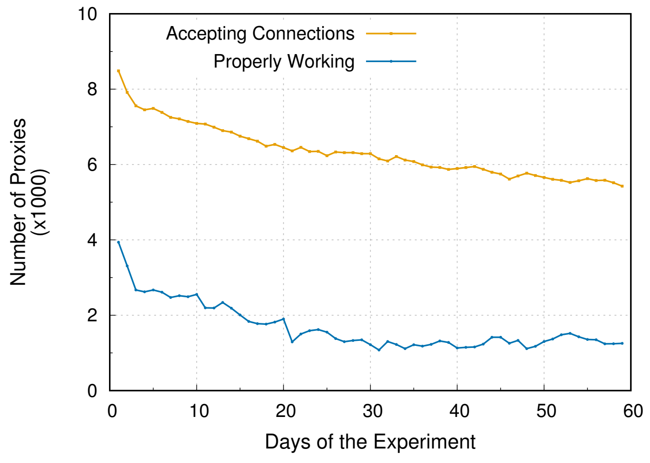

Fig. 7. Number of proxies found to be alive and working each day of the experiment, from the initial set of proxies collected on the first day.

the number of daily alive proxies increases, in general, it increases in a much lower rate than the collected proxies, and while at the beginning of the experiment about half of the proxies were alive, only one third is alive in a given day after two months. In addition, as depicted in this figure, the number of working proxies per day is around four thousands, for the whole duration of the experiment.

In Figure 4 we present the number of newly found proxies per day, for the duration of our experiment. We collected close to 10 thousand proxies each day from the sites we crawled automatically (i.e., A_1 - A_{10}) and only about 4% to 6% of them were new to our set. It is noted that the two sharp drops shown in Figure 4 are due to one proxy list website that was inaccessible for these two days. Interestingly, this drop did not affect the number of new proxies found, which indicates that the particular site does not update its list very often.

4) Occurrences in proxy list websites: Figure 5 presents the cumulative fraction of proxies in regards to the number of times each proxy occurs in the crawled proxy list websites (A_1 - A_{10}). This provides interesting insights about the persistence of proxies in these websites, and how the proxies appear across multiple different websites. Interestingly, we observe that the great majority on non-working proxies appear only very few times in the proxy list websites. Specifically, we observe that around 64% of the non working proxies appeared only once (in just a single website), and that 88% of them appeared only up to five times. This suggests, that many proxy list websites regularly update their lists by removing proxies that are non working. However, it is observed that about 10% of the non-working proxies appear multiple times, which span up to the whole duration of our experiment. On the other hand, only 20% of the working proxies appear only once, and about half of them (53%) up to five times. Interestingly, we observe that the malicious and benign proxies follow similar distributions to the working and non-working ones, respectively. Even though the number of occurrences depends on the first time a proxy was added to a list, this figure shows significant differences regarding the persistence of proxies in the crawled websites.

5) Lifespan and reliability: As discussed in Section III-B, our framework sends TCP probes to each proxy 22 times every day, almost in every hour, in order to identify which proxies are listening for incoming connections. As we send probes to all the proxies almost every hour, for the duration of the experiment, these probes can be used for estimating the fraction of time each proxy was alive. In Figure 6 we present the CDF of proxies that were found to be alive by our probes. As can be seen in Figure 6 (top), 15.66% of all the alive in our dataset responded to very few of our probes (less than 0.1%), around 50% of them were found to be alive less than 22% of the times they were probed, and only around 10% responded to more than 72% of the probes sent.

When focusing our analysis only on the properly working proxies, we observe that the legitimate proxies respond to a significantly large number of our probes, and that in general these proxies are alive for longer times. Specifically, we observe that 50% of the legitimate proxies respond to more that 64% of our probes, and that around 40% of them respond to more than 90% of the probes sent. Interestingly, the proxies we identified as malicious appear to be alive significantly fewer times. We observed that 50% of the malicious proxies responded to less than 9% of our probes, and that only 20% of them were found to be alive by more than 37% of our probes.

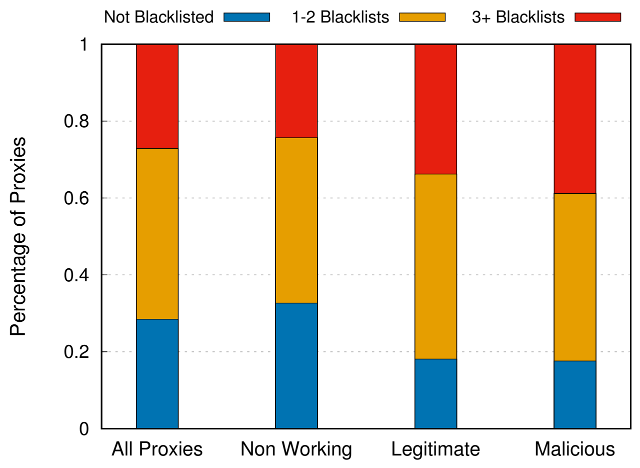

Fig. 8. Percentage of proxies that are included in DNS-based blackhole lists.

TABLE III. ERROR MESSAGES COLLECTED FROM THE NON-WORKING PROXIES

Type of Failure	Number of Proxies
Network Errors	15,133
Not Permitting Use	1,087
Misclassification	89
Partial downloads	27
Total	16,336

Furthermore, in order to examine the longterm lifespan of proxies, in Figure 7 we present the number of alive and properly working proxies for each day of the experiment, by considering only the initial 12,078 proxies we discovered on the first day of the experiment. We observe that the percentage of alive proxies starts at 70.25%, then drops to 52.06% after a month, and at the end of the experiment reaches 44.93%. Also, it is clear that while 46.40% of the alive proxies were properly working on the first day, after one and two months only 21.40% and 23.08% of were found to be still working.

6) Inclusion in blacklists: It is commonly believed that open proxies are not only used for legitimate purposes such as accessing blocked content and protecting a user's privacy, but also for other nefarious or questionable activities. Alonso et al. [7] examined the use of a proxy they set up, and observed many cases where their proxy was used for fraud. A question that arises is whether the public use of proxies results in their IP address to be blacklisted, and if this affects their availability.

To that end, we examined whether the proxies in our dataset are included in any online blacklists that contain IP addresses observed to engage in malicious activities. To accomplish that, we leveraged *dnsbllookup.com*, a service that checks if an IP address exists in 66 DNS-based blackhole lists (DNSBL). These lists contain IP addresses associated with compromised or otherwise malicious systems that perform illicit activities, mainly related to spam distribution. As can be seen in Figure 8, most of the proxies in our dataset are included in at least one blacklist, with only 28.46% of them not being found in any of these lists. When focusing our analysis on the working proxies, we observe that 18.1% of the legitimate proxies, and 17.59% of the malicious, are not included in any blacklist. We also observe that 32.66% of the non properly working proxies are not being included in any blacklist, which is significantly higher than the percentages observed for the legitimate and malicious proxies (i.e., working proxies in our dataset).

TABLE IV. AUTONOMOUS SYSTEMS IN WHICH MOST HTTP PROXIES ARE LOCATED (TOP 10 AUTONOMOUS SYSTEMS).

	Autonomous System	Proxies	Country	Owner
1	AS4837	6,834	CHN	China Unicom
2	AS4134	4,887	CHN	China Telecom
3	AS13335	2,578	USA	CloudFlare
4	AS8048	2,372	VEN	Cantv
5	AS17974	1,830	IDN	PT Telkom Indonesia
6	AS50896	1,399	RUS	MediaServicePlus LLC
7	AS200557	1,399	RUS	Petersburg Internet Network
8	AS15169	1,272	USA	Google Cloud
9	AS15003	1,076	USA	SpeedVM Network Group
10	AS14061	1,043	USA	Digital Ocean

TABLE V. AUTONOMOUS SYSTEMS IN WHICH MOST OF THE MALICIOUS PROXIES ARE LOCATED (TOP 5 AUTONOMOUS SYSTEMS).

	Autonomous System	Malicious Proxies	Country	Owner
1	AS4837	433	CHN	China Unicom
2	AS17974	234	IDN	PT Telekomunikasi Indonesia
3	AS4134	61	CHN	China Telecom
4	AS56041	16	CHN	China Mobile Communications
5	AS131269	14	IND	Beam Telecom

B. Non-Working Proxies

As mentioned earlier, only 38.38% of the alive proxies, that responded to our probes, succeeded in fetching at least one of the requested websites. In order to understand the main reasons behind this high percentage of non-working proxies, we examined the responses of Selenium and Firefox. In many cases there was no response or just a blank page. The absence of a response can possibly means that the particular proxies were not alive at that time, or that they were under heavy load. These reasons seem very possible, after considering the low, in general, reliability of open web proxies.

However 16,336 of the non-working proxies generated error messages related to their cause of failure. We were able examine these messages, since many of them were very similar across different proxies, and group them into generic categories, which are presented in Table III. We discovered that most of the proxies (92.63%) were not working due to network errors they encountered while trying to send/forward our request. Among many different network errors, the most common ones in this category were DNS-related errors. In most cases, proxies had trouble getting an answer from the DNS server regarding the website we were looking for.

The second category (6.65%) consisted of responses indicating that we were not allowed to use these proxies, such as "Authentication Required" and "Access Denied." Typically, the proxies in this category authenticate their users by their IP address, and typically such proxy services are not for free. The "misclassification" category is related to cases where the specific IP-port pair listed in the proxy list websites does not actually correspond to an HTTP proxy, but to a different service (e.g Tor exit nodes). Finally, we have a category

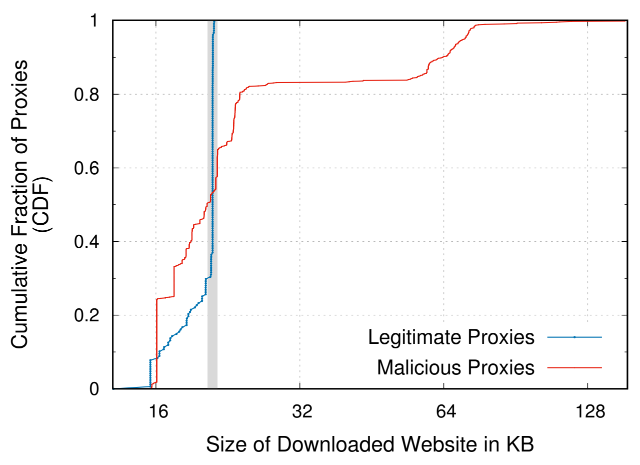

Fig. 9. Cumulative fraction of the downloaded content size when fetching the dynamic honeysite h_2.

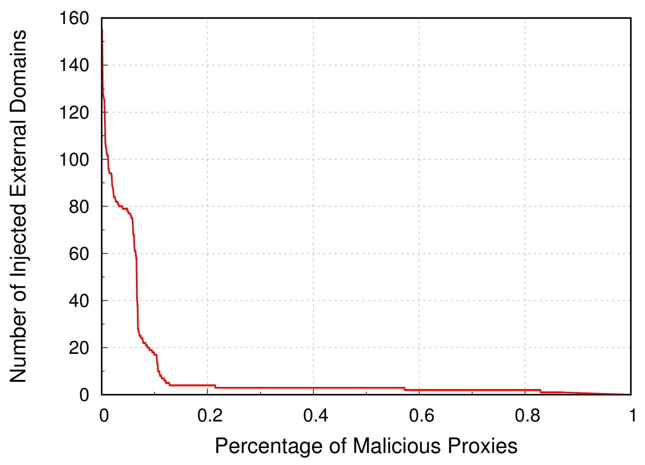

Fig. 10. Number of different domains contacted by the code injected by malicious proxies.

called "Partial downloads" for proxies that started fetching the requested website, but did not succeed in downloading it, probably due to network errors.

C. Differences between Benign and Malicious Proxies

1) Location: To get further insights about the proxies we collected in this study, we used the geolocation service offered by *ip-api.com*, which provides information regarding the country an IP address is registered, the ISP and Autonomous System (AS) it belongs to, and so on. In Table IV we present the most common Autonomous Systems, in which many of the proxies we collected belong to. From the 65,871 proxies we crawled in total, we found 60,708 unique IP addresses. We identified that 11.25% of the proxies belong to a single AS in China. Furthermore, as can be seen in Table IV, most of the proxies in our dataset belong to Autonomous Systems in China, USA and Russia.

As our main focus is on malicious proxies, we try to investigate if there is any correlation between the behavior of proxies and their location. As can be seen in Table V, Chinese Autonomous Systems dominate our list, specifically containing 50.79% of the malicious proxies we detected. Interestingly, another big batch of malicious proxies (23.3%) belongs to an Indonesian Autonomous System.

2) Fetched content size variation: A basic way malicious proxies differ from legitimate ones is in relation to the size of the downloaded HTML file. Since new code is typically injected, one would expect the size of the downloaded content to always be larger than the original. As mentioned, however, proxies that only block certain kinds of content, such as trackers and ads, which cannot be considered as malicious, may actually result in size reduction. Another type of proxies that we observed are proxies that block trackers and ad networks only to replace them with their own. Therefore, a downloaded HTML file belonging to these types of proxies does not have to differ much in size of that of the original page. This effect would be more evident for rich web pages that already contain trackers and ads, similar to our own honeysite h_2 (in order for the proxy to be able to remove some content, and thus reduce the size of the retrieved page)

Figure 9 shows the distribution of the downloaded content size when retrieving our most complex honeysite h_2, which has dynamic content due to the inclusion of trackers and fake ad networks. The expected download size for honeysite h_2 ranges around 21KB. It can be seen that both legitimate and malicious proxies have downloads falling within the expected size of the page, while also both retrieve fewer content, mostly due to proxies that block tracking and advertising content. Approximately half of the malicious proxies have downloads with bigger size than expected while about 20% of them double or more the size of the page.

V. Malicious Proxies

In this section we focus our analysis on the malicious proxies we have detected and thoroughly discuss about the types of content modification and injection these proxies perform on users' relayed traffic. We categorize content modification incidents according to well-defined malicious behaviors and we discuss the semantic nature of each injection, for shedding more light to the mechanics of a malicious proxy's operation. The following analysis was not a trivial task as we had to overcome many practical challenges, such as non-existing links, obfuscated JavaScript code, etc.

A. Resources Fetched from Third Parties

In order to track the culprits of the identified injections and gain a better understanding of the behavior of injected content, we executed the JavaScript code that was injected by malicious proxies in the websites they fetched, in an automated way, and monitored the resources fetched from third-party domains. We ran this experiment locally, in a controlled environment, and kept track of all the contacted IP addresses and external domains. In order to accomplish that, we used our own trusted proxy for logging and forwarding all the traffic generated by injected/modified content.

In addition, we used a *whitelist* containing all the domains we expected to be contacted by our honeysites, thus identifying which requests were sent due to malicious injected code and which ones due to the legitimate dynamic functionality. Two limitation we encountered during this analysis are: (1) some

TABLE VI. NUMBER OF MALICIOUS PROXIES THAT INJECT CODE FOR SENDING INFORMATION AND FETCHING RESOURCES FROM SPECIFIC THIRD PARTY DOMAINS.

	Domain	Proxies
1	tongji.baidu.com	556
2	cfs.uzone.id	140
3	a01.uadexchange.com	124
4	up.filmkaynagi.com	113
5	a.akamaihd.net	109
6	urlvalidation.com	107
7	i.qkntjs.info	106
8	adnotbad.com	106
9	ratexchange.net	105
10	i.tonginjs.info	105
11	www.onclickcool.com	104
12	agm.abounddinged.com	104
13	yellow-elite.men	103
14	qnp.demisedcolonnaded.com	102
15	intext.nav-links.com	102
16	www.tr553.com	101
17	ruu.outputsteddy.com	101
18	s.lm15d.com	74
19	rtax.criteo.com	72
20	www.donation-tools.org	69

of the domains/resources were no longer available and (2) our whitelist had the domains of the fake ad networks we used, which means that requests from proxy-injected content to these domains would be ignored.

By following this approach, we were able to collect information regarding all the domains to which the injected code sends requests, as well as the IP addresses that correspond to these domains. In Table VI we present a list that contains the Top-20 most contacted domains, as a result of malicious injected code. In total we observed requests towards more than 362 different domains. Additionally, we identified that 181 domains were contacted by five or more malicious proxies. We identified 98 domains that were contacted only by a single proxy each, while for the most popular domain (*tongji.baidu.com*), requests were sent by content that was injected from 556 different malicious proxies.

Figure 10 presents the number of unique domains that were contacted by each malicious proxy, without considering the domains originally included in our honeysites (i.e., whitelisted domains). As can be seen in Figure 10, 12.94% of the malicious proxies do not fetch any third-party resource. In reality, most of the aforementioned proxies try to fetch external resources but the third parties hosting these resources are no longer reachable. Also, we observed a few cases of proxies that do not attempt to fetch any resource, but try to load them from the browsers local storage. Furthermore, we observed that 4.08% of the malicious proxies fetch only one external resource, and that 57.17% fetch resources from more than two domains, while 8.56% of the proxies contact more than 20 different domains. Interestingly, the top 12 proxies issue requests to more than 100 different domains. We discuss about the purpose of these injected domains in detail in Section V-D. It is observed that many different proxies contact the same third-party domains in order to fetch particular libraries that implement specific malicious or questionable functionalities (e.g., user fingerprinting).

B. Generic Categorization of Rogue Behavior

Advertisements. Malicious proxies often inject ads in the websites they fetch. We visually inspected a random subset of the malicious proxies' downloads for determining what type of ads they inject. Several ads are displayed in a language compatible with the proxy's geographical location, while sometimes ads can be displayed in a language compatible with the client's location. Certain ads exercise a legitimate behavior, without tricking the user into clicking them. However, some of them completely cover the target website, while others are much more aggressive, e.g., embedding videos from YouTube. For example, *Fyne.in* covers the whole page and displays video messages, while several others display fake rewards and alerts instructing the users to fix their infected computer.

A fraction of the ads embed adult content and services for meeting people at the client's geographical area. We randomly clicked some of the injected ads and found that some of them redirect to malware distributing sites that have been already blocked by Google safe browsing.

Tracking. As described in Section III, we recorded all the requests to third parties with Firefox and Firebug. We observed that these parties typically set third party cookies to the users' browser. In some cases, these cookies share the same `ID`, which indicates that third parties are cooperating to track the user through cookie syncing [13]. Also, malicious proxies were found to inject JavaScript from Mixpanel [21], which are used to track the actions of the user in the visited websites.

Fingerprinting. Malicious proxies often attempt to maintain tabs open on the user's browser for launching fingerprinting attacks. We identified certain injected content that make requests to third parties such as *advedia360.com*, which emits JavaScript in the HTML code of the page that uses fingerprintjs2 [31]. Fingerprintjs2 is a well known library for identifying web browsers with a high accuracy. In a similar fashion, proxies that perform canvas fingerprinting [22] were also detected, like *104.238.146.90:80*. This specific proxy redirects the user to a website that sells skin products, and claims that the requested website could not be found.

Privacy leakage. Malicious proxies inject scripts, which once rendered, trigger requests to third-parties for exfiltrating sensitive user information. This information may include static fields like geolocation data, or may be the outcome of a fingerprinting methodology. For example, we found many instances of an injection that emits requests to particular third parties affiliated with the ad market, specifically to *BlueKai* and *Eyeota.com*, which are a data management platform and a data supplier.

We also uncovered injection libraries that target sensitive user information that exists in certain pages the user possibly visits, as well as JavaScript that harvests e-mails and phone numbers. In rare cases, we identified scripts that search for specific DOM elements of popular web pages (like QZone, a Chinese social network) for extracting personal information, such as the user's blood type. Another example is given by the proxy *186.237.46.32:8080* which embeds a search form from *booking.com* and *google.com* in the fetched website, and then tracks whatever information the user used them for. Therefore, we can undoubtedly infer that malicious proxies, and their injections, are seriously affecting users' privacy.

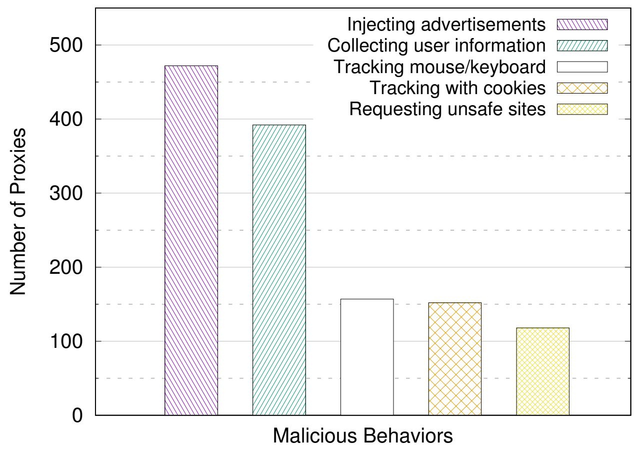

Fig. 11. High-level categorization of observed malicious behaviors.

Malware. Several injected libraries contain JavaScript code that triggers requests towards sites that are blacklisted and blocked by Google safe browsing. Upon inspection we found that malicious proxies often attempt to land malware on the user's host or redirect the browser to a drive-by download page [25].

Unclassified behavior. We had some difficulties in understanding certain injections, in a small fraction of cases. This is mainly due to obfuscated/minified JavaScript code, and also because some third party domains were no longer valid during the analysis phase. This is not unexpected, as very often the operators of servers that host malicious content change domains to avoid being detected and blocked. In certain cases, we identified proxies that inject a `script` or an `iframe` element with no target (i.e., `src` is empty). This may be due to a templating framework used, which mistakenly or on purpose adds this element, while a higher-level script chooses when to include a payload or not. Also, in some other cases instead of fetching data, the injected scripts tried to use resources from the browser's local storage. Proxies that exhibit such behaviors are marked as *suspicious*, but are not considered as malicious. Any new information about these cases may result in considering them malicious in the future.

We summarize all the different types of injections performed by malicious proxies in Figure 11. We have grouped injections according to the high-level semantic goal of the malicious proxy. If the injections of a proxy belong to multiple high-level categories, this proxy is counted in all the respective categories in Figure 11. Also, it is noted, that *suspicious* proxies are not counted at all, unless they simultaneously perform attacks the belong to the aforementioned categories.

As shown in Figure 11, we have discovered 472 malicious proxies that inject ads in the page served to the user. Some of these ads are fetched from legitimate ad networks, while others serve more devious purposes (e.g., redirect to unsafe sites). We consider that a proxy is tracking user's information if it attempts to extract information regarding the device, the OS or the browser of the user, as well as user's personal information (e.g., name, e-mail, address). Such behavior can be detected by looking for injected JavaScript code that extract information about the language and timezone of the user, the user agent, system language, platform, plugins, dimensions of the screen, device pixel ratio, color depth and others. We also include cases where the proxies use well known tools like `fingerprintjs`. We discovered 392 proxies guilty of this behavior. We also identified JavaScript code for tracking keyboard and mouse movements, and for setting and reading cookies, by 157 and 152 malicious proxies respectively. Moreover, we identified 118 proxies that make requests to sites that are blocked by Google's safe browsing, potentially for installing malware on the user's machine.

C. Content-dependent Injections

After following the methodology described in Section III for detecting malicious proxies and documenting their behavior, we wanted to investigate if their behavior changes according to the content of the website they fetch. By comparing the domains each proxy repeatedly (i.e., more than once) requests for each honeysite, we found that 37 proxies request 63 domains in total for (h_2) but not for (h_1). This behavior confirms our hypothesis that some proxies change their behavior depending on the content they encounter. More specifically, we detected 10 proxies that injected a domain inside the iframe of the AdSense ad for (h_2) and the real site (but not in the static honeysite h_1). It seems that this injection tries to insert an ad at the specific place where the AdSense ad should have normally been. Also, from the 362 domains injected in total in our honeysites, we found 48 that were not being injected in the real website we used for testing the proxies. This behavior indicates that some proxies tend to change their injection according to the content they fetch.

In order to investigate in more detail the behavior of malicious proxies in regards to the already existing ads, we used a script to automatically detect whether they attempt to modify the advertising IDs of any of the fake ads we had in our dynamic honeysite (h_2). We detected two such proxies, which targeted our advertisement from Media.net, and successfully changed our *fake* publisher's ID with theirs. This modifications performed 9 and 11 times respectively. The first proxy even used two different publisher's IDS, on different occasions. Interestingly these proxies did not inject anything else on the particular testing website, or the other two testing websites.

However, surprisingly, they did not always perform the ID replacement in (h_2). The first one injected the Media.net iframe for nine days in a row and then, the next two days fetched the page without any modifications. The second malicious proxy was fetching an unmodified version of the testing website for 18 days, without performing any content modification, and then suddenly started modifying the ID of the existing ad. This modification happened for 11 days in a row, and then, for the remaining period of our experiment, it was acting entirely as a benign proxy, not performing any modifications. To that end, we investigated if any other proxies in our dataset act in a similar way, and we detected that 41 (4.08%) of the malicious proxies do not always perform injections or modifications, but only exhibit such a malicious behavior sporadically.

D. Cases of Interesting Injections

As already discussed in the previous, in the majority of cases, malicious proxies inject JavaScript code that interacts with third-parties. It is interesting to explore the distribution and popularity of the domains used by these third-parties. A list of the most commonly contacted domains is given in Table VI.

The most contacted third-party domain is owned by a popular Chinese search engine, namely Baidu. This is *tongji.baidu.com*, which is a traffic analysis service similar to Google analytics. In this case, a `div` element is injected in the DOM tree of the fetched website and tries to load *tongji.baidu.com/logstat.swf*. The `allowscriptaccess` tag is set to *always*, meaning that the `.sfw` files are granted access to the HTML DOM regardless of where they are hosted. Almost half of the malicious proxies we detected were found to be in contact with that domain. Furthermore, *cfs.uzone.id* and *a01.uadexchange.com* belong to the same cluster of injections. In that case a `script` element is injected at the end of the website's DOM tree, for fetching and showing ads.

Many of the domains presented in Table VI are reached due to a particular piece of JavaScript code that is injected in the fetched website. We tracked this code and found it used in injections by 141 different malicious proxies. Some of them used different variations of the code, resulting in a few different domains to be requested, but the core remained the same. This JavaScript code, which can be used as a toolkit, allows the attacker to set and read cookies, to get information that can be used for fingerprinting the user (i.e., device, browser), to deploy ads, and to send AJAX requests among others.

Moreover, an interesting injection loads code from *ratexchange.net*, which behave differently depending on the site the user visits. For example, if the user visits *ok.ru* (i.e., a russian social network) it injects ads from *cdnpps.us* with the attackers' publisher ID. Alternatively, if it detects that the user is using the *Yahoo search engine* it hijacks the URL and redirects to *sugabit.net*, which is a site that looks like a search engine, but in reality, is a browser hijacker.

Another interesting cluster of malicious proxies fetches JavaScript code from *d.chaoliangyun.com*. It strips all of the user's HTTPS connections and tries to extract information about the user, as well as possible information the page may contain like phone numbers or e-mails, by using specifically crafted regexes. Additionally, it monitors all the sites the user visits and searches for site specific information that could reveal very sensitive user information.

VI. Discussion and Limitations

This work is an initial attempt to assess the extent of malicious content modification by rogue HTTP proxies. While it is possible that some popular proxies are not included in our proxy set, we consider this as highly unlikely. We believe that users look for proxies in the same way we do – via Google search of proxy lists. Thus, in order for a proxy to be popular, it must be included in proxy lists that are highly ranked in Google search results.

We opted for a simple but effective scheme that relies on decoy websites under our control to detect content modification without false positives. This approach, however, is not robust against determined rogue proxy operators who may anticipate our attempts, and refrain for performing any suspicious activity. Currently, detecting our testing efforts is relatively easy, since we use the same set of honeysites hosted on a single server, making them easily identifiable. A first step for mitigating this issue is to significantly expand our set of honeysites, and randomly expose only a few of them to each proxy. Eventually, a more generic strategy is needed against sophisticated proxies who may choose to target only a subset of pages, e.g., only those containing ads from a certain ad network, high-profile and popular websites, and websites of certain interests. Consequently, a broad, diverse, and constantly evolving set of pages should be used as targets, to achieve the required stealthiness and coverage.

Although we used the proposed methodology as part of a large-scale measurement effort, the core content modification logic in conjunction with honeysites, can also be used as a standalone tool, for on-demand testing of any proxy that users are about to trust their traffic with. To that end, we have implemented a web service that follows our methodology for regularly crawling proxy list websites, detecting content modification incidents, and generating on a daily basis a list of proxies that have not been found to perform any modification or injection. As some rogue proxies may only perform modifications selectively and sporadically, we keep this list relatively short, for being able to re-test each proxy multiple times every day.

VII. Conclusion

In this work, we carried out a large-scale analysis of open HTTP proxies, focusing on the detection of rogue proxies that engage in malicious or unwanted content modification. We developed a DOM comparison technique for detecting HTML alterations from third parties, and during a period of two months, we made multiple requests to 65,871 proxies, requesting honeysites under our control. Our findings suggest that 38.21% of the working proxies modify the page they fetch in some way, while 5.15% of the proxies we tested performed some form of malicious content modification. We presented a detailed analysis of the characteristics of legitimate and malicious proxies, and identified and analyzed various forms of malicious injections. Our results indicate the important privacy implications of trusting user traffic to unknown open proxies, operated under questionable motives. Among the observed behaviors, we encountered many privacy compromising incidents, with many proxies collecting and sharing private user information with third parties, let alone making requests to pages that deliver malware.

Acknowledgments

We would like to thank our shepherd, Jelena Mirkovic, and the anonymous reviewers for their valuable comments. This work was supported by the H2020 project SHARCS, under Grant Agreement No 644571, by the H2020 project UNICORN, under Grant Agreement No 731846, and by the H2020 project SMESEC, under Grant Agreement No 740787. In addition, this project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie Grant Agreement No 690972. The paper reflects only the authors' view and the Agency is not responsible for any use that may be made of the information it contains.

References

[1] "How hackers abused Tor to rob blockchain, steal bitcoin, target private email and get away with it," February 2015, http://www.forbes.com/sites/thomasbrewster/2015/02/24/blockchain-and-darknet-hacks-lead-to-epic-bitcoin-losses/.

[2] S. A. Alrwais, A. Gerber, C. W. Dunn, O. Spatscheck, M. Gupta, and E. Osterweil, "Dissecting ghost clicks: Ad fraud via misdirected human clicks," in *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC)*, 2012.

[3] J. Appelbaum, M. Ray, K. Koscher, and I. Finder, "vpwns: Virtual pwned networks," in *2nd USENIX Workshop on Free and Open Communications on the Internet. USENIX Association*, 2012.

[4] S. Arshad, A. Kharraz, and W. Robertson, "Identifying extension-based ad injection via fine-grained web content provenance," in *Research in Attacks, Intrusions, and Defenses - 19th International Symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings*, 2016, pp. 415–436.

[5] ——, "Include me out: In-browser detection of malicious third-party content inclusions," in *Proceedings of the 20th International Conference on Financial Cryptography and Data Security (FC)*, 2016.

[6] S. Chakravarty, G. Portokalidis, M. Polychronakis, and A. D. Keromytis, "Detecting traffic snooping in Tor using decoys," in *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2011, pp. 222–241.

[7] A. Chema and M. Fernandez "The Sur", "Owning bad guys {& mafia} with javascript botnets." DEFCON 20, 2012.

[8] T. Chung, D. Choffnes, and A. Mislove, "Tunneling for transparency: A large-scale analysis of end-to-end violations in the internet," in *Proceedings of the 2016 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '16. ACM, 2016.

[9] V. Dave, S. Guha, and Y. Zhang, "Measuring and fingerprinting click-spam in ad networks," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 175–186, Aug. 2012. [Online]. Available: http://doi.acm.org/10.1145/2377677.2377715

[10] X. d. C. de Carnavalet and M. Mannan, "Killed by proxy: Analyzing client-end tls interception software," in *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24*, 2016.

[11] K. Finley, "Wired - half the web is now encrypted. that makes everyone safer," 2017, https://www.wired.com/2017/01/half-web-now-encrypted-makes-everyone-safer.

[12] S. Ford, M. Cova, C. Kruegel, and G. Vigna, "Analyzing and detecting malicious flash advertisements," in *Proceedings of the 2009 Annual Computer Security Applications Conference*, ser. ACSAC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 363–372. [Online]. Available: http://dx.doi.org/10.1109/ACSAC.2009.41

[13] A. Ghosh and A. Roth, "Selling privacy at auction," in *Proceedings of the 12th ACM Conference on Electronic Commerce*, ser. EC '11, 2011, pp. 199–208.

[14] H. Haddadi, "Fighting online click-fraud using bluff ads," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 21–25, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1764873.1764877

[15] C. Haschek, "Let's analyze over twenty thousand proxies." 2015, https://blog.haschek.at/2015-lets-analyze-twenty-thousand-proxies.

[16] ——, "Why are free proxies free?" 2015, https://blog.haschek.at/post/fd9bc.

[17] R. Holz, T. Riedmaier, N. Kammenhuber, and G. Carle, "X.509 forensics: Detecting and localising the SSL/TLS men-in-the-middle," in *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, 2012, pp. 217–234. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33167-1_13

[18] V. T. Lam, S. Antonatos, P. Akritidis, and K. G. Anagnostakis, "Puppetnets: Misusing web browsers as a distributed attack infrastructure," in *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, 2006, pp. 221–234.

[19] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang, "Knowing your enemy: Understanding and detecting malicious web advertising," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 674–686. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382267

[20] M. Marlinspike, "sslstrip," http://www.thoughtcrime.org/software/sslstrip/.

[21] https://mixpanel.com/, mixpanel.

[22] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5," in *Proceedings of W2SP 2012*, M. Fredrikson, Ed. IEEE Computer Society, 2012.

[23] M. O'Neill, S. Ruoti, K. E. Seamons, and D. Zappala, "TLS proxies: Friend or foe?" *CoRR*, vol. abs/1407.7146, 2014. [Online]. Available: http://arxiv.org/abs/1407.7146

[24] J. Pitts, "The case of the modified binaries," 2014, http://www.leviathansecurity.com/blog/the-case-of-the-modified-binaries/.

[25] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose, "All your iframes point to us," in *Proceedings of the 17th Conference on Security Symposium*, ser. SS'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1496711.1496712

[26] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver, "Detecting in-flight page changes with web tripwires," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 31–44. [Online]. Available: http://dl.acm.org/citation.cfm?id=1387589.1387592

[27] L. Sassaman, "The faithless endpoint: How Tor puts certain users at greater risk," Katholieke Universiteit Leuven, Tech. Rep. ESAT-COSIC 2007-003, 2007.

[28] B. Schneier, "The further democratization of QUANTUM," April 2015, https://www.schneier.com/blog/archives/2015/04/the_further_dem.html.

[29] B. Stone-Gross, R. Stevens, A. Zarras, R. Kemmerer, C. Kruegel, and G. Vigna, "Understanding fraudulent activities in online ad exchanges," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 279–294. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068843

[30] K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. Mccoy, A. Nappa, V. Paxson, P. Pearce, N. Provos, and M. A. Rajab, "Ad injection at scale: Assessing deceptive advertisement modifications," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, ser. SP '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 151–167. [Online]. Available: http://dx.doi.org/10.1109/SP.2015.17

[31] https://github.com/Valve/fingerprintjs2, Valentin Vasilyev.

[32] N. Vratonjic, J. Freudiger, and J.-P. Hubaux, "Integrity of the web content: The case of online advertising," in *Proceedings of the 2010 International Conference on Collaborative Methods for Security and Privacy*, ser. CollSec'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1929808.1929811

[33] N. Weaver, "In contempt of bulk surveillance: It's too easy," September 2015, https://lawfareblog.com/contempt-bulk-surveillance-its-too-easy.

[34] N. Weaver, C. Kreibich, M. Dam, and V. Paxson, "Here be web proxies," in *Proceedings of the 15th International Conference on Passive and Active Measurement - Volume 8362*, ser. PAM 2014. New York, NY, USA: Springer-Verlag New York, Inc., 2014, pp. 183–192. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-04918-2_18

[35] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. Weippl, "Spoiled Onions: Exposing Malicious Tor Exit Relays," in *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2014.

[36] E. J. Zaborowski, "Malicious proxies." DEF CON 17, 2009.