Automatic Extraction of Indicators of Compromise for Web Applications

Onur Catakoglu
Eurecom & Monaco Digital Security Agency
catakogl@eurecom.fr

Marco Balduzzi
Trend Micro Research
marco_balduzzi@trendimicro.com

Davide Balzarotti
Eurecom
davide.balzarotti@eurecom.fr

ABSTRACT

Indicators of Compromise (IOCs) are forensic artifacts that are used as signs that a system has been compromised by an attack or that it has been infected with a particular malicious software. In this paper we propose for the first time an automated technique to extract and validate IOCs for web applications, by analyzing the information collected by a high-interaction honeypot.

Our approach has several advantages compared with traditional techniques used to detect malicious websites. First of all, not all the compromised web pages are malicious or harmful for the user. Some may be defaced to advertise product or services, and some may be part of affiliate programs to redirect users toward (more or less legitimate) online shopping websites. In any case, it is important to detect these pages to inform their owners and to alert the users on the fact that the content of the page has been compromised and cannot be trusted.

Also in the case of more traditional drive-by-download pages, the use of IOCs allows for a prompt detection and correlation of infected pages, even before they may be blocked by more traditional URLs blacklists.

Our experiments show that our system is able to automatically generate web indicators of compromise that have been used by attackers for several months (and sometimes years) in the wild without being detected. So far, these apparently harmless scripts were able to stay under the radar of the existing detection methodologies – despite being hosted for a long time on public web sites.

1. INTRODUCTION

Despite the constant effort by the security community and the attempts to raise awareness and better educate web developers, software vulnerabilities in web applications are still very common. Attackers routinely exploit them to steal sensitive data, to take control of the target system, or simply to deface web pages for political reasons or personal fame. In 2013, Canali et al. [4] performed a study to measure the typical behavior of an attacker after a website has been compromised – showing that many attacks result in the installation of new pages (e.g., phishing kits) or the modification of existing ones (e.g., to serve malicious code or redirect the victims to another location). This happens so frequently that it is very hard for the security community to react in time, detect the malicious or newly infected web pages, and update existing blacklists (such as Google SafeBrowsing [6] or Phishtank [20]) to protect users.

Existing approaches based on honeyclients [10, 26], web crawlers [5, 11, 24], or user reports [20], are not able to keep up with the current rate of infections. Therefore, we need new techniques to automatically distinguish, in a simple but effective way, "bad" pages from benign ones. To detect the presence of malicious programs in traditional systems, the forensics community uses the so-called *Indicators of Compromise* (IOCs), i.e., simple network or operating system artifacts whose presence is a reliable indicator of a computer intrusion or malware infection. For example, the presence of a certain entry in the Windows Registry or of a file with a given MD5 in a temporary directory may be associated to a certain banker trojan. These indicators are often used as part of malware detection and investigations [9] and are often shared between experts as part of other threat intelligence informations [16]. Unfortunately, to the best of our knowledge, the use of indicators of compromise has never been studied in the context of web applications.

Our work starts from a simple observation that we made after several years of operation of a web honeypot: Attackers often use external components in their malicious or compromised pages. For example, these pages often rely on JavaScript code to perform a wide range of actions. In our experience we noticed that these accessory scripts are rarely installed by the attacker on the compromised hosts, but they are instead included from public URLs hosted on remote machines. A possible reason for this behavior is that this choice provides more flexibility for the attacker to update these components without the need to modify all the pages they had previously compromised. However, this may also seem like a potential weakness, as this part of their infrastructure could be easily detected and taken down – jeopardizing a large number of infected pages.

Quite surprisingly, while investigating some of these remote components, we discovered that in the vast majority of the cases they were not malicious per se. For instance, we identified three main classes of external components: popu-

WWW 2016, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
http://dx.doi.org/10.1145/2872427.2883056.

lar JavaScript libraries (e.g., `jquery`), scripts to control the look and feel of the page (e.g., by adding dynamic effects to its text), or scripts that implement reusable functionalities (e.g., to fingerprint the user browser, to disable the right click of the mouse, to overlap the page with transparent frames, or to insert an advertisement banner in the page). Since none of these categories is harmful to the final user, these components can be safely hosted by the attackers on public pages or shared hosting services, with no risk of being detected and blocked by security scanners.

The main idea behind our work is that, while these components are indeed innocuous, their presence can be used to precisely pinpoint compromised or malicious pages. In other words, the link to a particular benign JavaScript can be considered as some sort of signature of the attack – therefore acting as an *indicator of compromise for web applications*. We call this new types of indicators, Web Indicators of Compromise (WIOCs).

We believe that the extraction and use of indicators of compromise has several important advantages. In particular, while most of the existing approaches focus on the detection of *malicious* pages, our solution allows to detect *compromised* pages. This category is much broader and much harder to identify in a black-box manner. In fact, compromised pages are not necessary harmful for the user browser, but also include defacements, phishing pages, or banners to redirect users into other web sites.

Our experiments show that our system was able to extract, in average, one new indicator per day. These indicators were then used by Trend Micro, a popular antivirus vendor, to cross-check their customers' requests in their web telemetry dataset, finding thousands of users each day visiting previously unknown compromised websites.

To summarize, this paper makes the following contributions:

- To the best of our knowledge, we are the first to propose the use of indicators of compromise for web applications.
- We propose a novel technique to automatically extract and validate these indicators – starting from the data collected by a web honeypot.
- We discuss several features that can be used to distinguish good indicators of compromise from components that are also used as part of benign websites.
- We tested our system over a period of four months. In this period, almost 100 new WIOCs were extracted and validated from our dataset. Finally, we use these indicators in collaboration with Trend Micro to estimate the number of users that are affected by compromised webpages that include these components.

The rest of the paper is organized as follows. In Section 2 we present an overview of what happens when an attacker compromise a web application, and we introduce an example of the indicators of compromise that are the focus of our paper. In Section 3 we introduce our approach and in Section 4 we present the results of our experiments. We then select and discuss in more details a number of case studies in Section 5. Finally, Section 7 discusses the related work and Section 8 concludes the paper.

2. AN OVERVIEW OF THE COMPROMISE OF WEB APPLICATIONS

In this section we introduce a real example of how attackers approach and compromise a vulnerable web application, summarizing their actions both during and after the attack is performed. This process, already described in more details in previous studies [4, 12], serves as motivation for our work.

Usually, attackers start by taking advantage of search engines to find their targets. They generally rely on automated bots to search for a set of keywords (typically called Google dorks) which allow them to find a number of web sites that are mis-configured or that are likely affected by a certain vulnerability. For example, web sites that expose MySQL history files can be retrieved using the Google query "`?intitle:index.of?".mysql_history`". Once the attacker finds her targets, she can proceed with the exploitation phase, again typically performed by automated scripts.

In this paper, we are particularly interested in what happens after the attacker has successfully compromised the target application – in what Canali et al. [4] called the *post-exploitation phase*. In this phase, attackers try to achieve their final goal which could be to install a webshell, to deface the home page with a political message, to send spam, or to install a phishing page. These goals are generally achieved by either uploading new files on the compromised machine or by modifying the sources of the existing HTML pages. Either way, the attacker often needs to use a number of JavaScript libraries which can be uploaded as well on the compromised machine or just included from a remote source. Since our primary goal is to identify indicators of compromise for web applications based on these remote components, in the rest of the paper we will not focus on means of exploitation and on the techniques commonly used to compromise the web applications.

One common misconception about the post-exploitation phase is to consider all the components uploaded by the attacker after a successful exploitation as malicious. Although among all these uploaded components a portion of them is indeed responsible to perform some sort of malicious activity (such as malware distribution, exploit kits, or phishing pages), we discovered that the majority of them are often not related to any type of malicious behavior. On the contrary, the post-exploitation phase usually involves the usage of a number of harmless JavaScript components to work properly.

For example, Figure 1 shows a snippet of code extracted from a compromised web application. In this example, the attacker injects a remote JavaScript code (i.e., `ciz.js`) to the defaced victim web page. By retrieving this remote component, we discovered that it only contained two lines of code, which are reported in Figure 2. The script first creates a new image object and then its source URL is set according to the value of `location.href`.

The goal of this component seems to be to log compromised web pages by sending a signal back to the attackers. Interestingly, the same script is included in a number of popular web shells which are distributed (and backdoored) by the same hacking group, as a mechanism to promptly detect and gain access to third party installations. Even though this code may look suspicious when manually examined because of the use of the `r00t` leetspeak in the URL, automated

```
...
<head>
<meta http-equiv="Content-Language"
    content="en-us">
<meta http-equiv="Content-Type"
    content="text/html;
    charset=windows-1252">
<title>4Ri3 60ndr0n9 was here</title>
<SCRIPT SRC=http://r57.gen.tr/yazciz/
    ciz.js> </SCRIPT>
...
```

Figure 1: HTML example of a compromised web page

```
a = new /**/ Image();
a.src = 'http://www.r57.gen.tr/r00t/yaz.
    php?a=' + escape(location.href);
```

Figure 2: Source code of `ciz.js`

scanners only looks at the maliciousness of the file itself and, inevitably, this simple piece of code is not detected as malicious by any available system or antivirus product. As a result, this JavaScript component could be hosted on any public page, without the risk of raising any suspicion from security tools. Moreover, this gives the attacker the advantage of rapidly changing the URL in all compromised pages, without the need to re-deploy the JavaScript file on all the target machines.

As we further investigate the websites which use this JavaScript as an external resource, we foud that other websites which include the same script were also compromised by the same hacking group. Also in the other compromised sites the script was included at the same place in the code as it is shown in Figure 1, and all the defaced pages looked identical when visited.

This very simple example perfectly summarizes the idea behind our technique: a little and harmless script included by attackers in compromised pages could be used to precisely fingerprint the action of these attackers and therefore can serve as an *indicator of compromise* for web applications. In our experiments, as described in more details in Section 4, we identified many of these examples ranging from few to thousands lines of code, and from custom scripts to popular libraries. We believe that this type of indicators of compromise can complement existing detection techniques that are purely based on manual reports or on automated scanners that – unfortunately – can only identify malicious components.

3. APPROACH

As explained in the example presented in the previous Section, our idea is to analyze compromised and malicious web pages – looking for seemingly innocuous external resources that can be used to identify a certain group of attackers or a certain attack campaign.

In the rest of this section we describe each step of our automated technique.

3.1 Data Collection

The first component of our system is a high-interaction honeypot that we use to observe the behavior of the attackers and collect the pages they modify or they upload into the vulnerable system. The role of this component is only to collect a large number of pages compromised by attackers. Other techniques could be used to obtain a similar dataset, for instance by crawling the web or by using intelligence feeds from security companies.

Our honeypot infrastructure, summarized in Figure 3, is implemented as previously described by Canali et al. [4]. The deployment consists of proxy services (associated to 500 different domain names), which redirect the traffic through a VPN gateway to seven virtual machines running in our premises. Each VM is responsible to run a different vulnerable web applications isolated in a Linux container. As attackers exploits these applications, they gain full control of the corresponding container – where they are free to modify existing web pages and install new ones.

In order to discover what has been modified after an attack is performed, each VM automatically collects and compares the original state of the container with the exploited state. When a difference is detected between two states, all the files that are modified or uploaded by the attacker are extracted by our system. Moreover, the vulnerable applications are reverted back to their original "clean" state at the end of each day. All the collected data is stored in a database hosted by the `manager` machine, which is also responsible to run the subsequent analysis.

We configured each virtual machine to prevent attackers from using the honeypot as stepping stone to run attacks and propagate over the network. For this end, we run all services as non privileged user and keep each of our honeypots up to date with software and security patches. Additionally, we drop all outgoing connection in order to prevent attackers to use our system to perform attacks or send spam messages. We also mitigate the problem of hosting malicious content by reverting virtual machine back to its clean state on a regular basis.

3.2 Extraction of Candidate Indicators

The second component of our system is in charge of preprocessing the collected data to automatically extracts the URLs of the components remotely included in the attackers' files, and to store them along with some additional information in our database. This requires our system to analyze each HTML file and collect the addresses of all the external resources.

In addition to the URL, we store the *base date*, which is the date when the URL was first seen in our honeypot, and the *last date* in which we observed it. Moreover, our system periodically probes on a daily basis each URL to verify if it returns a valid response. If it does not encounter any error, it updates the *last good response date* in the database. Finally, we also store the information of how many times a URL is included in the uploaded components from the base date to the last date.

While our technique can be applied to any resource type (e.g., JPEG images), in this paper we focus in particular

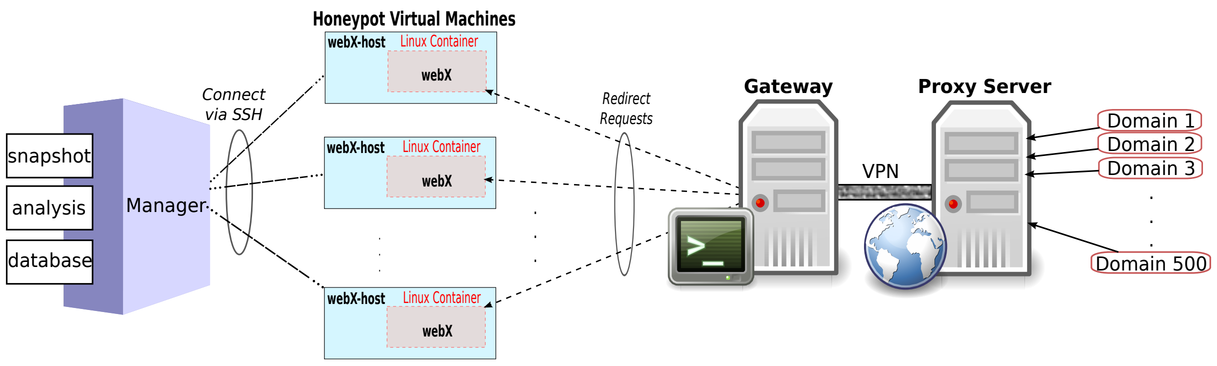

Figure 3: Overview of the Honeypot Infrastructure

on JavaScript files. In particular, since we extract the JavaScript URLs from the uploaded files after an actual attack is performed, one would probably expect that the vast majority of these scripts would contain malicious code. However, a manual inspection reveals that it is quite common for an attacker to include simple scripts that implement simple visual effects or common JavaScript libraries in their code. Unfortunately, this makes the identification of indicators of compromise much more complex. In fact, considering the fact that many of the scripts are not malicious in nature but they might still be used for malicious intents, it is impossible to tell whether a certain URL is in fact a good indicator of compromise by just looking at the content of the JavaScript code.

For example, simple scripts designed to prevent the right click of the mouse, which are not malicious per se, are widely used by attackers to prevent users from inspecting the source code of an infected page. However, to be certain that one of these scripts can be used to indicate that an attack has been successfully performed against the website which includes it, we need to extend our analysis to inspect not just the script content per se, but also the *context* in which it is used and the other pages on the Web that import it.

3.3 Searching the Web for Indicators

As we shift our focus more on the web pages that include each candidate indicator, we need a way to search the World Wide Web for a particular snippet of source code. Unfortunately, common search engines do not provide such functionality. For example, Google provides a search operator, `intext:`, that lets users search for a term contained in the text of the document. However, this only includes the content of a page that is displayed to the user, and not its HTML tags. As a result, it is not possible to use popular search engines to search for JavaScript entries or for other file included in a HTML document. Therefore, we needed a more sophisticated tool that indexes the source code of each visited page. For this reason, our prototype applications uses *Meanpath* [17], a search engine that also captures and index HTML and JavaScript source codes. While Meanpath does not have the same coverage as other classical search engines such as Google or Bing, its archive of over 200 Million web sites can help us to identify the web pages that actually *include* external scripts. In our context, these scripts are the ones pointed by our candidate indicator URLs.

3.4 Features Extraction

For each candidate indicators, we extract five different groups of features:

- **Page Similarity**
  The vast majority of the attacks are largely automated, and therefore attackers tend to re-use the same template for each website they compromise. We capture this characteristic by looking at the similarities of the web pages that include a candidate indicator URL as an external resource. For this purpose, our system automatically queries Meanpath to find websites that include the candidate URLs and it then downloads the HTML source code of the first 100 results. We use a fuzzy hashing algorithm (*ssdeep* [13]) to compute the similarity of the content of each website and then group the similarity of each unique pairwise comparison in one of five categories: *low* (similarity below 0.25), medium-low (0.25 to 0.5 similarity), medium (0.5 to 0.75 similarity), high (0.75 to 0.97 similarity) and perfect match (higher than 0.97 similarity). For each class we count the number of web pages that falls in the corresponding range. So if the *high similarity count* of a candidate indicator is high, it means that our tool came across almost the same content over and over again in the top 100 websites that include that indicator. Likewise, if the *lowest similarity count* is high, it means that all the websites that include the candidate URL have almost nothing in common.

- **Maliciousness**
  Although the majority of the candidate indicators are not malicious, they are often included as an external resource inside malicious pages. Hence, we also compute the maliciousness of the top 100 web pages that include a certain candidate URL, as part of the features we use to distinguish a normal script from a indicator of compromise. To this end, we automatically scan each website using the VirusTotal API [7] and Google SafeBrowsing [6]. We then categorize websites into three categories according to their maliciousness level: *maybe malicious* if less than five AV detected as so, *likely malicious* if five to ten AV return a positive match, and *malicious* if it is identified as so by SafeBrowsing or when the positive matches are more than 10 out of the 60 available AVs. Finally, we use the

total number of websites in each category as features for clustering candidate indicators.

- **Anomalous Origin**
  We also observed that attackers sometimes use popular JavaScript libraries in their pages. However, instead of including them from their original domain, they host their own copy on other servers under their control.

  For instance, an attacker may include the very popular JQuery library (e.g., `jquery-1.11.3.min.js`) not from `jquery.com` but from a personal server located in Russia. This could be a suspicious behavior, and in fact we encountered many examples in which web pages that include popular JavaScript libraries from external domains were compromised. In particular, we observed two different phenomena. First, some attackers use popular library names to hide code that has nothing to do with the library itself. For instance, we found a `jquery.js` file that was used to disguise a modified version of the `ciz.js` script shown in Figure 2. In a different scenario, attackers use instead a copy of the original script, often obfuscating its content (possibly to hide small modifications or customizations of the code). While this feature alone is not sufficient to generate WIOCs, our experiment demonstrates a high correlation between these cases and compromised websites.

- **Component Popularity**
  As the popularity of the external component increases, it is less likely that it is associated only to malicious activities, and therefore that it is a good indicator of compromise. For instance, some scripts associated to the Facebook Software Development Kit (e.g., `connect.facebook.net/en_US/all.js`) can also be found in the remote components uploaded by the attackers on our honeypot. However, since the same script is used by millions of other websites, it is unlikely that it is used only for malicious intents. Even if this was the case, it would have probably already attracted the attention of the security community and therefore other protection mechanisms and blacklists would be sufficient to protect the end users. Therefore, in our system we use the total number of search results from Meanpath as a feature to filter out very popular URLs.

- **Security Forums**
  In addition of using Meanpath to retrieve the pages that include a certain resource, we also query Google to collect how many times the candidate indicator is *mentioned* on the Web. From these results we extract two separate features: the total number of search results, and how many of the top 10 results mention the candidate indicator together with certain security related keywords such as "hacked", "malware", "compromised", and "antivirus". This is used to capture online forum discussions or threat web pages maintained by antivirus companies – in which people discuss the role of certain JavaScript files or ask for more information after a piece of JavaScript has been detected in their websites.

3.5 Clustering

After we automatically extracted all the features for each candidate external URL component, we applied an unsupervised learning algorithm to separate different classes of components. The reason for not using a supervised classifier is that it would require a considerable effort to build a ground truth. In fact, verifying if a certain URL is a good WIOC can take a large amount of time also for a skilled manual analyst. On the contrary, we believe that the features of good and bad indicators would differ enough to be clearly separated by a clustering algorithm.

In particular, we are interested in differentiating three main cluster categories:

- **Good Indicators of Compromise**
  This category includes the components that are, to the best of our knowledge, used **only** by attackers when they compromise a web page or install a malicious one. Although in our experiments the page similarity was the most distinctive feature to detect good indicators, all features contributed to the identification of this category.

- **Invalid Indicators of Compomise**
  This category covers the opposite case, in which a certain component is used as part of attacks but also as part of benign pages. As expected, the most distinctive feature in this category is the popularity of the candidate URLs.

- **Undecided**
  This cluster category describes those components for which the available data was not sufficient to take a final decision. Therefore, the URLs which fall into this category cannot be labeled as either good or bad indicators, even after a manual inspection. In fact, some components are so rare that both Google and Meanpath return no results (even though the remote JavaScript is online and can be retrieved by our system). In other cases, only few of matches are found in the search engines. Even if they were all examples of compromised pages, it would still be too risky to classify the indicator with such a limited amount of information.

We conducted a number of experiments with different thresholds and finally obtained the best results by using the K-means algorithm with k equal to eight. Other values of k may provide equivalent results, as our goal in this phase is only to show that it is possible to clearly separate the different behaviors in distinct groups. With this setup, the clustering algorithm was able to clearly separate each behavior and group candidate indicators in clusters that only contained a certain type (valid, invalid, or undecided).

To verify the accuracy of our approach, we manually verified a number of random items picked from each cluster. Out of the eight clusters identified by our algorithm, one contained only bad indicators, five only good indicators (three mainly defacements and two mainly malicious pages), and two were associated to the undecided group. In the Experiment Section we report on the accuracy of our clustering approach when applied to categorize potential indicators extracted by our live honeypot.

3.6 Impact on End Users

To measure the impact of our technique, we collaborated with Trend Micro, a popular antivirus vendor, to estimate how many real users have interacted with our WIOCs. Using a cloud-based infrastructure, the vendor collects over 16 terabytes of data per day from 120 million client installations worldwide. We based our analysis on a subset of this data, based on a telemetry feed that collects information on the URLs that are accessed by users over HTTP(S) – using their browser or any other client.

Whenever one of the AV client visits a page that includes our web indicators of compromise, her browser sends an HTTP request to fetch the missing component and we can detect and log this action.

In an operational environment, we envision that our approach could be deployed in three ways. First, to generate a blacklist that a company can use to prevent users from visiting compromised web pages. Second, by combining the `Referer` HTTP header with the telemetry information, a security company can use our indicators of compromise to automatically discover, in real time, new URLs of infected pages. While we were not able to test this configuration in our experiments, we believe that this scenario would provide even greater advantages compared with other existing mechanisms to detect malicious web pages. Finally, our indicators could be used as seeds to quickly search for malicious or compromised pages on the web. It would be enough to query for the pages that include these components to build a list of candidate targets, which can then be visited with more sophisticated scanners or honeyclients solutions.

4. EXPERIMENTS

In this section, we explain the tests we conducted to evaluate our approach and the results of our experiments. We also discuss the impact of our solution by correlating our data with the telemetry information of Trend Micro.

4.1 Dataset

Over a period of four years, our honeypots collected over 133K unique URLs of remote components – either uploaded by the attackers as part of their pages or as modification of the honeypot pages themselves. Note that in this study we were not interested in distinguishing between attack types, nor in measuring the frequency of attacks, or time period between successive threats. The reader may refer to previous studies [4] for a detailed analysis of these characteristics.

Out of all the remote components, our analysis focused on 2765 unique JavaScript files. In average, each of them was re-used several times (an average of seven and a maximum of 202) as part of different, likely automated, attacks. However, more than half of the JavaScript URLs were observed only once – as confirmation that our honeypot also captured unique events probably performed manually by the attackers.

To test our system, we trained our feature extraction and validation routines on the data collected between January and April 2015. While older data was available in our database (and it was used to analyze long-lasting campaigns), some of the features used by our technique need to be computed in real-time. Therefore, we were forced to operate only on the attacks performed after we started our study of indicators of compromise. We then used the result of the clustering to classify the new URLs observed by our honeypot over a period of four months starting in mid-April. The results are presented in the following sections.

4.2 Model Training

In order to evaluate our work, we first used our clustering approach to divide the URLs in the training set in different categories. The dataset included 373 candidate indicators. The clustering was performed using `Weka` [8] – a common open source tool for machine learning and data mining tasks. After the clustering operation was completed, we manually inspected the content of each cluster to assign it with the correct label (i.e., good indicator, invalid indicator or undecided), as described in Section 3.5.

This phase allowed us to tag five clusters as valid web indicators of compromise, for a total of 12% of the total number of candidate indicators. However, the goal of the clustering was not to detect indicators, but instead to separate the features space and provide reference values for the next phase.

4.3 Results

Our live experiment was conducted over a period of four months. During this time, the honeypot continued to collect new URLs of external components and to pass them to our analysis framework. The analysis system collected the external information and then computed the individual features. Finally, it computed the distance between the features of each URL and the different clusters built during the training phase and it assigned the URL to the category of the closest cluster. So, if a new candidate indicator was close to a cluster marked as "Invalid Indicators", the URL would be considered invalid as well and discarded. If, instead, the closest cluster was flagged as "Good Indicators", then the candidate URL was considered valid. Table 1 shows the results of our classification.

As we already mentioned, the page similarity was the most distinctive feature, followed by the presence in security forums and by the number of hits in VirusTotal. Interestingly, most of the websites that include an indicator URL were not detected as malicious. However, even a single page flagged by VT in the set of hundred results can be a very distinctive factor once combined with the other features. On the other end of the spectrum, the component popularity feature was the one with the highest negative correlation.

With a considerable manual effort, we investigated each single case to understand if our system was correct with its classification and to look for possible false positives. As we better discuss in the next section along with a number of examples and case studies, we only found two false positive out of 303 analyzed URLs.

The first false positive is a very popular library provided by Google and used as external resource by many websites (including some defaced and some malicious ones). Unfortunately, some of these websites were duplicated in different domains (therefore with exactly the same content) and this caused an increase in the similarity rate which, inevitably, results in a false positive. The other false positive is a JavaScript file used for video and animated online ads (AdInterax). Although there were no results on Meanpath for this URL, it was often discussed on security forums by users who were afraid it was a malicious component.

Except for these two isolated cases, we were able to confirm that the web indicators of compromise extracted by

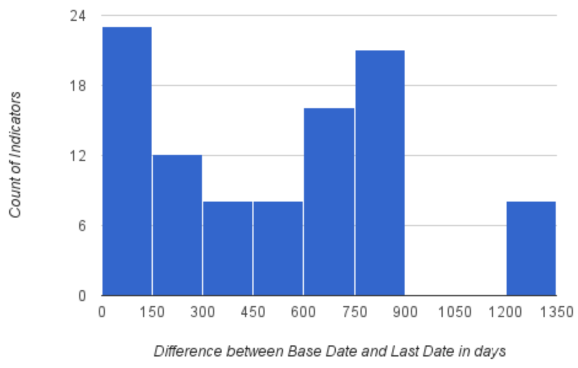

Figure 4: Differences in days between the first seen date and the last seen date of the Valid Indicators of Compromise

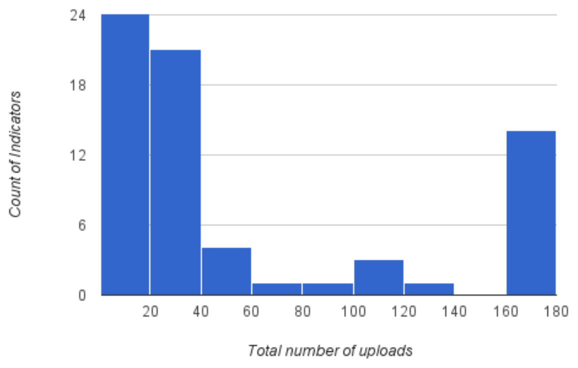

Figure 5: Total number of uploads of WIOCs

our tool were indeed constantly associated with malicious activities and did not appear in benign pages.

Almost 20% of those indicators were URLs of JavaScript component that we never observed before in our honeypot. Interestingly, the remaining 80% were instead components that were observed several times by our honeypot during the previous years. In average, these indicators of compromise were first observed 20 months before our experiment, with a maximum of 44 and a minimum of 5 months. Figure 4 shows the difference between the first seen date and the last seen date for each valid indicators of compromise identified by our tool. The graph shows that the average lifetime of these JavaScript components is very high. This is likely a consequence of the fact that the majority of these scripts are not malicious per se, and therefore the identification of these components is very difficult by using existing techniques. As a result, some stay unnoticed for months or even years. Figure 5 shows instead the total number of times each indicators was used in attacks against our honeypot.

Category	Number of Items
Invalid Indicator	22
Valid Indicator of Compromise	96
Not-enough-data	185

Table 1: Clustering results for the detection set

4.4 Antivirus Telemetry

To assess the impact of our indicators of web compromise in a real deployment, we asked Trend Micro to cross-check our results. For this purpose, we sent them the set of our indicators and asked them to match the URLs against their web telemetry dataset collected in the same time period.

Overall, over 90% of our web indicators were previously unknown to the vendor and were considered benign by their internal intelligence database. Moreover, the vast majority of pages that included those components were not detected as infected by any AV tool used by VirusTotal. In total, only 5.3% of the webpages including an indicator were detected by at least for one antivirus products. Once more, this confirms our initial hypothesis that existing automated scanners only flag a page when there is an clear evidence of malicious activity and fail to detect more subtle signs of a possible compromise.

Interestingly, some of our indicators were hosted on domains for which Trend Micro observed only hits toward the URL of that particular Javascript and nothing else in their telemetry dataset, as if that component was the only item hosted on the same domain.

In average, each indicators was requested by 47 different users per day. However, the distribution was quite unbalanced, with one case that was never observed in the wild and one case that had more than 800 visits per day.

However, we believe that an automated system that could prevent each day thousands of users from visiting malicious or compromised websites – not caught by any other detection systems and by the blacklists already in use at the antivirus vendor – is a promising result that shows the value of our idea and the need for more research in the area of web indicators of compromise.

5. CASE STUDIES

The 96 indicators extracted over four months by our system belong to different categories. For instance, 26% of the JavaScript components were used to implement simple visual effects (such as moving or teletype text, or a snow effect over the page) commonly used in defacement campaigns. Other were used as part of phishing pages, or to redirect the visitors towards other websites.

In this section we discuss in more details some of these cases, to try to show different scenarios and different types of malicious activity detected by our system.

Affiliate Programs

Attackers often try to monetize the traffic towards web sites they compromise, for example by joining affiliate programs or including Adwares banners. In our evaluation, we identified several cases of JavaScript dedicated to this purpose.

For example, one of the indicators found by our system[1] is part of a large affiliate network called VisAdd [25]. The script acts as a traffic redirection system – i.e., a sort of proxy that brings the users of the exploited web page to the affiliate web site. In this way, the miscreant get rewarded for each visitor of the site she was able to compromise. Interestingly, VisAdd also makes use of a malicious software called `A.Visadd.com`[2] to bring additional visitots into its network. By correlating this indicators with Trend Micro's web telemetry dataset, we confirmed that an average of 620 users per day were affected by sites including this JavaScript. In another example, a malicious browser plugin – in form of an Internet Explorer Browser Helper Objects (BHOs) was loaded by a JavaScript file at run-time in order to hijacking the browser's user's session. We observed the same Javascript embedded in a multitude of defaced web sites.

In both cases, it is interesting to observe that cyber criminals used a combination of client-side approaches – like malware and BHOs – and server-side compromised websites to redirect legitimate traffic to affiliate programs. We recorded an average of 594 visits per day to this indicator.

Since these JavaScript files were quite popular in the antivirus dataset, it would be possible to use them to track the activity and evolution of this large campaign, whose list of compromised websites is still increasing at the time this paper was written.

Web Shells

A second class of indicators that we automatically identified as malicious are related to web shells, which are often deployed by the attackers and hidden in compromised web sites. Their goal is to allow the attackers to easily control the compromised machine and execute arbitrary commands by using an intuitive web interface. We found several cases in which, despite the attacker protected the access to the web shell via password, our system was able to automatically flag these cases because they embedded a malicious indicator.

As already described in the example of Section 3, we also discovered a small JavaScript responsible to send a signal back to the attackers every time someone visited one of the installed web shells. Some of these (e.g. `http://r57shell.net/404/ittir.js`) automatically leak the information of the visit to the attacker's dropzone, e.g. by embedding the request in the form of a image retrieval – in a technique similar to a CSRF. The inclusion of this URL on compromised websites is a clear indicator of an organized network in which the attackers monitor the websites they infected as well as the ones infected by other groups that reuse the same web shells.

Code Repositories

In our dataset, we found a considerable amount of indicators of compromise hosted in public code repositories, such as Google Code. Even though it is not unexpected for attackers to use such repositories, it was surprising to observe how long these indicators can survive before they get noticed and taken down by the maintainer or the security community.

[1]`http://4x3zy4ql-l8bu4n1j.netdna-ssl.com/res/helper.min.js`

[2]`http://malwaretips.com/blogs/a-visadd-com-virus-removal/`

For example, we found two indicators hosted on Google Drive and eight on Google Code. Interestingly, one of them was online for at least 729 consecutive days before it was finally suspended by Google and just in a single month MeanPath reported dozens of defaced websites and drive-by pages using this script.

During the manual verification of this case, we realized that most of the web pages that include WIOCs look almost identical. Furthermore, even a quick search on Google returned many forums in which people complained about how their website got hacked as well as scan results from popular sandboxes. This case confirms that our features provides an accurate characterization of indicators.

Mailers

Another use of compromised websites is to turn them into a spam mailing server to send large amounts of fraudulent emails. Instead of switching between different providers or relying on botnet-infected machines, attackers often search for non-blacklisted vulnerable websites to use as part of their infrastructure.

In our experiments, our system reported two indicators of compromise corresponding to two copies of the *JQuery* library, hosted respectively on Google and Tumblr. The majority of websites using these libraries (e.g., `http://www.senzadistanza.it/` and `http://www.hprgroup.biz/`) contained pages injected with a popular mailer called Pro Mailer v2, which is often shared among hackers in underground forums. Because of the popular domains used by these indicators, and because of the fact that they were unmodified copy of popular libraries, these files very likely misclassified as benign by both automated scanners and manual inspection. Therefore, we believe this particular example is very important, since it emphasize the fact that even the most harmless and legitimate URLs can be valid indicators of compromise.

Phishing

Phishing pages are commonly found in our dataset, as attackers try to install copy of popular websites in our honeypot after they compromise one of our web applications. As a last example, we want to discuss two borderline cases we found in our results.

In these cases, the attackers installed phishing pages for the Webmail portal of two popular websites, AOL and Yahoo. Instead of simply uploading the entire content of the site on our honeypot, they re-used the original AOL and Yahoo JavaScript files hosted on their respective provider's domain. Since the components were clearly also used by benign websites, these URLs were misconceived as benign and classified as false positive during manual verification. However, a quick search for both examples returned many websites including these scripts that were clearly not related to AOL or Yahoo (e.g., `http://www.ucylojistik.com/` for AOL and `http://fernandanunes.com/` for Yahoo), and that turned out to be all compromised to host phishing pages.

We decided to discuss this case as it demonstrates how a benign URL can be used to leverage phishing pages. Even though both URLs also serve for benign purposes, they are also excellent indicators of compromise when they are observed on web sites registered outside of their original domain or autonomous system. In other words, any time users requested these JavaScript files while visiting a page that was not on the AOL/Yahoo domain, then they were victims of phishing. However, since these components are also used by their legitimate service, we did not count their hits in the AV dataset in our report.

6. LIMITATIONS

Since our method relies on the fact that attackers remotely include resources in their pages, it is possible to evade our technique by using a different deployment strategy. For example, attackers could include their code inline rather than importing the indicator's URL from an external source, or they could generate a different URL for each target. Even though these techniques would effectively undermine our ability to extract valid indicators of compromise, these changes would also result in a loss of flexibility (e.g., the attacker would not be able to change at once the code used in hundreds of compromised pages) or in an increased complexity in the deployment of the code.

In our current implementation, our system relies on a clustering phase to separate the good from the bad indicators. While we did not need to repeat this training during our experiments, it may be helpful to update the clustering at least once a year – to account for possible changes in the features distribution. For example, it is possible that security forums become more popular in the future, or that the results returned by Meanpath increase (or decrease) over time.

Finally, while this paper is the first to introduce the concept of web indicators of compromise, we expect more researchers to focus on this interesting problem and to propose more sophisticated and more robust solutions to extract WIOCs in the future.

7. RELATED WORK

Previous work on detecting compromised websites includes anomaly detection, content monitoring, and custom feature extraction techniques. Our work deals with the identification of indicator of web compromise. However, since there is no similar studies on this topic, this section includes previous works focusing on detecting and analyzing malicious URLs.

To the best of our knowledge, the work most closely related to ours is the study to automatically detect malicious web pages conducted by Invernizzi et al. [11]. The authors start with a set of URLs that are already known to be malicious and then make a guided search using a web crawler to find other pages that share certain similarities with the initial set of URLs. For this purpose, they use Wepawet, Google Safe Browsing, and their custom fake AV detector to check if the guided search results are successful in detecting malicious websites. In our work, we gathered URLs from remote components uploaded to our honeypot during real attacks. Instead of analyzing the maliciousness of web pages, we analyze the features of the URLs that are frequently used by the attackers. These URLs may or may not be malicious, but they still indicate a compromised or malicious page.

Most of previous work on maliciousness of URLs includes URL classification by using machine learning algorithms. For example, Ma et al. [14, 15] used lexical and host-based features (IP address, domain name, etc.) to classify malicious and benign URLs. Their aim is to differ malicious URLs from the benign ones by training their models with the data they gathered by querying blacklists for the malicious URLs and using Yahoo's random URL selector. Another study presented by Soska et al. [22] tries to predict if a benign web page will turn malicious in the future. The authors use traffic statistics, file system structure, and the web page content as features and they use the data gathered from a number of blacklist to build their ground truth. Zhao et al. [27] proposes two cost-sensitive learning algorithms in order to detect malicious URLs and they analyze their theoretical performance.

Provos et al. [21] described several server-side and client-side exploitation techniques which are used for the distribution of malware. The authors instrumented Internet Explorer in a virtual machine to analyze anomalies when a malicious binary is downloaded while visiting a defaced or malicious web site. They then visit a large number of URLs and look for suspicious elements such as an iFrame pointing to a host known to be malicious. If there is no such element, they further investigate the interpreted JavaScript in each page. Webcop [23] aims at finding the relations between malicious URLs and malware distribution sites. The authors used the data from commercial Anti-Malware clients to decide if a URL is hosting malicious code. Then they used a web graph constructed by a commercial search engine crawler to find the malicious URLs directly linked to malware distribution sites via hyperlinks.

Honeymonkey [26] is a client-side honeypot, which consists of several vulnerable web browsers in virtual machines of different patch levels. The honeypot can be used to scan web pages to detect the malicious ones. Moreover, for the malicious URLs, a redirection analysis is performed to get all the URLs involved in the exploit. Finally, the same malicious URLs are analyzed on fully patched virtual machines to see if the attacks are still successful. This process also helps to find zero-day exploits. On the contrary, in our work, we use server side honeypots and instead of scanning the web, and we use real incoming attacks as seeds for our analysis.

Nikiforakis et al. [19] draw attention to the fact that a website can be compromised if the remotely included libraries are changed by the owner of the remote server. They investigate the trust relationship of websites with their JavaScript library providers. They crawl the most popular websites to collect millions of URLs and measure the quality of the JavaScript providers based on various features including hosts availability, cookies, anti-XSS and anti-clickjacking protocols, and the SSL/TLS implementation. They then manually assign a weight for each feature and evaluate their metrics, showing that even the highly popular websites can get compromised through their external library providers. While they measure the quality of JavaScript providers that legitimate websites are using, we use JavaScript libraries that are not hosted in its original domain to identify defacement of web pages.

Bartoli et al. [1] presents a compromised website detection service, called Goldrake, based on anomaly detection. The authors monitored websites and analyzed various elements including contents of the web page, frequency of items, typical defacement signatures (e.g. common phrases, black background) without requiring any kind of assistance from the monitored web page. Although they managed to keep false positive rate low, their work is hard to extent in order to find defaced web applications in the whole Internet, due to the fact that they have to continuously monitor the websites.

Evil Searching [18] takes a different approach to the detection of compromised websites by analyzing the search queries that attackers use to find the vulnerable websites to deface. The authors goal is to analyze the methods which attackers use to detect possible targets in the wild. Their aim is to find phrases, called "evil searches", that can be used in phishing attacks. In our experiments, we instead use search engines to find "evil scripts" that can be used in many types of attacks including phishing.

Finally, several works focus on the detection of defaced pages [2, 3]. While not strictly related to our objective and methodology, our system can be adapted to detect defaced websites by identifying their possible use of certain indicators of compromise.

8. CONCLUSIONS

In this paper we present a novel idea to use the information collected by a high interaction honeypot of vulnerable web applications. Our approach starts from the observation that attackers often include remote JavaScript components in the pages they modify or they upload after a successful attack. These components are rarely malicious per se, but their URLs can still be used to precisely pinpoint the activity of a certain group and therefore the fact that a web page has been compromised. For this reason, in this paper we propose a technique to collect these components, validate them using a number of features, and finally use them as *Web Indicators of Compromise* (WIOCs).

We implemented our system and run it on our premises for several months. After an unsupervised training phase, we tested for four months its ability to automatically extract valid WIOCs. The results showed that these indicators cover several types of malicious activities, from phishing sites to defacements, from web shells to affiliate programs. Moreover, most of these components have been used for a long time by the attackers, who hosted them on public websites – since their apparently harmless content was not detected as suspicious by any of the existing tools and techniques.

We believe that more research is needed in this area, to help the security community to reliably extract and share this new type of indicators of compromise.

ACKNOWLEDGMENTS

The research was partially funded by the Principality of Monaco.

9. REFERENCES

[1] Bartoli, A., Davanzo, G., and Medvet, E. A framework for large-scale detection of web site defacements. *ACM Transactions on Internet Technology (TOIT) 10*, 3 (2010), 10.

[2] Borgolte, K., Kruegel, C., and Vigna, G. Delta: Automatic identification of unknown web-based infection campaigns. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security* (New York, NY, USA, 2013), CCS '13, ACM, pp. 109–120.

[3] Borgolte, K., Kruegel, C., and Vigna, G. Meerkat: Detecting website defacements through image-based object recognition. In *Proceedings of the 24th USENIX Conference on Security Symposium* (Berkeley, CA, USA, 2015), SEC'15, USENIX Association, pp. 595–610.

[4] Canali, D., and Balzarotti, D. Behind the scenes of online attacks: an analysis of exploitation behaviors on the web. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS)* (January 2013), NDSS 13.

[5] Canali, D., Cova, M., Vigna, G., and Kruegel, C. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th international conference on World wide web* (2011), ACM, pp. 197–206.

[6] Google Inc. Safe Browsing API. `https://developers.google.com/safe%2Dbrowsing/`, 2015.

[7] Google Inc. VirusTotal. `https://www.virustotal.com`, 2015.

[8] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The weka data mining software: An update. *SIGKDD Explor. Newsl. 11*, 1 (Nov. 2009), 10–18.

[9] Hun-Ya Lock, A. K. Using IOC (Indicators of Compromise) in Malware Forensics. Tech. rep., SANS, 2013.

[10] Ikinci, A., Holz, T., and Freiling, F. C. Monkey-spider: Detecting malicious websites with low-interaction honeyclients. In *Sicherheit* (2008), vol. 8, pp. 407–421.

[11] Invernizzi, L., Benvenuti, S., Cova, M., Comparetti, P. M., Kruegel, C., and Vigna, G. Evilseed: A guided approach to finding malicious web pages. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2012), SP '12, IEEE Computer Society, pp. 428–442.

[12] John, J. P., Yu, F., Xie, Y., Krishnamurthy, A., and Abadi, M. Heat-seeking honeypots: design and experience. In *Proceedings of the 20th international conference on World wide web* (2011), ACM, pp. 207–216.

[13] Kornblum, J. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation 3, Supplement*, 0 (2006), 91 – 97.

[14] Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. Beyond blacklists: Learning to detect malicious web sites from suspicious urls. In *Proceedingsof theSIGKDD Conference. Paris,France* (2009).

[15] Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. Learning to detect malicious urls. *ACM Trans. Intell. Syst. Technol. 2*, 3 (May 2011), 30:1–30:24.

[16] Mandiant. OpenIOC – An Open Framework for Sharing Threat Intelligence. `http://www.openioc.org`, 2015.

[17] Meanpath. Meanpath Web Search API. `https://meanpath.com/`, 2015.

[18] Moore, T., and Clayton, R. Evil searching: Compromise and recompromise of internet hosts for phishing. In *Financial Cryptography and Data Security*. Springer, 2009, pp. 256–272.

[19] Nikiforakis, N., Invernizzi, L., Kapravelos, A., Van Acker, S., Joosen, W., Kruegel, C., Piessens, F., and Vigna, G. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 736–747.

[20] PhishTank. PhishTank Website. `http://www.phishtank.com/`, 2015.

[21] Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N., et al. The ghost in the browser analysis of web-based malware. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets* (2007), pp. 4–4.

[22] Soska, K., and Christin, N. Automatically detecting vulnerable websites before they turn malicious. In *Proc. USENIX Security* (2014).

[23] Stokes, J. W., Andersen, R., Seifert, C., and Chellapilla, K. Webcop: Locating neighborhoods of malware on the web. In *Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats:*

Botnets, Spyware, Worms, and More (Berkeley, CA, USA, 2010), LEET'10, USENIX Association, pp. 5–5.

[24] Stringhini, G., Kruegel, C., and Vigna, G. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 133–144.

[25] VisAdd. Advertisement Solution. `http://visadd.com`.

[26] Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., and King, S. Automated web patrol with strider honeymonkeys. In *Proceedings of the 2006 Network and Distributed System Security Symposium* (2006), pp. 35–49.

[27] Zhao, P., and Hoi, S. C. Cost-sensitive online active learning with application to malicious url detection. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2013), KDD '13, ACM, pp. 919–927.