Scanner Hunter: Understanding HTTP Scanning Traffic

Guowu Xie, Huy Hang
Department of Computer Science
University of California, Riverside
Riverside, California, United States
{xieg, hangh}@cs.ucr.edu

Michalis Faloutsos
Department of Computer Science
University of New Mexico, Albuquerque
Albuquerque, New Mexico, United States
michalis@cs.unm.edu

ABSTRACT

This paper focuses on detecting and studying HTTP scanners, which are malicious entities that explore a website selectively for "opportunities" that can potentially be used for subsequent intrusion attempts. Interestingly, there is practically no prior work on the detection of these entities, which are different from web crawlers or machines performing network-level reconnaissance activities such as port scanning. Detecting HTTP scanners is challenging as they are stealthy and often only probe a few key places on a website, so finding them is a needle-in-the-haystack problem. At the same time, they pose serious risk because they perform the first, exploratory step to provide the seed information that may allow hackers to compromise a website. Our work makes two main contributions. First, we propose Scanner Hunter, arguably the first method to detect HTTP scanners efficiently. The novelty and success of the method lies in the use of community structure, in an appropriately constructed bipartite graph, in order to expose groups of HTTP scanners. The rationale is that the aggregated behavior makes identifying groups of scanners easier than attempting to profile and label IP addresses individually. Scanner Hunter achieves an impressive 96.5% detection precision, which is roughly twice as high as the precision of the Machine Learning-based methods that we use as reference. Second, we provide an extensive study of HTTP scanners in an effort to understand: (a) their spatial and temporal properties, (b) the techniques and tools used by the scanners, and (c) the types of resources they are looking for, which can provides hints as to what the subsequent penetration attempt may target. We use six months worth of web traffic logs collected in 2012 from a University campus, the websites hosted by which received over 1.9 billion requests from 12.8 million IPs. We found that the number of HTTP scanners is non-trivial with roughly 4,000 IPs engaging in this type of activity per week. Our work will hopefully raise the awareness of the community regarding this problem while at the same time provide a promising detection technique that can provide the basis for mitigating the risk posed by HTTP scanners.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASIA CCS'14, June 4–6, 2014, Kyoto, Japan.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2800-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2590296.2590297.

Categories and Subject Descriptors

J.0 [**Computer Applications**]: General

Keywords

HTTP scanning, HTTP scanner, scanning traffic, web security, detection, vulnerability

1. INTRODUCTION

HTTP scanning is a fascinating and lesser known activity with which hackers aim to discover the security weaknesses of websites as a precursor to an actual penetration attempt. Specifically, in HTTP scanning, malicious entities probe a website for particular resources that seem promising for exploits or contain sensitive information that could reveal its structure and its underlying technology. These resources can be files with security-sensitive information, files containing passwords, zipped archives of the entire websites, or web interfaces for authorized users such as *phpmyadmin.php*.

The amount of requests sent by scanners is small compared to the traffic the websites received because they only probe for very specific files. At the same time, the risk that scanners pose is high. For example, an intrusion attempt that seems to have been enabled by HTTP scanning was carried out by the hacktivist group NullCrew [1]. The group broke into a Department of Homeland Security website after identifying files that should have been invisible to the general public.

The motivation for our work is the little attention that the problem has received so far and the ever-increasing importance of website security. First, there is currently no existing solution for this problem. Second, scanning is a first exploratory step towards compromising a website. Third, having a clear understanding of scanning behaviors could help administrators secure their websites and provide assistance during forensics analysis. Fourth, understanding what scanners are looking for can reveal what are the preferred vulnerabilities. Finally, understanding the spatiotemporal dynamics of these scanners can provide insights into the ecosystem that enables them.

The focus of this work is on the problem of HTTP scanning and our goal is to detect and understand the behavior of HTTP scanners. Ideally, a solution to this problem takes in as input web traffic logs that record all HTTP requests sent by external IPs and produces as the desired output a list of IPs that potentially engaged in HTTP scanning activities. For such a solution to be useful, it should be able to accomplish its task with minimal false alarms. This is a challenging problem for the following reasons: (a) scanners usually generate few requests, (b) it is not easy to distinguish between a scanner and a legitimate web user, and (c) the number of scanners is significantly smaller than the number of legitimate users, leading to a needle-in-the-haystack problem. We substantiate and quantify all these claims in later sections.

Surprisingly, this problem has attracted very little attention from the research community. In fact, ours is arguably the first work focusing on this problem, as we discuss in section 5. At the same time, it is equally important to clarify what we are **not doing**. Detecting HTTP scanners is different from identifying: (a) web crawlers, which collect information from the entire website for intelligence purposes (for instance, a retailer may want to know about the pricing offered by a competitor to adjust the prices of its own products), (b) network-level scanners such as IP or port scanners that try to discover vulnerable services running on specific ports of machines in the network, and (c) web application penetration attempts, which is the step following a successful HTTP scan, In section 5, we present related work in these areas, and discuss how the problems they aim to solve are different from the one we focus on.

As our main contribution, we present Scanner Hunter, arguably the most effective method to detect HTTP scanners, and study scanners and their behaviors over six months. The key novelty of Scanner Hunter is that it is a graph-based approach that detects HTTP scanners by focusing on the collective actions of scanners, who search for similar resources. In addition, we provide an extensive study of HTTP scanning activities over a six-month period to understand their scanning patterns and their spatiotemporal behavior. We use real web-logs collected from a University campus from March 2012 to September 2012 with over 1.9 billion HTTP requests from 12.8 million external IPs.

Note that, to enable further research in this novel direction, we will share with the research community the access logs from the HTTP scanners we have detected (with the IP addresses appropriately anonymized). Interested researchers may contact us for more information.

1. Scanner Hunter detects scanners with 96.5% precision. Our approach identifies malicious communities of IPs in an appropriately constructed bipartite graph that captures the interactions between two sets of entities: the IPs that generate HTTP requests and the resources[1] they have requested. Using a soft co-clustering technique, our approach identifies groups of IPs with similar requests. It then uses a labeling step that separates groups that engage in scanning from those of benign users. The use of clustering techniques on the graph makes the detection more accurate: scanning behaviors are easier to recognize for a group of IPs than for individual ones, especially since scanners are often stealthy. As a proof of concept, we conducted experiments with simple heuristics and standard Machine Learning techniques and observe that they only achieved 19.1% and 54.3% precision respectively, compared to 96.5% for Scanner Hunter. Even though a false positive rate of 3-4% may seem high, we want to stress that detecting scanners from millions of users at such a high precision is not trivial. We provide more details about the experiments in section 3

2. HTTP scanning is widespread: about 4,000 scanners per week for a medium-sized University network. We conducted an extensive study on scanning activities for a period of six months to understand: (a) their spatial and temporal properties, (b) their mechanisms and the effort to evade detection, and (c) what they are looking for. We highlight the most interesting observations.

a. Intensity: Roughly 4,000 unique IPs scan at least one website hosted by the University every week. At the same time, 80% of the IPs that appear every week have never been seen before.

b. Spatial distribution: Scanners are widely distributed across the IP space, which suggests that the IP prefix-based blocking of scanning activities would require a large and fine-grained filter set.

c. Complacency: Scanners do not seem afraid of getting caught, as they appear complacent and sloppy. For example, more than one third of the scanners used an unusual User-Agent in their requests and this User-Agent is `mozilla/4.0`, which, as we will show in section 4, is not a legitimate User-Agent.

d. Categories of scanners: We identify four major categories of scanners based on the resources they look for: (i) user registration and login pages, (ii) website management interfaces, (iii) pages with potential vulnerabilities that may allow activities like remote code execution, and (iv) compressed web archives such as `wwwroot.zip`, which are the products of website backup activities and should not be publicly visible.

Scope and intended use. Scanner Hunter is not meant to be: (i) a tool that can perform the same duties of an Intrusion Detection System, or (ii) a real-time solution for scanner detection.

Regarding (i), Scanner Hunter is an intelligence-gathering and advisory tool that can give the administrator a view into the malicious entities that are scoping out the website and the kinds of vulnerabilities that they are looking for. This knowledge can guide the administrator to take protective measures, as we discuss at the end of section 4.3.

Regarding (ii), real-time detection of scanners is inherently difficult. On the one hand, the labeling of an individual IP, especially one with few HTTP requests, is not easy given the wide variations in the behaviors of scanners as well as in the behaviors of legitimate users themselves. On the other hand, if we resort to traffic aggregation to obtain a more complete picture of behavior, like our approach does, we end up having to collect data over a sufficiently long window of observation. We discuss and substantiate this issue throughout the paper.

2. METHODOLOGY

In this section, we first provide some definitions and background and then present our method, Scanner Hunter.

2.1 Definitions and background

There are several types of **HTTP requests**, as one request can be a `POST`, a `GET`, a `HEAD`, or others. In this work, we are only interested in `POST`s and `GET`s for simplicity. A `POST` request sends information to the target website (e.g. filling an online form) and a `GET` request retrieves informa-

[1]A resource in this context is a file of any type and the full path leading to it. For example, `/phpmyadmin/scripts/inc.php`.

tion from the website. When a remote web server receives a HTTP request, the server will process it and return a response along with a response code that indicates whether the request succeeded.

A failed HTTP request is indicated by response code `40X`, where `X` is a single-digit number. For instance, the response code `404` indicates that the resource does not exist and `403` says that the server is refusing to return the requested resource [2]. Here, we refer to requests that trigger a response with code `40X` as failed HTTP requests, and the others, successful HTTP requests.

A User-Agent is a string contained in the `User-Agent` field of an HTTP request that provides some description about the application that generates the request. A typical value is the name of a web browser with version information.

The resource requested by an HTTP request (`example.com/inc/login.php?user=admin`) usually has four separate components: a domain name (`example.com`), a directory name (`/inc/`) that contains the file sought by an HTTP request, the name of the file (`login.php`), and a set of parameters (`user=admin`) that may alter the server's response. We define a **Re-Path**, which stands for Resource-Path, to be the portion of an URL that contains both the directory path and the name of the file. For example, the Re-Path in the example is `/inc/login.php`.

Although we have already defined HTTP scanners, here we want to highlight some interesting behaviors that we will exploit in detecting them. Note that these behaviors on their own are not sufficient without the additional techniques of our method, as we will show in section 3. Recall that scanners explore a website in a targeted fashion and their probing is a preparation for a subsequent exploitation. The behavior of scanners exhibits the following characteristics:

- Because scanners blindly look for resources of interest, many of their HTTP requests will fail.
- They do not request or even download embedded objects that are unlikely to lead to exploits like JPEG images.

Benign crawlers are automated programs chiefly used by search engines to index the Web. They are different from HTTP scanners in that they follow existing links instead of searching for resources that may not exist, b) they state what they are in the User-Agent field (e.g. Googlebot, Yahoo! Slurp, etc), c) the IP of the machine running the crawler program is usually registered by the company it belongs to, and d) they would (optionally) request the `robots.txt` file from the website that contains instructions as to where they're allowed to crawl.

A **stealthy crawler** is similar to a benign crawler in behavior except that it customizes its User-Agent field to make it look like their HTTP requests come from a regular web browser. There already exists work devoted to the detection of stealthy crawlers [3], and it is not a focus for our work.

2.2 Methodology

To identify HTTP scanners, there are four main steps, which Scanner Hunter executes as follows.

1. **Preprocessing**: Scanner Hunter processes the web traffic logs to filter out requests unlikely to be from HTTP scanners.
2. **Bipartite graph construction**: Scanner Hunter captures the suspicious HTTP requests in a bipartite graph where one set of vertices correspond to IPs and the other to the Re-Paths they requested.
3. **Co-clustering**: Scanner Hunter uses a soft co-clustering algorithm to group the IPs into communities where IPs in the same community have similar behaviors.
4. **Community labeling**: Scanner Hunter inspects each community output from previous step and label each of them as a community of users or scanners.

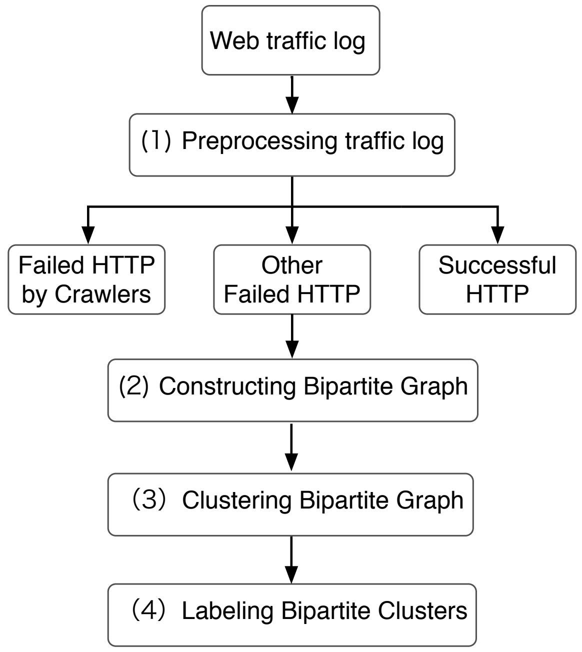

Web traffic log
(1) Preprocessing traffic log
Failed HTTP by Crawlers
Other Failed HTTP
Successful HTTP
(2) Constructing Bipartite Graph
(3) Clustering Bipartite Graph
(4) Labeling Bipartite Clusters

Figure 1: Scanner Hunter's main operations.

We now describe the operation and rationale of each step in detail. At the end, we will also explain why each step is necessary for achieving high precision in identifying scanners.

1. Preprocessing: Given web traffic logs as inputs, which may be collected by passively monitoring HTTP traffic or collected from web server logs (like Apache log files), Scanner Hunter reads in all HTTP requests from the logs and separate them into three different categories: successful HTTP requests, failed HTTP requests generated by web crawlers, and other failed HTTP requests. As their names suggest, the first category contains only successful HTTP requests, the second failed HTTP requests generated by known and benign web crawlers, and the third all other failed HTTP requests.

There are several reasons for including this step. First, popular websites receive an extremely high number of HTTP requests each day and processing all of them would consume a lot of time and resources. Second, we have seen from manual inspection that most scanners produce a large number of failed requests, which contain enough information for Scanner Hunter to identify the scanners.

In addition, we filter out requests that were followed by server-side errors and failed requests that may come from benign crawlers because they may generate them in the process of following all possible links whether they may be broken or not. The same crawlers may frequently re-check the web pages they have indexed before even after those same web pages have been removed. This means that crawlers exhibit similar behaviors to those of scanners and by removing the requests generated by legitimate crawlers, Scanner Hunter avoids mistaking the activities of benign crawlers for those of scanners and achieves higher accuracy. To this end, we first attempt to detect the presence of well known and benign web crawlers in the data according to the properties we mentioned in Section 2.1. Although there exist methods to detect web crawlers, as it is a well-studied problem [3], here we find it sufficient to label an IP as a benign crawler if two conditions are met:

C1: The User-Agent fields of its requests indicate that the application that generated the requests is a benign crawler (Googlebot, Yahoo! Slurp, etc.) and the IP has requested the file `robots.txt`.

C2: The information collected from reverse-DNS lookup of the IP indicates that the IP belongs to the organizations that User-Agent claims it is.

2. Bipartite graph construction: In this step, Scanner Hunter constructs a weighted undirected bipartite graph $G = \langle I, P, E\rangle$ where I is the set of IPs that produced the failed HTTP requests, P the set of Re-Paths that the IPs requested unsuccessfully, and E the set of edges. An edge $e \in E$ between IP $i \in I$ and Re-Path $p \in P$ if and only if IP i requested the Re-Path p but failed to access it during the monitoring interval. We assigned a uniform weight of 1 to all edges in the graph. Note that we did experiment with several other weight-assigning schemes (such as one where the weight of an edge is the number of times the IP requested the Re-Path) but we observed that the uniform weight allows Scanner Hunter to achieve the highest accuracy.

3. Co-clustering: Given the bipartite graph G generated in the previous step, Scanner Hunter uses a soft clustering algorithm[2] to partition G into a set of communities $C = \{c_1, c_2, ... c_n\}$. We adapted and used an algorithm inspired by Phantom [4], which we tailored to our problem. Each community c_i is a bipartite subgraph of the original G. Therefore, $c_i = \langle I_i, P_i, E_i\rangle$ where I_i is the set of all IPs in c_i, P_i the set of all Re-Paths in c_i, and E_i the set of all edges exclusively between I_i and P_i. We expect the output communities (used interchangeably with bipartite subgraphs from now on) from the soft clustering algorithm to have the following properties:

- The IPs (I_i) in each community are well-connected to each other through the Re-Paths (P_i) they requested.
- The soft-clustering approach may put an IP in more than one community because the IP may request many different Re-Paths.

The input to our adapted co-clustering algorithm is the bipartite graph G, represented by a weighted adjacency matrix M. In the beginning, the algorithm considers the graph as a single community that consists of all of the IPs and Re-Paths. For each community c:

1) The algorithm leverages a Singular Value Decomposition (SVD) technique to figure out how to best "cut" c into two child communities c_a and c_b so that a vertex in c_a will be more strongly connected to the other vertices in c_a than with those in c_b.

2) The algorithm computes the **cohesiveness** value of each of the two child communities. The cohesiveness of a child community indicates how well separated it is from its sibling.

Given a child community c_a and its sibling c_b, let $\gamma(c_a)$ and $\gamma(c_b)$ be the cohesiveness of c_a and c_b respectively, and they are calculated as follows. If E_c is the set of edges contained in the initial community c and E_{c_a} the set of all edges contained in c_a, $\gamma(c_a) = |E_{c_a}|/|E_c|$

3) Let T_γ be a predefined threshold for the cohesiveness value that dictates when Scanner Hunter would stop dividing communities. More specifically, if $\gamma(c_a) \geq T_\gamma$ and $\gamma(c_b) \geq T_\gamma$, the algorithm will retain both c_a and c_b and proceed to check if each of them can be divided further into smaller communities. Otherwise, the community c will not be divided further.

4) The algorithm stops when there is no community that can be subdivided.

The inquisitive reader can consult [4, 5] for details.

4. Community labeling: Even legitimate users may form communities in a bipartite graph because some popular web pages or embedded objects may become unavailable by accident. For this reason, in this step, Scanner Hunter inspects each community individually and employs a heuristic to determine whether the community consists of benign IPs or HTTP scanners. To this end, we carefully studied the scanners' activities to find potentially useful metrics. After many experiments, we narrowed down to two metrics that gave high precision: *average HTTP failure ratio* and *average embedded objects ratio*. The two metrics in combination capture the behavior of the scanners sufficiently for our algorithm to achieve high precision. Even though legitimate users may produce failed requests, their failure ratio is usually low. Moreover, scanners tend to not download embedded objects such as images or video and instead focus more on resources like `php`, `asp`, and `mdb` (database) files, which are more likely to be exploited.

We now define the average failure ratio and average embedded object ratio of a community c to capture the difference between communities of scanners and legitimate users.

- For IP address i, let T_i be its number of requests, F_i its number of failed requests, and N_i be the number of requests that IP i made for non-HTML document type resources (for example, images, javascript objects, etc.), i's **failure ratio** and **embedded-object ratio** are respectively: $FR(i) = F_i/T_i$, $ER(i) = N_i/T_i$
- The average failure ratio and the average embedded-object ratio of all IPs in a community c are respectively:
  $FR(c) = \frac{1}{|c|}\sum_{i\in c} FR(i)$, $ER(c) = \frac{1}{|c|}\sum_{i\in c} ER(i)$

If there exists a community c where $FR(c) > T_{FR}$ and $ER(c) < T_{ER}$, Scanner Hunter will label c as a community of HTTP scanners. All IPs in these community are considered as HTTP scanners. This heuristic is very useful because it allows Scanner Hunter to avoid mislabeling communities that may be formed by benign IPs requesting misplaced/temporarily unavailable resources. We will discuss the effectiveness of this heuristic in the following section.

We also would like to stress that both co-clustering and labeling are integral to the performance of Scanner Hunter because: i) inspecting individual IPs and relying on the IPs' failed requests alone will generate a high amount of False Positives (which we will show in section 3 and ii) blindly labeling each community after co-clustering will also lead to

[2] A soft clustering algorithm is one that allows a single node in the graph to be in more than one communities.

mislabeling benign IPs as scanners, as we have explained in Step 4 of Scanner Hunter.

3. PERFORMANCE EVALUATION

In this section, we present the dataset used in our study (3.1), how we select values for the parameters of Scanner Hunter (3.2), and how we evaluate the performance of Scanner Hunter (3.4).

3.1 Dataset and evaluation method

Our real-world dataset consists of 28 weeks of Web traffic collected from a University campus network.

Our data collection tool was deployed at the only edge gateway router connecting the University to the Internet, which all incoming and outgoing Web traffic passes through. At this gateway router, we captured the full payload of all packets arriving at TCP ports 80, 8000, and 8080 of the University's servers. Afterwards, we performed Deep-Packet Inspection (DPI) on each packet to extract all of the HTTP requests sent toward the websites internal to the University as well as the associated HTTP responses. The important fields that we logged from each HTTP request and response are: timestamp, URL, server IP, client IP, content-type, content-length, response code, referrer, and User-Agent.

On a average, we recorded 70.6M requests a week from 929.6K external IPs, 409.8K of which generated a total of 7.5M failed requests. We presented these weekly averages because we apply Scanner Hunter on the weekly partitions rather than the full dataset. We will discuss the motivation for partitioning data into weeks in the next section.

Metrics: We use two metrics to measure Scanner Hunter's performance: **Precision (P)** and the total number of labeled scanners. Given the numbers of **True Positive (TP)**, **False Positive (FP)**, the Precision, which is the fraction of IPs we label as scanners that are in fact scanners, is calculated as: $P = \text{TP}/(\text{TP} + \text{FP})$

Validation. Given that we do not have the ground-truth in our dataset, we did not evaluate the recall for Scanner Hunter. Instead, we focus on assessing the accuracy of identification, which we try to achieve through sampling and manual verification. We manually assess if each IP is an HTTP scanner or not based on all its HTTP transactions during the monitoring interval. Specifically, we consider its User-Agent, the Re-Paths it asked for, its failure ratio, its embedded-object ratio, the referrers for each of its requests, and the HTTP response codes that it received when the requests themselves failed. We are very conservative in our manual verification and only label those IPs with very obvious scanning activities as scanners.

To evaluate the precision, we randomly select 300 IPs from labeled scanners by Scanner Hunter, then manually verify them as described before, and label an IP as a True Positive, if it clearly exhibited the characteristics of an HTTP scanner. Otherwise, we count it as a False Positive.

3.2 Parameter selection

a. Cohesiveness threshold T_γ: The soft co-clustering algorithm has the cohesiveness threshold T_γ as its sole parameter. To recap, the cohesiveness threshold determines when the algorithm should stop subdividing a community. As Figure 2(a) shows, the lower the value of cohesiveness, the co-clustering algorithm creates more smaller but more strongly connected non-trivial communities. A non-trivial community is one in which there is more than one IP and one Re-Path. Essentially, lower cohesiveness value leads to the creation of communities in which the IPs are more similar in behaviors because they have requested similar resources.

To determine the right threshold value, we conducted an experiment where we varied T_γ and tried to assess the performance of Scanner Hunter on the training partion d. For each value of T_γ shown in Figure 2(b), we can see that when we set the value of T_γ to 0.975, we achieve a good balance between precision and the total number of labeled scanners. This is the value that we use in the rest of this paper.

b. Embedded and failure ratios: Once we have the communities, Scanner Hunter performs its labeling step. To recap, given a community $\mathbf{c_i}$ in the set of all communities $\mathbf{C}$, the community's failure ratio is $\mathbf{FR(c_i)}$ and $\mathbf{ER(c_i)}$ its embedded-object ratio. Let T_{FR} and T_{ER} be thresholds such that if $FR(c_i) > T_{FR}$ and $ER(c_i) < T_{ER}$, we label c as a community of HTTP scanners and c_i as a community of users otherwise. We will show below the process that helps us decide the values for T_{ER} and T_{FR}.

We first picked one weekly partition of our data as a training dataset, henceforth referred to as d, and ran Scanner Hunter's clustering step on d with a cohesiveness value of 0.975. Given the set of communities $C = \{c_1, c_2, ..., c_n\}$ produced by this step, we represent each c_i as a pair of coordinates (x_i, y_i) where $x_i = FR(c_i)$ and $y_i = ER(c_i)$ and plot the set C as a heat map as shown in Figure 2(c).

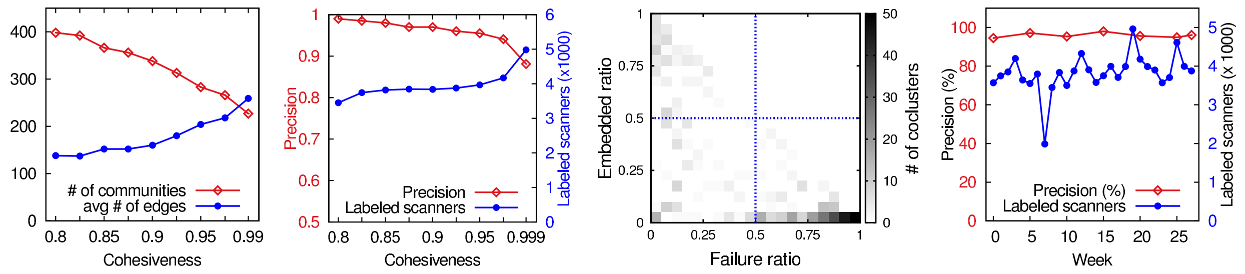
Recall that an HTTP scanner tends to have a higher failure ratio and a lower embedded-object ratio than a legitimate user because the scanner does not know exactly where the vulnerable resources reside and does not want to waste bandwidth downloading objects it does not need. This observation, in turns, implies that the communities located in the lower right quadrant of the heat map in Figure 2(c) would be the most suspicious.

Selecting threshold values for T_{FR} and T_{ER}. We begin with the community with the highest failure ratio and the lowest embedded-object ratio and set T_{FR} and T_{ER} to this community's values, in essence defining a rectangular area limited by the lines $y = T_{ER}$, $x = T_{FR}$, $x = 1.0$, and $y = 0$. We then manually inspected each community inside this area to judge whether each IP in that community is an HTTP scanner before we expanded the area by increasing T_{ER} and decreasing T_{FR}. What we observed in this process is that when we vary the values of T_{ER} and T_{FR} within the range of [0.25, 0.6], Scanner Hunter achieved a reasonable balance between the number of correctly identified scanners and the number of mislabeled legitimate users. We also repeated the same process with different values of cohesiveness and came to the same conclusion each time: with [0.25, 0.6] as a range of candidate values for T_{ER} and T_{FR}, Scanner Hunter achieves a high level of accuracy.

In conclusion, our study showed that the performance of our approach is quite robust to the values of the thresholds in the range [0.25, 0.6]. In the rest of this paper, we report results using $T_{ER} = 0.5$ and $T_{FR} = 0.5$.

3.3 Understanding False Positives

We want to further investigate why our approach misclassifies some IPs as HTTP scanners. An added benefit of using a manual verification process is that it provided us with a first-hand knowledge of falsely labeled IP addresses. We find two basic groups among the False Positive cases:



(a) # and avg. size of communities as a function of cohesiveness

(b) Precision and # of IPs labeled as scanners with varied cohesiveness

(c) T_{FR} and T_{ER} of communities generated by cohesiveness value 0.975

(d) # of IP prefixes of scanners and users

Figure 2: Parameter selection. In 2(b), the values for T_{ER} and T_{FR} are respectively 0.5 and 0.5

Benign web crawlers: 89% of all misclassified IPs belong to stealthy crawlers masquerading as regular users, as we determined in our manual inspection. These IPs downloaded very few embedded objects and they crawled a small number of pages, some of which happened to be unavailable due to broken links. The resulting high failure ratio and low embedded-object ratio confused our algorithm.

Legitimate web users: The remaining 11% of the False positives are legitimate users. Let's say that there exists a web page w_1 at the URL `abc.com/index.html` and there is an embedded object o on w that is hosted by website w_2 at the URL `xyz.com/tmp.php`, where our monitoring tool is placed. In the case that w_1 is very popular and o is suddenly unavailable, all of the requests intended for o will consequently fail.

Because our monitoring point is at w_2, it would appear as if a large number of users are requesting a non-existing resource because our monitoring tool is not aware of w_1. As a result, Scanner Hunter will group the IPs requesting o into a community in which the failure ratio and embedded-object ratio exceed the predefined thresholds. All of the IPs in this community will then be labeled as HTTP scanners. This is the reason why Scanner Hunter mislabeled that very small percentage of benign IPs in our dataset.

3.4 Evaluating Scanner Hunter

a. Precision is stable over time. We used one week's data to establish the values of the three parameters of our Scanner Hunter, namely cohesiveness threshold T_γ, embedded-object ratio threshold T_{ER}, and failure ratio T_{FR}.

We assess the performance of our approach on five other weekly data partitions in Figure 2(d) (top flat red line). On average, the precision of Scanner Hunter is 96.5%. This is a success, given that there are nearly one million external IPs in each weekly data partition. We validated these Precision rates in through sampling and manual inspection, using the same process as in the training set.

Figure 2(d) (lower jagged blue line) shows the total number of IPs labeled as scanners by Scanner Hunter for each of the weekly partitions. Note that the sharp drop in identified scanners on the 8^{th} weekly partition is an artifact of the data collection: for three days out of seven of the 8^{th} week, there were issues with the monitor device and the collection was incomplete.

b. Comparing Scanner Hunter with our baseline approaches. Here, we introduce two baseline approaches as reference points in evaluating the performance of our approach. We consider two fundamental approaches: a simple heuristic and a group of machine-learning algorithms. Both of these methods develop profiles and compare IPs in isolation, in contrast to our graph-based approach. We show the results for these two baseline approaches to provide some insight into the complexity of the problem of detecting HTTP scanners. Ultimately, this comparison demonstrates that the graph-based technique used by Scanner Hunter is crucial for obtaining high precision and recall rates.

First, the simple heuristic decides if an IP is a scanner only based on its failure and embedded-object ratios, as we explain below. Second, the machine-learning (ML) approach develops a profile of how a HTTP scanner behaves, and uses that to determine if an IP is a scanner or not. We used the WEKA machine learning software tool, and considered multiple ML algorithms and did significant fine-tuning and parameter optimization.

We introduce both approaches in order to provide some insight into the difficulty of the problem Scanner Hunter was designed to solve. Overall, both the simple heuristic and the ML approach perform poorly (19.1% and 54.3% respectively), compared to the 96.5% precision of Scanner Hunter. We attribute this to the use of communities that helps reveal the similar behaviors of scanners as they are viewed in terms of groups while looking at each IP in isolation does not provide enough information to identify scanners.

1. Simple heuristic: The simple heuristic determines if an IP i is an HTTP scanner or not based on the two metrics: $FR(i)$ and $ER(i)$. Specifically, if $FR(i) > T_{FR}$ and $ER(i) < T_{ER}$, then IP i is labeled as a scanner. Recall that Scanner Hunter determines whether a community is one of HTTP scanners based on the community's failure ratio FR and embedded-object ratio ER.

For evaluating the simple heuristic, we randomly selected a weekly data partition D_i and applied the heuristic on it. We varied the values for T_{ER} and T_{FR} and chose the ones that produced the **highest** Precision rates, which we obtained in the same way that we described in Section 2.

2. Machine-learning algorithms: From the machine leaning software collection WEKA [6], we selected three commonly used algorithms: Support Vector Machine (SVM), K-Nearest Neighbors (K = 1, 3, 5), and Decision Trees. We created our training data from HTTP scanners identified by Scanner Hunter, since we are not aware of any other method to obtain a large training set. We considered using only manually verified scanners, but the training set would have been to small.

Feature selection. For each algorithm, we consider a total of 15 features: failure ratio, embedded-object ratio, suspicious referrer ratio, the number of and the ratio of requested non-existing Re-Path, the maximum, average, and minimum size of connected component in the referrer graph, the maximum number of consecutive failed HTTP requests, the Inter-Arrival Time (IAT) between two consecutive failed requests, and the number of retries after a failure. Using WEKA's feature elimination capability, we narrowed the set of 15 features down to a total of 9: failure ratio, embedded-object ratio, suspicious referrer ratio (a referrer value is suspicious if it is the empty string or it is exactly the same value as the URL of the website), the ratio of requested non-existing Re-Paths, the size of the largest connected component in referrer graph, the maximum number of consecutive failed HTTP requests, the IAT between two consecutive failed requests, and the number of retries after a failure.

Training the ML algorithms. We picked two consecutive weekly data partitions D_i and D_{i+1}, the former for training and the latter for testing. We first applied Scanner Hunter on D_i to obtain a list of n potential scanners and selected also n IPs determined to be benign by Scanner Hunter; we call this first set M and the second B. We then used the HTTP requests produced by each IP in both sets to train the ML algorithms, which were then used to classify the IPs in the data partition D_{i+1}.

Note that the IPs in set M do not include all of the IPs labeled as scanners by Scanner Hunter in D_i. We have to exclude some IPs from Scanner Hunter's results because the ML algorithms require that there are enough HTTP requests produced by each IP (in this case, at least 5) to be effective. Each of the IPs in M then had at least 5 HTTP requests associated with them.

From our experiments, the three ML algorithms exhibited comparable performances. The Decision Tree algorithms (more specifically "Adtree") achieved slightly higher precision in a few of our experiments. As a result, we selected Decision Trees as the representative from the ML-based approach, and we use the term "ML learning algorithm" to refer to Decision Trees for the remaining of this paper.

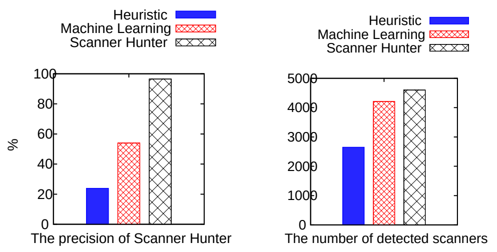


Figure 3: Performances of Scanner Hunter and baselines

Comparison: the precision of Scanner Hunter is close to twice that of the ML algorithm. We can see from Figure 3 (top) that the precision for the simple heuristic and ML approaches are only 19.1% and 54.3%, respectively. It must be stressed that for the heuristic approach, we varied the parameters and provided the **highest** precision it could achieve. For the ML approach, we picked the algorithm that gave the best performance with respect to precision. The fact that both approaches performed poorly underscores an important observation: scanners and users exhibit a wide range and often similar behaviors that classifying individual IPs is a difficult problem. This suggests that there is a need to aggregate the behavior and classify groups instead, which forms the basis of our graph-based approach.

Comparison: Scanner Hunter identifies more scanners more accurately. Not only is the precision of Scanner Hunter higher, it also identifies more scanners than the two baseline methods. In Figure 3 (bottom), we plot the number of scanners that each method identified. Although the difference between the numbers of detected scanners for Scanner Hunter and the ML algorithm is not large, the fact is that Scanner Hunter enjoys a much higher precision and therefore is capable of capturing more scanners while having much fewer false alarms.

3.5 Discussion

Here, we discuss various practical deployment tips, how adversaries may evade Scanner Hunter, and the effect of NAT and dynamic IP allocation.

Selecting the right window of observation. Throughout the paper, we used a week as the time interval units and partitioned our data accordingly. We experimented with daily web logs, but we observed that the amount of scanner traffic in each partition was too small. As a result, there was limited information available to the co-clustering step, which is a critical component of Scanner Hunter. Specifically, with an insufficient number of HTTP requests from scanners, cohesive communities would not be formed and scanners would eventually be isolated into their own trivial communities. When we applied Scanner Hunter on multiple daily data partitions, we observed that Scanner Hunter still exhibited high precision, but the amount of detected scanners dropped significantly.

As a practical tip, if a network administrator wants to use Scanner Hunter, we would recommend them to tune the monitoring interval according to the size of their network, as one that is large enough may attract sufficient requests from scanners to enable a finer-grained partitioning of data.

Evading detection. At a high level, the scanner should have to (a) visit available web pages that are unrelated to its purposes, (b) download embedded objects such as pictures and video, and (c) send HTTP requests from as many different IPs as possible, ideally, one HTTP request per IP. However, these efforts would increase the cost of scanning in terms of time, download bandwidth, or the number of IP addresses under the malicious entity's control. Furthermore, this would force the scanner to stand out from the targeted website's point of view, as the scanner would have no choice but produce much more traffic toward the website.

How dynamic IP allocation and NAT affect Scanner Hunter. Because Scanner Hunter captures the behaviors of HTTP scanners, the dynamic allocation of IPs with moderate dynamic behavior would not impair Scanner Hunter's performance. Scanner Hunter does not use any information that can be retrieved from the IP address for the purpose of detection. In fact, the use of a group-based method will likely circumvent the issue as the same scanner behind many different dynamic IPs would search for similar non-existent resources and be put into the same community.

It is possible that in the case where there are many machines sharing the same public IP due to the effect of Network Address Translation (NAT), the amount of failed requests produced by a malicious scanner may become "diluted" by of many more legitimate requests from the other machines with an identical IP. However, in practice the effect of NATs will depend on: (a) how popular the targeted website is, and (b) the number of real IPs behind the NAT.

Both factors determine the probability that a non-scanning IP behind the NAT will contact the same website that the scanner behind the NAT is probing. Finally, if NATs become a significant concern, we can modify the way in which Scanner Hunter operates: instead of grouping requests by the IP addresses can group the requests by the tuple ⟨IP, User-Agent⟩. We believe this would be a good solution as it has been shown in [7] that when a User-Agent is combined with an IP address, they can become a fingerprint that can effectively distinguishes remote browsers.

Using Scanner Hunter to assist in creating honeypots. Once Scanner Hunter has been run, the network administrator can certainly gather the nonexistent URLs requested by the scanners to create a set of fake webpages not linked to by any other legitimate pages. They then can monitor these fake pages and observe the IPs trying to access them.

4. PROFILING SCANNERS IN THE WILD

In this section, we present the results of the extensive study we conducted on the behaviors of HTTP scanners in the wild. These are the highlights from this study: 1) scanners are widely dispersed across IP space; 2) more than 90% of the returning scanners look for new resources and at least half of the Re-Paths that they request are new; 3) scanners spent little attention on disguising themselves to avoid detection, as seen by their complacent use of User-Agent and referrer fields in the requests; 4) there are four categories of vulnerabilities that the scanners in our dataset are interested in.

4.1 Spatiotemporal properties of scanners

One of the more obvious questions that comes to mind is whether a network administrator can prevent scanning activities by blocking HTTP requests from certain IP prefixes. The answer is that such a solution will not be very practical or efficient because of two conclusions that we made once we have studied more closely the spatial and temporal properties of scanners: 1) the filter set will have to be large enough to cover all of the diverse IP prefixes, 2) the filters would have short life-span and will be of limited use because most (80%) HTTP scanners, as we will show, are seen only once and never come back.

Figure 4(a) demonstrates the reason why we conclude that the IP addresses black list keep track of for the purpose of blocking scanning IPs will have to be large. In this figure, we show the percentage of distinct prefixes that remain each time we remove the least octet from the IP address. We can see that, for example, the number of distinct /16 IP prefixes accounts for at least 60% of the total number of original IPs. Furthermore, because HTTP scanners are as diversely distributed across the IP space as users, it follows that an extremely large and fine-grained filter set would be needed to block the scanners.

Figure 4(b) demonstrates the reason why the aforementioned filters will have limited life spans and usefulness. In this figure, we show the result of the experiment in which we kept track of each scanner labeled by Scanner Hunter and count how many daily visits they paid to the monitored websites during one week and during the entire six-month period. Interesting enough, no more than 20% of the scanners return to the websites hosted by the University campus in either periods. In fact, less than **5%** of them came back for a third time. This clearly indicates that no IP prefix-based, static blacklists can keep up with the scanning activities.

What we would like to know then, given that some (20%) scanners do return to the same websites, is whether scanners look for the same resources or different ones during subsequent visits. To find out, we performed an experiment on the entire six-month-long dataset as follows. Let the set S be scanner IPs that visited the monitored websites more than twice on different days. For each $s \in S$ we calculated the Average Ratio of new resources that s looks for using the following formula:

$$AR(s) = \frac{1}{n-1} \sum_{i=2}^{n} \frac{|W_i \setminus \bigcup_{j=1}^{i-1} W_j|}{|W_i|}$$

n is the total number of daily visits that s paid to the monitored websites, W_i is the set of Re-Paths sought by s on the i^{th} visit in the dataset, and $\bigcup_{j=1}^{i-1} W_j$ represents the accumulative set of every resource that s has requested up to the $(i-1)^{th}$ visit. The $(i-1)^{th}$ and i^{th} visits do not have to be on consecutive days.

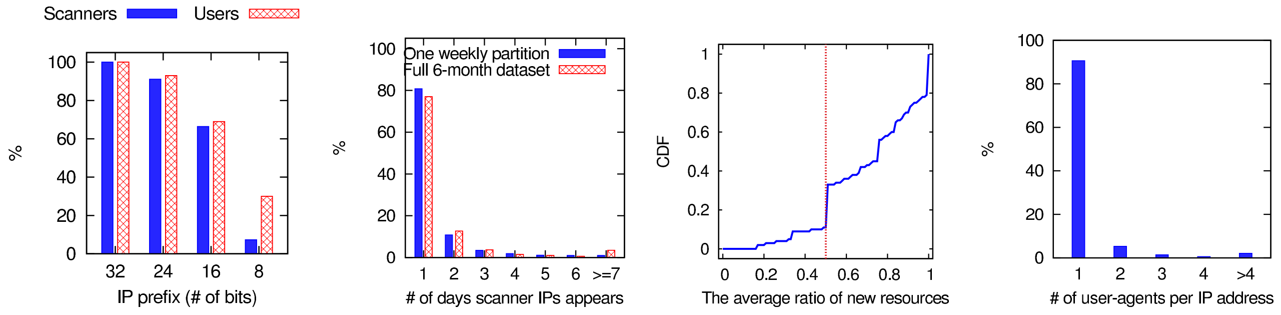
What we discovered from the experiment is shown in Figure 4(c): for more than 90% of the returning scanners, half of what they look for each time is new. This indicates that the scanners that do come back tend to probe for resources that they did not request before.

4.2 The tools used for HTTP scanning

Not only are we interested in identifying HTTP scanners, we also would like to know about the existing tools that allow them to carry out the act. As it turns out, the values of the User-Agent fields in the HTTP requests can give us an insight into the types of applications the scanners use. In Appendix A we show the values and statistics for the top 10 most popular User-Agents used by the HTTP scanners identified by Scanner Hunter so the more inquisitive reader can take a look. Although most of them are simpler than User-Agent strings produced by major browsers, there are three that deserve the most scrutiny:

- `mozilla/4.0`: Even though the User-Agent string mentions mozilla, the requests with this User-Agent string were not generated by the popular Firefox browser because it has no information on the system on which the browser runs or the information about the plugins within the browser. An example of a legitimate User-Agent string is: `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/ 28.0.1500.71 Safari/537.36`
- `<empty>`: This means the User-Agent field is empty. By default, none of the major web browsers generate requests with empty User-Agent field.
- `ineturl:/1.0`: This User-Agent string can be found by a number of HTTP-based applications that use a specific library. In this case, it may be possible that a number of malicious entities used the library to implement their scanning tools.

At the same time, we were also interested in the distribution of the number of User-Agents per scanning IP address. From Figure 4(d), we can clearly see that each IP in the 90% of the IPs labeled as scanners by Scanner Hunter produced requests with exactly one User-Agent string. We take this to suggest that most scanners use only one scanning application.



(a) Number of IP prefixes of scanners and users (b) Number of IP prefixes of scanners and users (c) Scanners look for new things when coming back (d) Scanners tend to not have empty User-Agents.

Figure 4: Spatio-temporal behaviors of scanners in the wild

There are IPs associated with more than one User-Agent string, as seen in the same Figure. When we manually looked into the HTTP requests produced by those IPs, we observed that a) different User-Agents showed up at different times during the day, and b) different User-Agents requested different Re-Paths. Therefore we believed that most of them were different scanners that happened to fall under the same IPs due to the effect of Network Address Translation (NAT).

Interestingly, however, there is a reason for us to believe that there were also scanners that used more than one application at a time. For example, there are four IPs located inside the Autonomous System number `50543` that are suspicious because:

- Each IP appeared on multiple blacklists [8].
- Each IP is associated with the same two User-Agents.
- Their behaviors are similar. For example, suspicious IP s_1 sent two set of requests, each with a different User-Agent. The requests are close in time and the two User-Agents almost always appeared in a specific order.

Given the above facts, we came to the conclusion that there is a strong likelihood each of those IPs used two applications for malicious purposes: one application to probe for the existence of vulnerable resources and the other to analyze the resources that the first application actually discovered. Even more interesting is the fact that the same two User-Agents could be seen with each of the 4 IPs: `xpymep.exe` and `mozilla /4.0 (compatible; msie 6.0; windows 98; win 9x 4.90)`. According to [9], `xpymep.exe` is the name of a binary that is considered unsafe to download.

One natural question that can be asked is whether a network administrator can block scanning activities using the values of the User-Agent fields in the HTTP requests. The answer is no because of two reasons: 1) the False Positive rate would be high, and 2) even though such a solution may mitigate scanning activities in the short term, it would not be effective in the long run due to how trivial it is to change the value of the User-Agent field. After all, scanners leave their User-Agent strings as-is possibly because of the little attention the problem has received. Once the scanners realize that the User-Agent strings may be a liability, we believe they would spend more effort on disguising the User-Agent more carefully.

4.3 What resources scanners look for

We present our findings regarding the resources that a scanner may be most interested in because armed with this knowledge, network administrators can gain a better understanding of the scanners' motivations and better secure their networks. For this purpose, we collected all the Re-Paths requested by the IPs labeled as scanners by Scanner Hunter for the entire 6-month duration of our full dataset, extract their file types, and ranked them according to their popularity.

There are three types of files that dominate the list and account for 97% of the requests: `php` (43%), `asp/aspx` (26%), and `rar/zip/7z` (24%). The most probable explanation would be that `php` and `asp/aspx` scripts are the least secured with respect to access permissions or more likely to contain exploitable vulnerabilities. The `rar/zip/7z` are compressed archives that, in the context of web server management, tend to be files used for backup or storing outdated data and, if left unsecured, would provide information that could enable future exploitations.

Identifying groups of resources systematically. We wanted to obtain a deeper understanding as to the intention of the scanners beyond the file types they are looking for. For that, we tried to identify categories of scanners based on the targeted Re-Paths. More precisely, we performed the following actions on each of the communities labeled as scanners by Scanner Hunter:

(a) We extract all Re-Paths from the community and remove all non-alphanumeric characters from them.
(b) We convert each processed Re-Path into **trigrams**, which are sequences of three consecutive characters. Once the trigrams are extracted, the community is represented by the set of unique trigrams.

Given the set of communities $C = \{c_i\}$, $1 \le i \le |C|$, let t_i be the trigram set representation for each c_i. We then compute the Jaccard similarity measurement between every possible pair: $J(c_i, c_j) = |t_i \cap t_j|/|t_i \cup t_j|$

We then identify groups of similar communities of scanners as follows:

(a) We pick the pair of communities with the largest similarity value and, if this similarity exceeds a threshold T_J, we merge them to form a new community.
(b) We calculate the similarity measurement between the new community with all old ones.
(c) We repeat the above two steps until the largest similarity between any two communities is less than T_J.

We experimented with different values for the Jaccard similarity threshold T_J and found out $T_J = 0.3$ works reasonably well. Table 1 shows the top 10 largest groups (out of 42) that remain once the merging concludes. The first column contains the ranks of the groups according to the number of unique IPs, the second the number of scanning IPs in each

#	#IPs	#Re-Paths	File types	URL patterns	Description
1	913	337	asp(58.3%) php (36.7%)	reg.asp, user_reg.aspx bbs/reg.php, login.asp	User registration and login pages
2	664	1143	asp (91.2%)	*member/index_do.php	Unclear
3	569	11273	rar/zip/7z (94.7%)	*wwwroot.zip	Backup files
4	433	9098	php (87.1%)	{include | data | plus}/*.php	Unclear
5	203	4836	asp (99.8%)	wp-{admin | login | comments | trackback}*.php	wordpress-related vulnerabilities
6	126	689	php (94.7%)	wp-content/*/[tim]thumb.php	Vulnerabilities that may allow remote code execution
7	112	401	php (100.0%)	*/demo/index.php	Demo versions of web services. May be more vulnerable than actual services
8	88	638	asp (88.3%)	*admin*	Admin-related pages
9	59	123	asp/aspx (87.2%)	*/{fckeditor | ewebeditor | htmledit}/*	Some websites let users change contents through various editor interfaces, which the scanners searched for
10	58	139	php (90.6%)	*{phpmyadmin | sql}*	Control panels for backend databases

Table 1: The 10 largest groups of HTTP scanners and their targets. "*" indicates any sequence of characters.

group, the third the number of unique Re-Paths found in each group, the fourth the most dominant file types as well as the percentage of the Re-Paths with these file types, the fifth a brief description for the observed patterns in each group's Re-Paths, and the final column some explanations regarding the vulnerabilities the scanners were aiming for.

Note that there are many ways to do the above grouping, but we adopted this approach because it is simple and the groups it produces exhibit similar scanning behavior as we discuss below.

The following are the key take-aways from our close inspection of the final groups.

1. Individual groups are homogenous in terms of User-Agents and file types even though we only merged communities based on the trigrams in their Re-Paths. In fact, for each group, the most dominant User-Agent often accounts for at least 90% of the scanner IPs and the most popular file type appears in at least 87% of the Re-Paths except for Group number 1.

This observed homogeneity within groups shows our grouping scheme works well for the purpose of discovering popular categories of vulnerabilities the scanners are interested in.

2. There are four major categories of resources the scanners want to find on the websites hosted by the University campus. They are as follows.

(a) **Public login and registration pages** for regular users. Related group(s): 1.

(b) **Control panels exclusive to administrators**. For example, web interfaces for website management and interactions with backend databases. Related group(s): 5, 8, 9, 10.

(c) **Vulnerable pages**, which may allow intrusion through remote code execution, cross-site scripting, or SQL injection [10]. Related group(s): 6.

(d) **Backup files** that may contain sensitive information for exploits. Related group(s): 3.

The resources found in these categories are quite sensitive for security purposes. If the scanning entities successfully exploit them, they can seriously compromise the security of the websites. For example, the scanners in Group number 6 were primarily concerned with exploiting a known vulnerability in Wordpress [10] that would allow remote execution of malicious code.

There are groups whose intents are unclear (groups 2, 4, and 7), so we did not list them in the above categories.

3. Backup files are a major target of scanners. It is surprising how aggressive (11,273 Re-Paths for 569 IPs) they are in Group number 3 of the table in the way they tend to have more unique Re-Paths per IP than other groups.

4. Toward a systematic categorization of scanner intention. This grouping scheme helps us uncover the top 10 major vulnerability categories from the web traffic of the University campus. This represents another significant use of our approach. Using Scanner Hunter and the grouping scheme, network administrators would be well-informed about the most sought-after vulnerabilities, and thus be able to proactively secure vulnerable resources.

4.4 The complacency of Scanners

We have seen that scanners do not attempt to disguising their User-Agents. Further inspection shows a continuation of this complacency, where they do not try to make the values of the referrer field in the requests to look similar to those of legitimate users: **90%** of the times, the referrer field of a scanner's request is empty versus **10%** for a user.

Recall that a request usually has empty referrer field when a user types the URL directly into the URL bar of their browser or clicks on a bookmark. However, there are many embedded objects on a web page, so the referrer field in every request to an embedded object will be the URL of the original page. This is why only a small percentage of legitimate requests have empty referrer fields.

This is not the case with scanners because: a) the pages they requested do not exist most of the times and therefore there were no subsequent requests for embedded objects b) even if the pages existed, the scanners tend to only request the pages without downloading the embedded objects. Also noteworthy is the fact that in about 5% of the HTTP requests generated by scanners, the referrer field and the URL field are identical.

5. RELATED WORK

To the best of our knowledge, there exists no work on the specific problem of HTTP scanners. There are, as noted before, many studies that focus on the problem of detecting crawlers [3, 11, 12].

Much work has also been done on detecting port scanners [13, 14]. Port scanning is different from HTTP scanning in that port scanners check for open ports, which are limited and well-defined, to determine which applications are running. HTTP scanners explore a much wider field where they do not know where a file of interest (e.g. `myadmin.php`) may be on a remote server or even whether the name of the file has been changed. Moreover, the existing work may not translate to solutions in the case of HTTP scanner. For instance, solutions like [13] relies on the assumption that users rarely initiate failed connections while port scanners rarely make successful ones. This assumption doesn't hold with

HTTP scanners, which often locate an existing web directory (eliciting a successful HTTP response), then search for files under it. Even users can make failed HTTP requests in succession when a website goes down. Other solutions such as [14] work on port scanners because the port range is limited and fixed, which is not the case with HTTP scanners.

The use of honeypots [15, 16] may detect some scanning activities, but we see these efforts as a complementary effort to ours. We focus on the question of "who are the HTTP scanners exploring *my* network", which may benefit from learning about scanners who are targeting other networks. However, the entities operating the scanners may be clever enough to simply target already established websites.

There is also a body of work on developing applications for security professional to test the security of specific applications [17, 18, 19] and there have been some efforts invested in detecting covert, malicious web traffic [20]. [21, 22] are two works in the literature that confirms our observation that the behaviors of web users are diverse, which is why it is difficult to automatically look at an IP in isolation and decide whether the IP is engaging in scanning activities.

6. CONCLUSION

In this work, we identify and study the problem of HTTP scanners, which is by far not trivial and has received little attention to date.

Our main contribution is Scanner Hunter, an effective method to detect HTTP scanners that achieves a 96.5% detection precision. The key novelty of Scanner Hunter the use of a graph-based approach that detects HTTP scanners by comparing the group behavior of scanners versus that of legitimate users. The precision of Scanner Hunter is more than double of that of baseline solutions, while detecting more scanners at the same time.

As our second contribution, we provide an extensive study of HTTP scanning over a six-month period to understand: (a) what they are looking for and (b) their spatial and temporal behaviors. It is clear that the problem is acute, with roughly 4,000 unique IPs scanning a medium-sized University network every week and 80% new IPs every week.

Finally, we provide an initial effort towards systematically inferring the intention and targets of scanning. We identify four major categories based on the targeted resources: (i) user registration and login pages, (ii) website management interfaces, (iii) pages with potential vulnerabilities, and (iv) compressed web archives (e.g `wwwroot.zip`), the products of website backup activities and should not be publicly visible.

The ultimate goal of our work is to: (a) raise the awareness to this problem and (b) provide an initial but promising detection technique to mitigate the risk posed by HTTP scanners. Our goal is to provide one more complementary layer of defense to website security, at the early stage, when hackers are scoping out potential targets.

7. ACKNOWLEDGEMENTS

The authors would like to thank the reviewers for their helpful comments and suggestions regarding browser fingerprints and using Scanner Hunter to create effective honeypots.

8. REFERENCES

[1] P. Ducklin, "DHS website falls victim to hacktivist intrusion," http://nakedsecurity.sophos.com/2013/01/07/dhs-website-falls-victim-to-hacktivist-intrusion/, 2013.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," 1999.

[3] G. Jacob, E. Kirda, C. Kruegel, and G. a. Vigna, "Pubcrawl: protecting users and businesses from crawlers," in *Proceedings of the 21st USENIX conference on Security symposium*. USENIX Association, 2012, pp. 25–25.

[4] R. Keralapura, A. Nucci, Z.-L. Zhang, and L. Gao, "Profiling users in a 3g network using hourglass co-clustering," in *Proceedings of the sixteenth annual international conference on Mobile computing and networking*. ACM, 2010, pp. 341–352.

[5] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 269–274.

[6] I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, and S. J. Cunningham, "Weka: Practical machine learning tools and techniques with java implementations," 1999.

[7] P. Eckersley, "How unique is your web browser?" in *Privacy Enhancing Technologies*. Springer, 2010, pp. 1–18.

[8] "HE BGP Toolkit," http://bgp.he.net/.

[9] "http://f.virscan.org/xpymep.exe.html," http://f.virscan.org/xpymep.exe.html.

[10] "Wordpress timthumb plugin - remote code execution," http://www.exploit-db.com/exploits/17602/.

[11] A. Stassopoulou and M. D. Dikaiakos, "Crawler detection: A bayesian approach," in *Internet Surveillance and Protection, 2006. ICISP'06. International Conference on*. IEEE, 2006, pp. 16–16.

[12] A. Lourenço and O. Belo, "Applying clickstream data mining to real-time web crawler detection and containment using clicktips platform," in *Advances in Data Analysis*. Springer, 2007, pp. 351–358.

[13] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. IEEE, 2004, pp. 211–225.

[14] C. Gates, "Coordinated scans detection," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009*. The Internet Society, 2009.

[15] N. Provos, "A virtual honeypot framework." in *USENIX Security Symposium*, vol. 173, 2004.

[16] B. Borisaniya, A. Patel, D. R. Patel, and H. Patel, "Incorporating honeypot for intrusion detection in cloud infrastructure," in *Trust Management VI*. Springer, 2012, pp. 84–96.

[17] A. Doupé, L. Cavedon, C. Kruegel, and G. Vigna, "Enemy of the state: a state-aware black-box web vulnerability scanner," in *Proceedings of the 21st*

USENIX conference on Security symposium. USENIX Association, 2012, pp. 26–26.

[18] T. Jensen, H. Pedersen, M. C. Olesen, and R. R. Hansen, "Thaps: automated vulnerability scanning of php applications," in *Secure IT Systems.* Springer, 2012, pp. 31–46.

[19] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "Secubat: a web vulnerability scanner," in *Proceedings of the 15th international conference on World Wide Web.* ACM, 2006, pp. 247–256.

[20] K. Borders and A. Prakash, "Web tap: detecting covert web traffic," in *Proceedings of the 11th ACM conference on Computer and communications security.* ACM, 2004, pp. 110–120.

[21] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Y. Jin, "Resurf: Reconstructing web-surfing activity from network traffic," in *IFIP Networking Conference, 2013.* IEEE, 2013, pp. 1–9.

[22] M. Meiss, F. Menczer, and A. Vespignani, "On the lack of typical behavior in the global web traffic network," in *Proceedings of the 14th international conference on World Wide Web.* ACM, 2005, pp. 510–518.

APPENDIX

A. SUSPICIOUS USER-AGENTS

User-Agent	% requests
mozilla/4.0	31.2%
mozilla/3.0 (compatible; indy library)	14.5%
mozilla/4.0 (compatible; msie 6.0; windows nt 5.1; sv1)	6.0%
mozilla/4.0 (compatible; msie 6.0; windows nt 5.1; sv1;)	3.8%
<empty>	2.5%
mozilla/5.0 (windows; u; windows nt 5.1; en-us; rv:1.9.2) gecko/20100115 firefox/3.6	0.8%
mozilla/4.0 (compatible; msie 6.0; windows nt 5.1)	0.8%
mozilla/5.0 (compatible; msie 9.0; windows nt 6.1; trident/5.0)	0.8%
ineturl:/1.0	0.5%
mozilla/4.0 (compatible; msie 6.0; windows nt 5.1; sv1; .net clr 2.0.50727)	0.5%
	61.4%

Table 2: The 10 most popular User-Agents in our dataset.

In the table above we show the top ten most popular HTTP scanners User-Agents that we observed in our dataset. As mentioned in section 4, the scanner User-Agents are suspicious because they tend to be too short and do not contain information about specific browser plug-ins.

B. TIPS ON ENHANCING Scanner Hunter's PERFORMANCE

1. Opportunistically improving the accuracy of detection via customization. Here, we present several potential methods for improving the detection accuracy, if additional information or the assistance of the network administrator is provided. Note that these methods are **not used** in our current implementation of Scanner Hunter. We did not implemented these ideas due to two reasons: (a) we wanted to assess the performance of the algorithm in its own merit and in its generality, as customization and the necessary additional information may not be available at some deployment sites and (b) we did not have the necessary information for our dataset for some of the proposed improvements.

- **Using an IP whitelist.** The network administrator could over time identify web crawlers, either by considering external sources of information or by observing crawling behavior on their website. We expect this to have a considerably positive impact on precision since we saw in section 3.3 that 89% of the False Positives were stealthy crawlers.
- **Take into account the web resources that become unavailable.** The idea here is to explicitly consider web sources, such as pages or files, that become unavailable either temporarily or permanently, and exclude the failed requests for those resources from the bipartite graph. This would require a collaboration between the website administrator and the network administrator, which may not be the same person in large institutions. The addition of this technique could be particularly helpful for websites that undergo significant restructuring, which for many many is not that often.
- **Focus on requests for particular file types.** Scanners seem to be interested in particular resources and file types, such as `php`, `asp`, and `zip` as we will discuss further in section 4. Our algorithm could be configured to consider only those files or give them more weight during: (a) the creation of the bipartite graph, (b) creating the communities, or (c) labeling a community.

2. Handling non-standard HTTP failure messages. Some websites do not use the standard HTTP failure messages, but Scanner Hunter can easily consider such non-standard messages. In fact, our algorithm will be deployed by the network administrator who manages the website, or at least with their collaboration. Therefore, if the website responds with non-standard failure messages, it is trivial to specify those so that they algorithm can detect them properly. Note that the website in our dataset was using conforming, `40X` HTTP failure responses, so we had no reason to make this trivial extra step. In addition, most websites respond with a `40X` code when a request fails for one reason or another. Non-conforming failure responses is often accompanied by a customized web page containing a message that states the reason of the failure. Scanner Hunter will still work as expected as long as failed requests are properly specified by the network administrator or via observation. Scanner Hunter then would take into account this information when it preprocesses the log files.