Improving Malicious URL Re-Evaluation Scheduling Through an Empirical Study of Malware Download Centers

Kyle Zeeuwen
SophosLabs Canada, Sophos Inc.
University of British Columbia
kyle.zeeuwen@sophos.com

Matei Ripeanu
University of British Columbia
Vancouver, Canada
matei@ece.ubc.ca

Konstantin Beznosov
University of British Columbia
Vancouver, Canada
beznosov@ece.ubc.ca

ABSTRACT

The retrieval and analysis of malicious content is an essential task for security researchers. At the same time, the distributors of malicious files deploy countermeasures to evade the scrutiny of security researchers. This paper investigates two techniques used by malware download centers: frequently updating the malicious payload, and blacklisting (i.e., refusing HTTP requests from researchers based on their IP). To this end, we sent HTTP requests to malware download centers over a period of four months. The requests are distributed across two pools of IPs, one exhibiting high volume research behaviour and another exhibiting semi-random, low volume behaviour. We identify several distinct update patterns, including sites that do not update the binary at all, sites that update the binary for each new client but then repeatedly serve a specific binary to the same client, sites that periodically update the binary with periods ranging from one hour to 84 days, and server-side polymorphic sites, that deliver new binaries for each HTTP request. From this classification we identify several guidelines for crawlers that re-query malware download centers looking for binary updates. We propose a scheduling algorithm that incorporates these guidelines, and perform a limited evaluation of the algorithm using the data we collected. We analyze our data for evidence of blacklisting and find strong evidence that a small minority of URLs blacklisted our high volume IPs, but for the majority of malicious URLs studied, there was no observable blacklisting response, despite issuing over over 1.5 million requests to 5001 different malware download centers.

Categories and Subject Descriptors

K.6.5 [**Computing Milieux**]: Management of Computing and Information Systems Security and Protection; C.2.0 [**Computer Systems Organization**]: Computer Communication Networks General

WebQuality '11, April 28, 2011 Hyderabad, India.

General Terms

Security, Measurement

Keywords

Malware Download Centers, Server Side Polymorphism, Low Interaction Honeyclient, Sample Collection, IP Blacklisting

1. INTRODUCTION

Information security researchers constantly download web content and system executables from the Internet looking for new threats. The typical goal in this scenario is to get the same treatment as the average Internet user: that is, to receive a malicious executable, often via a path of redirections, culminating in a browser exploit or social engineering trick that initiates the download of the executable onto the victim computer [9]. The data collected from this process is used to update URL blocklists and anti-virus detections. An ancillary goal for researchers working on algorithms to detect polymorphic malware is to download multiple samples from the same source. Thus, distributors of malware have a clear incentive to identify and thwart these acts of Internet reconnaissance.

In order to gain assurance on the accuracy and completeness of the data gathered in such an adversarial information retrieval (AIR) scenario, it is necessary to study the countermeasures employed by malware distributors. These countermeasures also impact cost factors for security researchers such as network bandwidth, computing resources, and IP allocation. Frequent updates to malicious payloads require security researchers to revisit malware download centers to retrieve the updated binaries. Significant resources can be spent repeatedly downloading the same sample from a URL, or downloading unique samples from server-side polymorphic sites [12] that deliver unique samples on each request. In addition to the crawler resources, every new sample entering a security lab requires effort in terms of system resources for automated analysis or, even more costly, attention from a human analyst.

Persistent use of finite, identifiable IP ranges, made worse by the need to reevaluate URLs after initial processing, makes the identification of researcher IPs easier for malicious adversaries. Once an IP range of a research lab is identified, the malware network can engage in various cloaking techniques, including but not limited to refusing to respond to HTTP connections, rendering a static sample as opposed to a new sample, rendering or redirecting to benign content, serving out larger than normal samples, or holding connec-

tions open for longer than normal.

Several studies have investigated HTTP servers that vary, or "cloak", their content based on the characteristics of the requesting client. Past research has identified the HTTP useragent [16, 17], HTTP referrer header [8], and other client characteristics [5] as triggers for cloaking. There is much anecdotal evidence of IP blacklists[1] being compiled and distributed [14, 1], even commercially available databases of crawler IPs [3], but few studies of their mechanics and prevalence. Similarly, several studies [13, 6, 10, 9] focus on malicious web servers and provide a quantitative measurement of their distribution, yet they do not focus specifically on the defensive countermeasures these servers deploy. Lacking a qualitative and quantitative evaluation of these behaviours, researchers must speculate when developing AIR systems.

The purpose of this study is to identify different types of malware update patterns and their relative prevalence. Using this data we propose and perform a limited evaluation of a recrawling strategy that accounts for these different patterns. A secondary objective of this study is to trigger and document cloaking behaviour based on IP blacklisting by malicious servers. To conduct this study we have developed a distributed crawler that revisits sites at various intensities. The crawler consumes lists of URLs, makes multiple HTTP requests to the URLs from different clients deployed across a range of IP addresses, and analyzes the responses looking for patterns in the sample update behaviour as well as for evidence of IP blacklisting.

The analysis of the results reveals several distinct update behaviours. We note that these distinct behaviours must be handled differently by AIR systems, and propose a configurable algorithm that dynamically updates the fetch interval for URLs being monitored by the system. We evaluate several variations of our proposed algorithm using the HTTP responses collected. Using a simple success metric that can be tuned to adjust the tradeoff between malware collection and resource conservation, we find a combination of parameters resulting in a 34% reduction in the volume of fetches with only a 10% drop in new sample discovery. By adjusting the bias to favour resource conservation, we can achieve a 94% reduction in fetch volume at a cost of 55% drop in new sample discovery. We conclude that our proposed algorithm provides an effective means to balance the conflicting objectives of sample collection and resource conservation, and provides a foundation for further research in this area.

We also find indications of blacklisting: four URLs stopped serving payloads to our *researcher* IPs (that is, IPs we use for high volume crawling) while continuing to server payloads to our low volume IPs. These URLs had many similarities, sharing the same top level domain, URL path, initial registration date, and anti-virus detection of their payloads. This evidence suggests that at least one organization is blacklisting our high volume IPs. However, for the majority of malicious URLs that we studied, there is no observable blacklisting response, despite issuing over 1.5 million requests to 5001 different URLs from our highest volume IP.

To the best of our knowledge, this is the first empirical study of malware download centers over time that specifically focuses on their update patterns and blacklisting behaviour. The data gathered, the classification scheme we propose, and the re-fetch scheduling algorithm we introduce can be directly applied to existing systems to increase their efficiency in terms of resource use and data quality.

[1]We could equally call this an IP whitelist of security researchers, used for blocking purposes, but we found that slightly more confusing than discussing an IP blacklist of whitehat IPs used by malicious servers

2. RELATED WORK

The study of malicious webservers is an active area of research. Studies of malware download infrastructure by Provos et al. [10, 9], and spyware by Moshchuk et al. [6], provide quantitative measurements of the prevalence of the online threat and insights into the techniques used by these networks. The Provos work includes statistics on the distribution of binaries across URLs, but does not provide detailed analysis on the update behaviours. Our work is similar to these studies but focuses on different behaviours of the malicious site and how the sites update over time. We also perform our research and draw conclusions from the perspective of a security research lab, as opposed to the end user experience. Online crawlers [13, 11], actively download and analyze web content in order to detect malicious URLs. Again, this approach differs from ours in that they are classifying unknown URLs based on their maliciousness, whereas we are starting with known malicious servers and studying their behaviour.

The web spam technique of "cloaking", that is to return altered content to search engine crawlers, for the purposes of search engine optimization (SEO) became a popular research topic around 2005. Wu and Davison [16, 17] performed several studies of sites that performed *semantic cloaking*. Similar to our research, they impersonated regular Internet users as a baseline as well as automated crawlers. The objective of their research differs from ours in that they are trying to *detect* cloaking behaviour and apply this to URL classification, whereas we study the cloaking behaviour in terms of how it is *triggered*. Additionally, their experiments used a single IP and did not focus specifically on malware download centers. A team from Microsoft and UC Davis [8] performed a similar study focusing on the problem of forum-based spamming as a black SEO technique. They identified a new type of cloaking known as *click through cloaking* that differentiates user from crawler based on the value of the HTTP referrer. Like our research, their crawler attempts to trigger this cloaking by varying characteristics of the HTTP client. They vary the referrer behaviour and use the presence of cloaking as a spam sign to aid in URL classification, whereas we study the temporal characteristics of the HTTP client that cause an IP to be blacklisted. Additionally, unlike our study, they do not focus on malware download centers.

3. BACKGROUND AND MOTIVATION

The ubiquity of the web and the increasing complexity of web browsers has made the Internet the attack vector of choice for distributors of malware. A malicious executable is typically delivered over the web by a multi-staged process involving several malicious agents, which we refer to collectively as a malware distribution network (MDN). Users first arrive at landing sites, which are usually recently registered domains or compromised sites. Users find the landing sites in search engine results via blackhat SEO techniques, in their email inbox via spam, or mixed in with user content on blogs and social networking sites. The landing site

redirects, via server side logic, client side JavaScript , or a combination thereof, to a malware download center [9].

As malicious content is studied and new protections are released, the malware authors typically respond by updating the malware such that it is no longer detected. However, in many cases the URL for the malware download center does not change. This means that there is an incentive for security researchers to repeatedly visit known malware download centers, looking for updated malware samples. Repeated visits to malware download centers by honeyclients provide an opportunity for adversaries to find identifiable traits and block subsequent requests made by the honeyclient, and thus the stage is set for yet another arms race in the security industry.

A malware download center can conditionally alter responses to requests from security researchers in order to prevent researchers from analyzing the malicious samples. However, the strategy of denying researchers access to malicious binaries has several limitations. The security researcher can change their IPs once IP blacklisting has been detected, thus temporarily defeating the technique. The security researcher could use an anonymizing network such as TOR, and some do, which has resulted in malicious delivery networks actively blocking all TOR exit nodes [15]. A more fundamental limitation of cloaking arises from multiple well established sample exchanges and many private data exchanges within the security community. Even if a malware download site successfully identifies and evades several security researchers, it is likely that at least one researcher will eventually be able to retrieve the malicious content, and share it with the community.

Given the high probability that security researchers will eventually get copies of malicious content distributed via the Internet, malware distributors have also begun to periodically alter malicious binaries. This technique, combined with forms of cloaking, provide an effective evasion solution. The cloaking techniques are no longer required to keep samples out of the hands of researchers indefinitely; as long as the collection of the new binary is delayed, there will exist a period where anti-virus detections for the new sample will be unavailable to customers. Once detections are updated and protection is provided, the sample can be updated again.

Both cloaking and sample updating affect honeyclients. An obvious impact of IP blacklisting is that crawlers using blacklisted IPs will not be treated in the same manner as the average Internet User, thus impeding the ability to learn from the HTTP response and protect Internet users. Malware sites that update their payload frequently affect AIR systems more subtly. The number of suspicious URLs reported daily has been consistently increasing from year to year. Sources [Ken Dunham, Private Communication] estimate between 600,000 - 1,000,000 unique potentially malicious URLs are reported daily within the industry. An AIR system must be able to quickly evaluate all new URLs, as well as periodically revisit URLs from previous days. Fetching too frequently results in wasted bandwidth and storage, and increases the risk over time that an AIR system will be identified for blacklisting purposes. Fetching too infrequently leads to delays and gaps in anti-virus protection caused by missed detections on new, unseen samples.

It is important for developers of AIR systems to understand the details of these countermeasures and the extent of their use. This research focuses on two types of countermeasures: frequent sample updating, and IP blacklisting. We focus specifically on how widely deployed these techniques are, and on gathering specifics about how the techniques are used "in the wild" with the aim of improving AIR system design in the future.

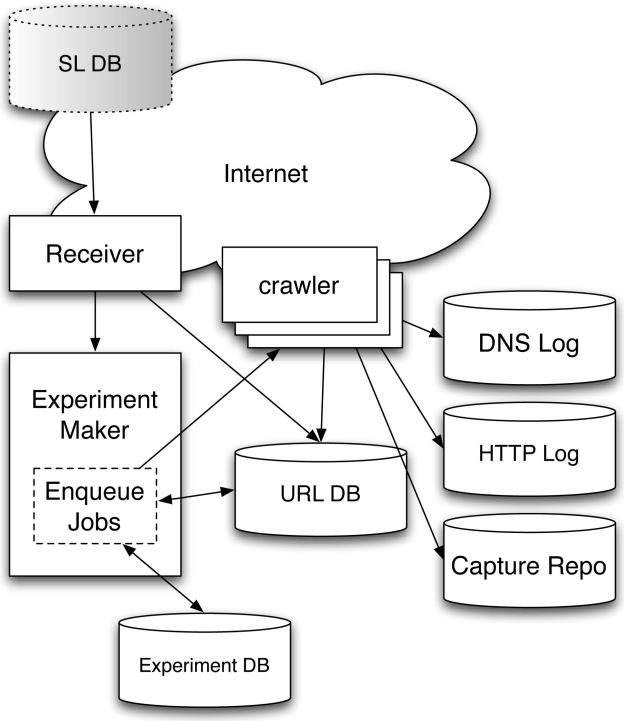


Figure 1: Architecture of the Tachyon Detection Grid. URL data from SophosLabs (SL DB) enters the TDG, where the "experiment maker" conditionally adds URLs to experiments. URL included in experiments are sent to clients to be fetched repeatedly. The HTTP responses and fetch logs are collected by the master and processed.

4. SYSTEM DESIGN

In order to study the behaviour of malware download centers over time, we built a low interaction distributed honeyclient and named it the "Tachyon Detection Grid"[2] (TDG). Figure 1 shows the TDG architecture. To limit the scope and complexity of this research, we limited our study to URLs of known malware download centers; sites that are directly serving malicious executables. This eliminates the need to fully emulate browser behaviours, particularly functions such as DOM manipulation and Javascript execution, allowing the use of a more primitive crawler.

The TDG consumes three feeds of known and suspected URLs from SophosLabs[3]. Two of these streams are batched and delivered on an hourly basis, and the third delivers URLs in real time as they become available. As we add URLs to the system, we also tag these URLs with attribute

[2]This is a geeky Star Trek reference. The Tachyon Detection Grid was used by the Federation to detect cloaked Romulan vessels.

[3]These feeds are private industry exchanges that are available to researchers on request

count	month
424	September 2010
450	October 2010
1305	November 2010
1751	December 2010
873	January 2011
198	February 2011

Table 1: Number of URLs added to the experiment

key value pairs to record information like the original source of the URL and the reported malware spread using the URL. The system monitors new attributes and selects URLs for further analysis based on configurable experiment definitions. The definitions specify criteria for URL inclusion, the remote clients used to execute the fetch, the behaviour that the remote clients should exhibit while executing the fetch, the specific times when fetches should occur, and the duration of fetching.

The crawlers are distributed across multiple sites, the majority deployed on PlanetLab [2] servers. The crawler is written in Perl and uses the CPAN LWP::UserAgent module, with modifications to the TCP and HTTP behaviour. It follows HTTP and HTML redirects and generates a log of all DNS and HTTP responses. The logs are periodically collected by the master, and added to a summary object maintained for the original URL. The system analyzes the URL summary object after each update and applies a classification if certain patterns are present. These classifications are discussed further in the Section 5. The classifications are used as triggers to take an action on the URL, such as including the URL into subsequent experiments or updating the current fetch pattern.

5. RESULTS

We conducted multiple experiments on the TDG between September 2010 and February 2011 using URL feeds from SophosLabs. In this section we present our results. We present the different update patterns that emerged and then propose and evaluate algorithms to control re-evaluation logic of AIR systems based on the observed update pattern of the URL. We also discuss the details of the blacklisting behaviour we observed.

5.1 Data Sets

We selected 5001 URLs from the SophosLabs feeds for experimentation between September 2010 and February 2011. This was done on a random basis, the only criteria for inclusion was that the URL was confirmed to be directly serving malicious executables, as confirmed by Sophos before sending the link. The rate of inclusion is shown in Table 1. URLs that were added between September and November were harvested from a single client once every hour with a random jitter of 0-15 minutes. URLs added after November 2010 were harvested from multiple clients, with the busiest client polling URLs once every 15 minutes. The difference in polling behaviours is an artifact of resource availability as the system was brought online.

Of the 5001 sites included, 1007 did not serve any executable samples, 2514 served a single executable sample for the entire duration of the experiment, and 1480 served multiple executable samples. Of the 1480 that served multiple samples, we discarded data from 589 because our experiments did not poll frequently enough for the data to be usable in subsequent analysis, leaving 891 URLs that updated their binaries. Of these 891, 162 were determined to be server side polymorphic (SSP) sites, based on this heuristic: if a site has responded with at least 30 distinct samples and at the conclusion of the experiment less than 1% of these samples were seen on more than one fetch attempt, then the site was exhibiting server side polymorphic behaviour. This left us with 729 URLs that updated their malicious binary but were not strictly server side polymorphic according to our heuristic. We used results from these 729 to generate the results presented in this section.

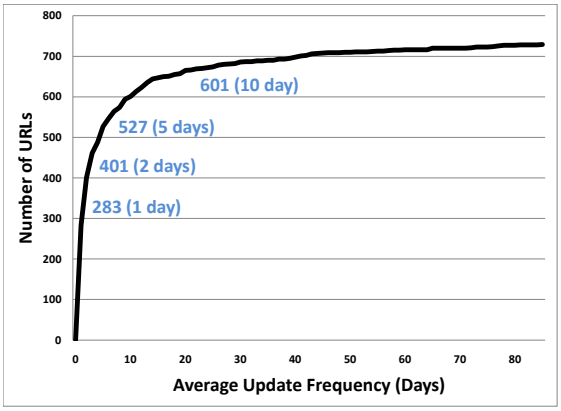
Fourty five percent of the 729 were *.com* domains, the remainder split among 40 other TLDs. There were 400 unique domains in the set, and anti virus scans of the samples reveal over 70 distinct families of malware being served from these 729 malware download centers.

5.2 Refetching Periodic Updaters

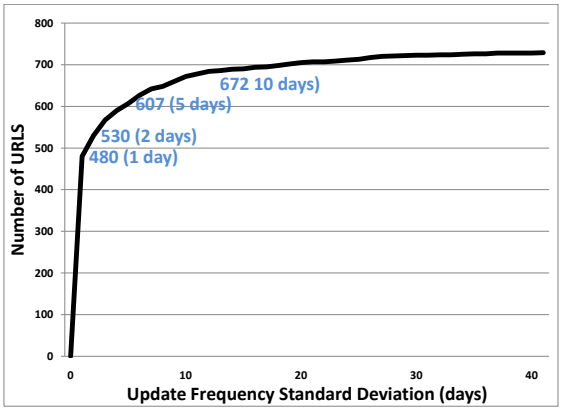
Over the four month period we performed 368,000 HTTP requests to the 729 URLs, resulting in 243,000 downloads of over 10,000 unique binaries. The distribution of the binary update behaviours for these URLs is shown in Figure 2. The graphs show that a majority of the sites update on a daily basis; however, more than 250 URLs have a standard deviation of over 1 day, suggesting that the actual delta between updates varies widely between different URLs and for the same URL over time. This makes it impractical to balance between new sample collection and resource conservation using a single, static fetch interval. Instead, the fetch interval for each URL should be dynamically adjusted based on new fetch results. We propose a multi-step algorithm to balance the objectives mentioned above.

The concept behind the algorithm is to increase the fetch interval when a duplicate result is found, and decrease the time between fetches when a new sample is found. The function that adjusts the fetch interval, referred to as the correction function, is critical to the success of the algorithm, therefore we evaluate multiple functions with a range of parameters, as described below. It is also crucial for this algorithm to quickly differentiate between periodically updating sites and sites using server-side polymorphism (SSP). An SSP site will produce a new sample on *every* fetch request, regardless of the fetch interval; therefore an algorithm designed to optimize for periodically updating sites will quickly reduce the fetch interval to 0 when operating on an SSP site. Thus, we include a heuristic to identify SSP URLs that analyzes the fetch results after each new fetch. If we see unique samples for a specified number of consecutive fetches, we classify the URL as an SSP and handle it separately, as discussed in the Section 5.3.

To evaluate the effectiveness of this algorithm, and find suitable correction functions, we simulated the fetches that this algorithm would have made on the 729 URLs from our collection that served more than one sample. We track the number of samples that the algorithm would have downloaded, and the number of fetches it would have made. The simulation ends when all the samples from the empirical set are successfully downloaded, or when we reach the chronological end of our empirical data set. We evaluate success based on the ability of the algorithm to collect all samples using a minimum number of fetches. To quantify this success

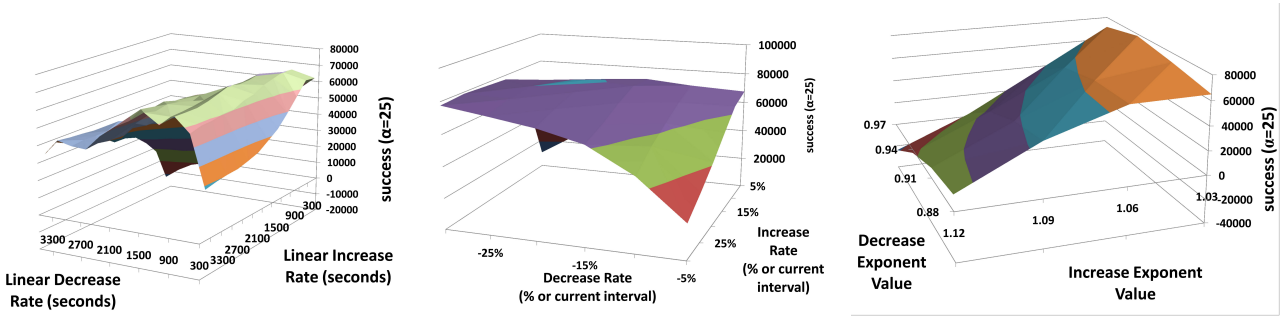


(a) Average CDF



(b) Standard Deviation CDF

Figure 2: CDF graphs showing the average and standard deviation update frequency of all non-SSP URLs. We calculated the average and standard deviation of the interval between sample updates for each URL independently and then plot the collection of averages and standard deviations as CDFs.



(a) Linear correction (b) Percentage correction (c) Exponent correction

Figure 3: Evaluation of the dynamic refetch algorithm using $\alpha = 25$ and an initial fetch interval of 10 minutes ($I(0) = 600$). Each data point was generated by simulating the fetches made by our proposed algorithm using the data we collected for the 729 URLs showing update behaviour. Note that we exclude Server Side Polymorphic sites as they do not have a finite update interval.

value we use the following equation:

$$success = reduction\ in\ fetches - \alpha \times missed\ samples$$

where α is a tunable parameter controlling the bias between sample collection and fetch resources. Increasing α will select correction functions that result in fewer missed samples at the expense of system resources. Determining representative α values based on current system load and the expected impact of missed samples is an area of future research we discuss in Section 7; to evaluate our simulation results we fixed α at 25. We evaluate using a linear correction function, a percentage based correction function, and an exponential correction function. These correction functions are provided below for clarity. We use $I(n)$ to denote the interval to wait before issuing the n^{th} request, and X to represent the coefficient value:

$$\begin{aligned} linear\ correction\ function: I(n) &= I(n-1) + X \\ percentage\ correction\ function: I(n) &= I(n-1) \times X \\ exponent\ correction\ function: I(n) &= I(n-1)^X \end{aligned}$$

The results are presented in Figure 3 and Table 2. The evaluation shows that we can significantly reduce fetch volume over our baseline, depending on how much sample loss is tolerable. Using a linear correction function with a low positive coefficient and a high negative coefficient (300 and 3300 seconds respectively) results in a 34% reduction in fetch volume with a 11.8% missed sample rate.

5.3 Refetching Server Side Polymorphic Sites

Analysis of our fetch results uncovered some unexpected behaviours by SSP URLs. In 6 cases, a site exhibiting SSP behaviour began serving the same content for a short period of time, and then reverted back to serving a unique sample on each fetch. Algorithms to detect an SSP by analyzing

α	Success Range	Fetch Delta	Missed Samples	Function	+X	-X
0-2	221608 - 201952	94.0811%	55.8822%	power	0.12	0.03
3	192352	92.6585%	49.0988%	power	0.09	0.03
4-6	184089 - 170035	90.0858%	39.9556%	power	0.06	0.03
7-8	163528 - 157882	86.2025%	32.1033%	percentage	0.3	0.15
9-10	152618 - 147385	84.7867%	29.7549%	percentage	0.3	0.2
11-12	142239 - 137323	83.3432%	27.9525%	percentage	0.3	0.25
13-19	132712 - 105172	81.6735%	26.0988%	percentage	0.3	0.3
20-21	100739 - 96580	78.0807%	23.6481%	percentage	0.2	0.25
22-33	92553 - 4944	75.8951%	22.2835%	percentage	0.15	0.2
34-57	45682 - (-31368)	67.7487%	19.0482%	percentage	0.1	0.2
58-62	-34384 - (-46406)	59.3955%	17.0865%	percentage	0.05	0.1
63-9000	-49401 - (-1861150)	34.5786%	11.8099%	linear	300	3300

Table 2: *Results of the Periodic Updater Algorithm Evaluation.* The most effective correction function/parameter combinations are shown while varying the α value. It is clear that power functions quickly reduce fetch volume, but at the expense of sample collection, whereas linear correction functions biased toward reducing the fetch interval (3300 vs 300) can be used to strongly favour sample collection.

observed fetch history should account for this by using tolerance values in their heuristics. For example, instead of identifying only sites that serve out 100% unique content per fetch, include any site above 95% that has served at least 30 unique samples.

Every HTTP request to an SSP site results in a new sample. It seems useful to collect as many samples as possible from a research and protection standpoint, however the probability that the exact samples downloaded will be used to infect a victim PC is negligible, therefore each sample is only valuable in its contribution to a corpus of similar malware, and by extension valuable in its contributions to improvements in understanding the malware and protecting against infection. There are several costs associated with downloading the sample, in addition to the AIR system resource cost. On entry to a security lab the sample is stored, processed by automated systems, potentially shared to other labs and repeatedly scanned in the future to test for detection regression. If the sample is not detected on entry or after automated analysis, it adds to the queues of undetected samples that must be triaged for manual analysis.

We propose the following approach to deal with SSP sites once they have been identified[4]. Once an SSP site has been identified, the fetcher will gather a batch of specified size from the site and then suspend further fetching. Anti-virus scans will be run across the set of samples to determine the current detection level in terms of a percentage of the samples that would be identified as malware. If the coverage is below a specified threshold, anti-virus detections will be updated to bring detection for this batch of samples above the threshold. Once this is complete, harvest another batch of samples and repeat the process. Once a batch of samples receives a coverage value over the desired threshold upon entry to the labs, suspend or discontinue fetching the SSP site and move on to another SSP site.

5.4 Blacklisting

To detect IP blacklisting we look for changes in the HTTP code responses received by different clients harvesting the same URL. We expect that a high volume fetcher will begin to receive negative responses from malicious web servers while low volume fetchers will continue to retrieve malicious payloads. For these experiments we limited our definition of negative responses to: HTTP 500 responses, HTTP 404 responses, HTTP 200 responses with no content, and HTTP 200 responses with text content where we had previously received executable content. On each update to the URL summary we check for evidence of blacklisting. We determine that a site is showing evidence of blacklisting behaviour if all of the following criteria are met:

1. A low volume fetcher client has received at least one positive response (i.e., 200)
2. A high volume fetcher client has received at least one negative response (i.e., 500)
3. The most recent positive response to the a low volume client is more recent than the first negative response to the high volume client

Our blacklisting heuristic triggered on 14 malicious URLs, but in 9 cases the malicious server resumed serving the high volume client malicious payloads after the detected blacklisting event. Detailed analysis of the logs for these servers shows what looks to be intermittent server availability causing false positives in our blacklisting heuristics.

In the five other instances, we have seen a low volume client receive content for several hours after the high volume fetcher stops receiving positive responses, but the low volume fetcher also stops receiving positive responses after a period of time (4 - 30 hours later). Four of these URLs share many common attributes. They all have the same top level domain: .ru. Three of the URLs share the same path: au.exe. Three of the domains were registered on January 24, 2011, all of them were first seen by the TDG during the last week of January, and all of the detected samples harvested from these domains were detected as 'Mal/Zbot-AV' by the Sophos AV product. The most convincing evidence that this is blacklisting is the timing of the blacklisting response across the URLs. Requests from our high volume client to all four URLs stopped retuning results within a 10 minute period. This response was triggered after completing over 1500 downloads of executable content from these four URLs over a period of 7 days. Low volume fetchers running on

[4]To date, we have not evaluated the effectiveness of this approach, as it requires a substantial resource commitment from a security lab

three separate IPs were able to download malicious content from these URLs after the blacklisting event.

We feel this is indisputable evidence of blacklisting from this set of malware download centers. However, aside from this single event, we have not observed systematic or statistically significant occurrences of blacklisting, despite issuing over over 1.5 million requests from our busiest high volume fetcher.

6. DISCUSSION

Our evaluation of update frequency reveals a wide variance of update behaviours. Based on this observation, we conclude that that crawlers targeting malware download centers should classify URLs based on their update behaviour and handle each behaviour type separately. It is particularly important to quickly differentiate server side polymorphic sites from periodic updaters, as algorithms tuned to optimize crawling of periodic updaters will fail when applied to server side polymorphic sites. We go on to show that the fetch interval for periodically updating sites should be updated dynamically for each URL based on observed update behaviour. The evaluation of our proposed algorithm to handle periodically updating sites showed that small changes to the refetch scheduler correction function can have a significant impact on the number of new samples retrieved, as well as the system resources being used. The tuning of these algorithms provides a simple formula to allow system maintainers to balance the objectives of sample collection and minimized resource utilization.

The trend of increasing URL volumes is likely to continue, making it important for AIR systems to be able to tune their behaviour based on current system load, and to employ intelligent fetching strategies to operate more efficiently. The evaluation of the proposed correction functions suggests that the most effective correction function and coefficient (i.e., X, -X) combination depends on the α value. Power functions are not appropriate unless sample collection is a low priority. A percentage based correction function provides the best results of the three functions when α is between 3 and 62, after which point a linear correction function provides the best results. Irrespective of the correction function, choosing a larger positive coefficient will reduce the number of fetches, whereas increasing the negative coefficient will favour sample collection. This suggests that the negative coefficient should be increased in proportion to α. We have not yet evaluated techniques for choosing the α value; we discuss our plans for such research in Section 7.

Another factor to consider when determining the refetch interval of an AIR system is the potential for blacklisting. While our study did not find strong evidence of systemic IP blacklisting among malware download centers, this does not mean IP blacklisting is not an issue facing AIR systems. Our experiments were short and limited in the number of sites visited, whereas AIR systems in security labs must process all new suspected URLs, reprocess a subset of those URLs for weeks and months into the future, and typically run continuously for years. It is entirely possible that IP blacklisting by adversaries does occur, is a manual process, and the network traffic generated by the TDG was not enough to get noticed (Google crawlers process orders of magnitude more data on a daily basis than the TDG processed in four months). If an AIR system is designed with a bias toward sample collection at the expense of network resource consumption, there should be mechanisms in place to detect blacklisting, so that the maintainers of the system can respond by allocating new network resources or by changing some identifiable aspect of the crawler.

7. FUTURE WORK

Empirical evaluation and improvement of proposed algorithms. The algorithms to optimize refetch patterns for periodic updaters and server side polymorphic sites can likely be improved after studying performance of these algorithms on live systems. Time constraints prevented us from exploring further fetching optimizations based on sample similarities across sites. We speculate that malware download sites that serve the same samples from the same malware families will also exhibit similar update behaviours.

Tuning the α for the rescheduling algorithm. We introduce an α value in Section 5.2 to control the tradeoff between sample collection and resource conservation objectives. However, we quickly assigned an arbitrary value to α and moved on. We posit that the α can be determined for specific environments based on an economic assessment of the expected cost of missed samples vs the actual cost of fetching resources, similar to recent approaches to quantify the value of stolen personal data [4] and CAPTCHAs solvers [7]. In addition to calculating α for a specific environment, we plan to evaluate techniques to compute α for each URL. To do so we will estimate the relative cost of missing samples based on the current detection rate of the malware family downloaded from the URL.

Continued investigation of blacklisting behaviour. The Tachyon Detection Grid continues to operate and we plan to add checks for more types of reported blacklisting techniques. Our current approach only detects blacklisting if the the malware download center refuses HTTP connections; there are many other blacklisting techniques that would currently go undetected.

8. ACKNOWLEDGEMENTS

The authors would like to thank SophosLabs for the data feeds and expertise, PlanetLabs for the use of their infrastructure, and the anonymous reviewers of the paper.

9. REFERENCES

[1] AV Tracker. http://avtracker.info/, February 2011.

[2] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, July 2003.

[3] Fantomas spiderspy. http://searchbotbase.com/, February 2011.

[4] J. Franklin, A. Perrig, V. Paxson, and S. Savage. An inquiry into the nature and causes of the wealth of internet miscreants. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31*, pages 375–388. ACM, 2007.

[5] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: An empirical analysis of spam marketing conversion.

In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*, pages 3–14, Alexandria, Virginia, USA, October 2008.

[6] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy. A crawler-based study of spyware in the web. In *Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, California, USA*, 2006.

[7] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: CAPTCHAs: understanding CAPTCHA-solving services in an economic context. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, pages 28–28, Berkeley, CA, USA, 2010. USENIX Association.

[8] Y. Niu, Y.-M. Wang, H. Chen, M. Ma, and F. Hsu. A quantitative study of forum spamming using context-based analysis. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS)*, pages 79–92, San Diego, CA, February 28 - March 2, 2007.

[9] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iframes point to us. In *USENIX Security Symposium*, pages 1–16. USENIX Association, 2008.

[10] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The ghost in the browser analysis of web-based malware. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[11] Google safebrowsing API. http://code.google.com/apis/safebrowsing/, February 2011.

[12] R. Sherstobitoff. Server-side polymorphism: Crime-ware as a service model (CaaS). *Information Systems Security Association Journal*, 6(5), 2008.

[13] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. T. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, California, USA*, 2006.

[14] Webmasters cloaking forum. http://www.webmasterworld.com/forum24/, February 2011.

[15] M. Wood. Turning scareware devious distribution tactics into practical protection mechanisms. http://nakedsecurity.sophos.com/2011/02/14/scareware-distribution-tactics-practical-protection-mechanisms/, February 2011.

[16] B. Wu and B. D. Davison. Cloaking and redirection: A preliminary study. In *In Proceesdings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, Chiba, Japan, May 10 2005.

[17] B. Wu and B. D. Davison. Detecting semantic cloaking on the web. In *Proceedings of the 15th Annual World Wide Web Conference (WWW)*, pages 819–828, Edinburough, Scotland, May 23 - 26 2006.