SAM SMALL, JOSHUA MASON, RYAN MACARTHUR, AND FABIAN MONROSE

Masquerade: simulating a thousand victims

Sam Small is a PhD candidate in the Computer Science Department at Johns Hopkins University. His research interests include network security and information security for resource-constrained devices.

sam@cs.jhu.edu

Joshua Mason is a PhD candidate at Johns Hopkins University. His primary research interest is applying natural-language processing and data-mining techniques to information security.

josh@cs.jhu.edu

Ryan MacArthur has a Master of Science in Security Informatics from Johns Hopkins University. His current research deals with discovering new malware and finding ways to prevent it.

rpm@jhu.edu

Fabian Monrose is an associate professor in the Computer Science Department at the University of North Carolina at Chapel Hill. Prior to joining UNC, he was an associate professor at Johns Hopkins University and a member of the technical staff at Bell Labs, Lucent Technologies.

fabian@cs.unc.edu

ATTACKS AGAINST WEB-SERVER APPLICATIONS and their clients' Web browsers have recently increased in popularity. These automated attacks rely not only on weaknesses in a wide variety of applications but also on identifying potential victims with popular search engines. We have built a system that attracts these attacks by representing many different victims in Web searches and simulating their behavior when attacked. Its deployment has succeeded in attracting hundreds of thousands of attacks in a two-month period.

As time passes and system security improves, familiar attack vectors become less common and new, more successful techniques emerge. For instance, the increased presence of end-user firewalls, NAT (network address translation), and better operating systems security have reduced the presence and potency of malware worms, despite their broad notoriety just five years ago. Also, the monetization of vulnerabilities and stolen personal data motivates more clandestine attacks. Consequently, it is no longer common for attackers to write worms that randomly scan the Internet for potential victims, and attackers are forced to shift their strategy to promote wide-scale malware infection accordingly. Increasingly, attackers now covertly compromise servers, lying dormant except to covertly infect their visitors as well. This method of infection is commonly referred to as a *drive-by download* and its victims are typically Web servers running vulnerable software and personal computers with browser vulnerabilities [1]. Left undetected, this method of infection affords attackers the opportunity to control large networks of compromised machines.

Crawling for Victims

The recent increase in this underhanded tactic, infecting visitors to compromised Web sites and automatically installing executables on the victims' machines unbeknownst to them, was well documented by Provos et al. [2]. Their investigation showed that during a 10-month period, more than 1% of all queries to the Google search engine yield at least one recommended URL that resolves to a Web server suspected of hosting malicious content. After categorizing a subset of the malicious URLs with the Open Directory Project [3], the researchers discovered that, although user browsing be-

havior can affect the likelihood of encountering such URLs, Web servers in all major content categories are affected. Among other causes, Web servers are often compromised via unreported vulnerabilities in insecure third-party Web applications (e.g., popular online discussion forum software, administrative interfaces, and content management systems).

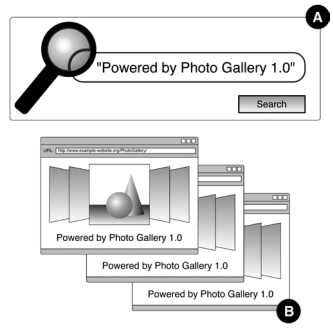

FIGURE 1: SEARCH WORMS AUTOMATICALLY IDENTIFY HOSTS RUNNING VULNERABLE WEB APPLICATIONS BY (A) USING DIRECTED SEARCH-ENGINE QUERIES THAT (B) REVEAL VISIBLE INSTALLATIONS. SOMETIMES THESE QUERIES ARE AS SIMPLE AS DEPICTED IN THE FIGURE, ALTHOUGH AT OTHER TIMES THEY ARE MORE ADVANCED, TAKING ADVANTAGE OF OBSCURE SEARCH-ENGINE FEATURES.

Attackers find Web applications an attractive target for many reasons. A unique combination of insecure or amateur development, far-reaching network visibility, and the opportunity to further infect Web site visitors provides attackers with strong motive to target Web applications. Moreover, Web applications are notoriously insecure. The SANS Institute has reported that, from November 2006 to October 2007, Web application vulnerabilities were responsible for just under half of all reported vulnerabilities and that hundreds of new vulnerabilities and exploits in both commercial and open-source Web applications are reported each week [4].

Worse yet, under some circumstances, by abusing popular search engines attackers can easily identify Web servers hosting vulnerable Web applications. As depicted in Figure 1, if an attacker has discovered a vulnerability in version 1.0 of a Web application named Photo Gallery, the attacker can identify Web servers running the application (i.e., potential victims) by simply submitting the query "Powered by Photo Gallery 1.0" to a search engine. If we assume that the software always displays the phrase in question, the search engine will likely identify URLs to these Web servers, which the attacker then attempts to compromise. As demonstrated in Figure 2, these attacks are typically constructed the same way.

When automated, this attack strategy can be quite virile. These attacks enable fast propagation

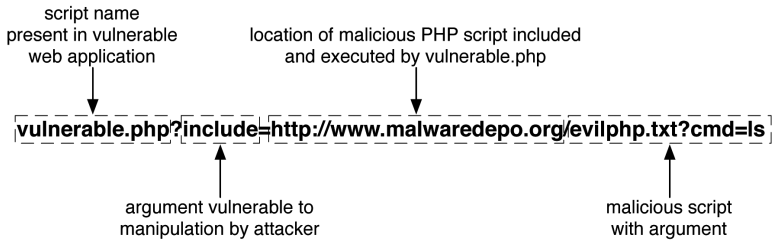

FIGURE 2: AN EXAMPLE OF A PHP REMOTE INCLUSION ATTACK, A VERY COMMON ATTACK OWING TO THE POPULARITY OF PHP AS A DEVELOPMENT LANGUAGE AND THE FREQUENTLY DEMONSTRATED INSECURITY OF WEB APPLICATIONS

owing to their ability to quickly and accurately identify potential victims (i.e., generate a hit list). The automated variant of this attack is referred to as a *search worm* [5].

Although it is well known that this category of attacks has recently become more popular, little is known about the scope of this growing trend. In response, researchers have begun to develop low-interaction, *Web-based honeypots* to monitor automated attacks directed at vulnerable Web applications by extending the scope of more traditional, daemon-centric honeypots [6]. Historically, honeypot systems have significantly helped researchers to identify the

extent to which automated network attacks take place, to identify new patterns in malware, and to generate signatures for security appliances and applications [7].

However, merely adopting the tools and techniques typically used to monitor traditional automated attacks (i.e., random-scanning worms) would be ineffective. Herein lies a significant challenge: To effectively monitor meaningful attacks, a Web-based honeypot must be indexed under the queries used by attackers; however, attacks for which queries (or signatures) are already known have diminished utility to researchers, since they are often abandoned by attackers once disclosed and patched.

For instance, one might consider instrumenting the most common Web applications to create a Web-based honeypot. This approach illustrates a number of problems. First, many applications are neither free nor open source. Second, the sheer diversity and availability of Web applications across the Internet render this approach insufficient, inefficient, and intractable as a general approach. Simply put, it is too difficult to predict which of the thousands of widely installed Web applications attackers will target next.

The Great Pretender

To address these limitations and quantify the scope of this threat, we developed a method that, when disguised as a Web server, simultaneously and efficiently represents a wide range of Web applications [8]. Its implementation elicited over 368,000 attacks from more than 29,000 unique hosts, which targeted hundreds of distinct Web applications in under two months. The observed attacks include several exploits detected the same day the related vulnerabilities were disclosed publicly. Furthermore, an analysis of the captured payloads highlights some interesting insights into current malware trends and the post-infection process.

To provide some grasp of its function, consider the automated voice-driven systems commonly used to handle customer-service phone calls. These systems prompt customers to state the purpose of their call so that each is directed to either an appropriate service department or a relevant recorded response. The best of these systems are remarkably effective despite the unique speaking characteristics of each user and the diversity of spoken words and phrases with similar meanings. Behind the scenes, such systems employ an amalgam of technologies built on concepts developed by the natural-language processing and machine-learning research communities.

Scientists train such systems to automatically evaluate and estimate the meaning of requests in real time, using large sample sets of requests and responses. For our example, the requests in these sets likely represent diverse diction and speech patterns. The content and meaning of the sample data are known a priori and treated as catalysts; therefore, each system's response can be conditioned on known responses to a known catalyst. Once this training process is complete, responses to unobserved requests (such as those posed by a new customer) are often estimated by, for instance, identifying a request's most similar counterpart from the sample set and selecting its response. This entire process is referred to traditionally as *supervised learning,* and it is in this manner that many systems are often able to satisfy online requests accurately [9].

Our method is similarly built using a statistical response-estimation engine. Unlike the previous example, however, our approach produces responses to protocol requests rather than vocal requests. Rather than determine whether the demands of two customers are similar using only information from sample requests, we instead consider whether two network requests are similar. In this case, requests come from the search-engine spiders that index Web pages and the automated attacks launched by search worms. The produced responses are aimed at ultimately enticing search worms to contact our "honeypot," allowing us to observe the scope and nature of such attacks.

UNDER THE HOOD

For new and unfamiliar requests, simulating a response requires identifying which sample requests are most similar. During initialization, similar sample requests are partitioned using a variant of the k-means clustering algorithm so that, generally, each cluster loosely represents a

specific type of application request [10]. New requests are then paired with the most representative cluster. The metric used to quantify the difference between each pair of requests is called term-frequency/inverse document frequency cosine-similarity, or simply TF/IDF distance [11]. This metric is an attractive choice because its construction is purely statistical and does not rely on any protocol-specific knowledge. TF/IDF distance is also commonly used to match queries with relevant documents in information retrieval and data-mining applications.

Assigned to each cluster is a smoothed *n*-gram language model [12]. Each cluster's language model is trained with the set of responses that correspond to each request in the cluster. Then, when handling an unobserved request, the language model paired with the cluster it best fits generates a dynamic (and statistical) response. Many messages contain session-specific fields that match between requests and response pairs (e.g., sequence numbers or session identifiers). When this behavior can be inferred, we post-process responses to satisfy such dependencies by using byte-sequence alignment algorithms [13].

VALIDATION

The success of this approach is predicated on a simple assumption in analogy with the example provided earlier: To elicit protocol interaction, the protocol responses artificially produced for online network requests must be acceptable to network agents much the way the responses produced by automated telephone systems must be accepted by its callers to guarantee their participation. To frame this assumption differently, consider that such a system is built and no one attacks. Some form of validation is necessary to determine whether our assumptions simply do not hold or the method is flawed, whether the data sets used to train the system are unrepresentative or (less likely) whether the attackers have given up.

To assess whether these techniques (typically used with natural languages) could synthesize network traffic and elicit Web-based attacks is a challenging problem in its own right. After all, there is hardly a rigid or universal definition governing the acceptability of HTML. Although standards do exist, HTML is overwhelmingly parsed by best-effort means. However, since these techniques represent knowledge derived only from the inferences and estimation encapsulated in sample data, we reason that the method can be validated under the strict guidelines of a less-forgiving protocol such as DNS. Again, since none of the methods used to simulate responses is protocol-specific and relies only on inferences from sample data, the method is fundamentally protocol-agnostic.

Unlike HTTP, the DNS protocol has a fixed binary format and its correctness is well defined for all messages, providing us with a quantitative benchmark for validation. First, we used DNS traffic produced by our colleagues over the course of three months as the training data set. We then generated and submitted 20,000 random queries to what is essentially an impostor DNS server and evaluated the correctness of its responses. The experiment confirmed our assumption: Valid responses are produced with a success rate that correlates positively to the size and diversity of its training set.

In-the-Wild Evaluation

Earlier, we asserted that the automated exploitation of Web applications poses a serious threat to the Internet. To support this hypothesis we built a system to catch and detect search worms using the techniques previously described. Since building a useful supervised learning system requires representative sample data, we obtained a list of over 3000 of the most searched queries to Google by known search worms and queried Google for the top 20 results associated with each query. Our corpus is comprised of the protocol interaction captured when requesting these URLs.

As mentioned previously, search worms only target Web servers that are indexed by search engines. To artificially boost the popularity of our system, we first placed hyperlinks on several popular pages. Additionally, we were able to expedite the indexing process by disclosing the existence of a minor bug in a common UNIX application to the Full-Disclosure security

mailing list. Bulletins from this list are mirrored on several high-ranking Web sites and are crawled extensively by search-engine spiders.

Shortly after being indexed, search worms began to attack at an alarming rate, with attacks rapidly increasing over a two-month deployment period. The results are shown in Figure 3. The sheer volume of attacks is shocking: In total, we observed well over 368,000 attacks targeting just under 45,000 unique scripts. During this time, we also recorded the number of times Google indexed our system (in total, just shy of 12,000). As expected, our results indicate a positive correlation between the index rate and the attack rate. The attacks we captured also reveal that many search worms target multiple vulnerabilities and distinct Web applications in tandem. In many cases, different worms attempt to inject malware hosted on the same remote servers.

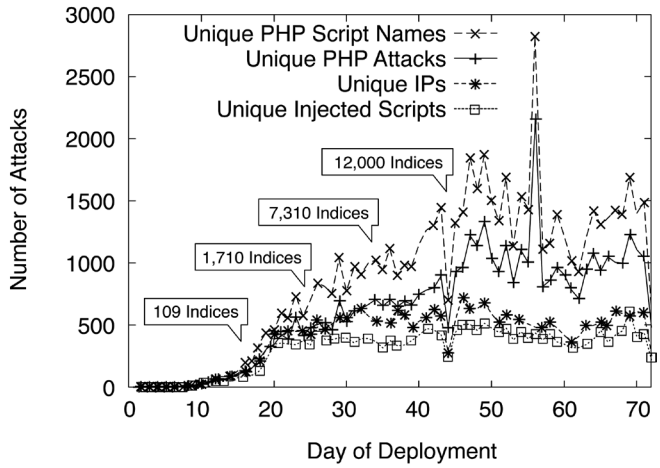

FIGURE 3: DAILY PHP ATTACKS ANNOTATED WITH SEARCH-ENGINE INDEX RATE. IN TOTAL, WE OBSERVED OVER 368,000 ATTACKS IN JUST OVER 2 MONTHS. THE VALLEY ON DAY 44 IS DUE TO AN 8-HOUR POWER OUTAGE. THE PEAK ON DAY 56 IS BECAUSE TWO BOTS LAUNCHED OVER 2,000 SCRIPT ATTACKS.

In general, classifying the number of unique Web applications targeted by search worms is difficult, because many of the targeted script names are ubiquitous (e.g., index.php). In these cases, search worms are either targeting a vulnerability in one specific Web application or arbitrarily attempting to inject malicious scripts. Despite this difficulty, we were able to map the content in our sample data to over 500 unique Web applications. We then linked the attacks themselves back to 295 distinct Web applications, which is indicative of the overall diversity of targets being attacked.

EMERGENT THREATS

Although the original intent of our deployment was to elicit attacks from search worms exploiting known vulnerabilities, we became indexed under broader conditions owing to the variability of our sample data. As a result, we sometimes attracted attacks targeting undisclosed vulnerabilities. For instance, according to *milw0rm*, over 65 PHP remote-inclusion vulnerabilities were released during the time span of our deployment. Since our deployment used the same training data for its entire duration, we know that captured attacks against these vulnerabilities were not explicitly represented by data in the training set.

Nonetheless, we witnessed several emergent threats, because many of the original queries used to bootstrap the supervised learning process were generic, representing a wide number of applications. During the deployment, we identified more than 10 attacks against vulnerabilities disclosed after its launch (see Table 1); thus, these attacks were not explicitly represented by the training data. It is unlikely that we witnessed these attacks simply because of arbitrary attempts to exploit random Web sites; indeed, we never witnessed many of the other disclosed vulnerabilities being attacked.

Disclosure	Attack	Signature
Day 9	6 days later	/starnet/themes/c-sky/main.inc .php?cmsdir=
Day 26	2 days later	/comments-display-tpl.php ?language_file=
Day 27	Same day	/admin/kfm/initialise.php ?kfm_base_path=
Day 30	Same day	/Commence/includes/db_connect .php?phproot_path=
Day 33	Same day	/decoder/gallery.php?ccms _library_path=

TABLE 1: SOME OF THE ATTACKS TARGETING VULNERABILITIES THAT WERE UNKNOWN AT THE TIME OF DEPLOYMENT. IN AT LEAST 3 CASES, WE OBSERVED ATTACKS ON THE SAME DAY THAT THEIR VULNERABILITIES WERE DISCLOSED.

Given the frequency with which these types of vulnerabilities are released, we argue that a honeypot without dynamic response generation will likely miss an overwhelming amount of attack traffic. In the attacks we witnessed, several search worms began attacking vulnerabilities on the same day as their disclosure! An even more compelling case for our architecture is embodied by the attacks against the vulnerabilities that have not yet been disclosed. We believe that the potential to identify these attacks exemplifies the real promise of this approach.

PAYLOAD ANALYSIS

To better understand what the post-infection process entails, we conducted a rudimentary analysis of the remotely injected malicious scripts and its malware. We analyzed malware using a basic sand-boxed environment that hooks system calls and libraries to discover malware functionality. Table 2 provides an abbreviated summary. Overwhelmingly, the attacks attempt to install PHP Web-based shells. These provide attackers with a direct and easy way to arbitrarily control infected systems. As is now typical, many of the scripts are obfuscated, erase evidence of infection, and perform automated self-updates. In some cases, the malware profiled the systems (e.g., by copying /etc/passwd and performing local scans). To our surprise, only eight scripts contained functionality to automatically obtain root access.

Script Classification	Representation (%)	
PHP Web-based shells	32	
Echo notification	22	
PHP bots	14	
Spammers	13	
Downloaders	7	
Perl bots	5	
Email notification	3	
Text injection	1	
Information farming	<1	
Uploaders	<1	
Image injection	<1	
UDP flooders	<1	

TABLE 2: CLASSIFICATION OF OBSERVED MALWARE. WE ANALYZED MORE THAN 2,600 MALICIOUS SCRIPTS AND INSTANCES OF MALWARE ORIGINATING FROM AUTOMATED ATTACKS.

As can be expected, we also observed several instances of spamming malware using email addresses pulled from the databases on infected machines. For Web servers hosting applications such as phpBB, this can be highly effective, because most users enter an email address during registration. Cross-checking the IP addresses of these worms with the Spamhaus project revealed that roughly 36% of them currently appear in its spam blacklist [14]. Lastly, we note that although we observed what appeared to be over 5,648 unique injection scripts from distinct worms, nearly half of them belonged to orphan botnets. These networks no longer have a centralized control mechanism and the remotely included scripts are no longer accessible. They are, however, still responsible for an overwhelming amount of our observed HTTP traffic.

Conclusions

Our work uses a number of multidisciplinary techniques to generate dynamic responses to protocol interactions. We demonstrate the utility of our approach through the deployment of a dynamic content generation system targeted at eliciting attacks against Web-based exploits. During a two-month period we witnessed an unrelenting barrage of attacks from attackers that scour search-engine results to find victims (in this case, vulnerable Web applications). The attacks were targeted at a diverse set of Web applications and employed a myriad of injection techniques. We believe that the results herein provide valuable insight into the nature and scope of this increasing Internet threat.

For real-world honeypot deployments, detection and exploitation of the honeypot itself can be a concern. Clearly, our system is not a true Web server and, like other honeypots, it can be trivially detected using various fingerprinting techniques. The fact that our Web honeypot can be detected is a clear limitation of our approach, but in practice it has not hindered our efforts to characterize current attack trends. The search worms we witnessed all appear to use search engines to find the identifying information of a Web application and attack the vulnerability upon the first visit to the site without verifying its presence, presumably because explicit verification reduces the rate of infection. Nonetheless, dealing with multi-stage attacks is an area of future work. For information, see our publication from USENIX Security '08 [8].

REFERENCES

[1] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, "The Ghost in the Browser: Analysis of Web-based Malware," *USENIX Workshop on Hot Topics in Botnets (HotBots)* (2007).

[2] N. Provos, P. Mavrommatis, M.A. Rajab, and F. Monrose, "All Your iFRAMEs Point to Us," *Proceedings of the 17th USENIX Security Symposium* (July 2008), pp. 1–15.

[3] Open Directory Project: http://www.dmoz.org/.

[4] SANS Institute Top-20 2007 Security Risks (2007 Annual Update), November 2007: http://www.sans.org/top20/.

[5] N. Provos, J. McClain, and K. Wang, "Search Worms," *Proceedings of the 4th ACM Workshop on Recurring Malcode* (2006), pp. 1–8.

[6] The Google Hack Honeypot, 2005: http://sourceforge.net/projects/ghh/.

[7] N. Provos, "A Virtual Honeypot Framework," *Proceedings of the 12th USENIX Security Symposium* (August 2004), pp. 1–14.

[8] S. Small, J. Mason, F. Monrose, N. Provos, and A. Stubblefield, "To Catch a Predator: A Natural Language Approach for Eliciting Malicious Payloads," *Proceedings of the 17th USENIX Security Symposium* (2008), pp. 171–183.

[9] Machine learning: http://en.wikipedia.org/wiki/Machine_learning.

[10] J.B. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1 (Berkeley, CA: University of California Press, 1967), pp. 281–297.

[11] G. Salton and C. Buckley, "Term-weighting Approaches in Automatic Text Retrieval," *Information Processing & Management* 5(24):513–523 (1988).

[12] I.H. Witten and T.C. Bell, "The Zero-frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression," *IEEE Transactions on Information Theory* 37(4):1085–1094 (1991).

[13] S.B. Needleman and C.D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," *Journal of Molecular Biology*, 48:443–453 (1970).

[14] The Spamhaus Project: http://www.spamhaus.org/.