2009 Eighth IEEE International Symposium on Network Computing and Applications

Attribute-based Prevention of Phishing Attacks

Michael Atighetchi
BBN Technologies
Email: matighet@bbn.com

Partha Pal
BBN Technologies
Email: ppal@bbn.com

***Abstract*—This paper describes a set of innovative attribute-based checks for defending against phishing attacks[1]. We explain a number of anti-phishing algorithms implemented as plugins and highlight which attributes of phishing sites they consider. To assess the effectiveness and applicability of this prototype, we performed extensive experimental testing. We present a fully automated crawling framework that we developed for testing, along with the main experimental results.**

I. Introduction

Over the past decade, online identity fraud has transformed from being a small scale criminal activity of a few unethical computer hackers to a widespread syndicated crime costing billions of dollars in damages each year.

Cyber criminals use phishing predominantly as a technique for obtaining identity-related information, such as social security numbers or bank account numbers. In a typical phishing scenario, a cyber criminal sets up a fake web site that looks similar to the login page of a target financial institution and sends out a massive amount of email to trick people into logging into the fake web site and entering personal information. The cost incurred by criminals is very low and within a short period of time they can successfully complete an attack cycle and hide their tracks. These facts have fueled the phenomenal growth of phishing attacks [1].

A vast majority of existing anti-phishing products follow a signature-based approach. Users, ISPs, and security staff of financial companies provide suspected URLs as input to a centralized blacklist service, which disseminates vetted blacklists to end clients (mostly browser extensions) for enforcement. This approach ties the effectiveness of anti-phishing products to the accuracy and timeliness of signature updates. It is impractical to assume that all active phishing sites will be reported to the centralized service on time. Vetting of the reported sites adds some amount of time delay. Therefore, attackers always have a window of opportunity during which a significant fraction of end clients operate without the protection of updated signatures.

We have developed a different anti-phishing approach in PhishBouncer which uses attribute-based checks to implement both reactive and proactive anti-phishing defenses. We claim that because of the focus on common attributes of phishing attacks (rather than specific signatures) and user behavior over time, the PhishBouncer approach does not require timely updates and therefore does not suffer the problems associated with them. Since attribute checks operate on a wide variety of phishing attack instances by looking at generic features and gathering data autonomously if needed, we are able to significantly reduce the amount of pushed content to the end systems and thereby reduce a large part of the operational cost of anti-phishing services. Furthermore, PhishBouncer provides enhanced protection against previously unknown phishing attacks as in most cases some common attributes stay invariant.

[1]This work was supported by HSARPA under contract number NBCHC050096.

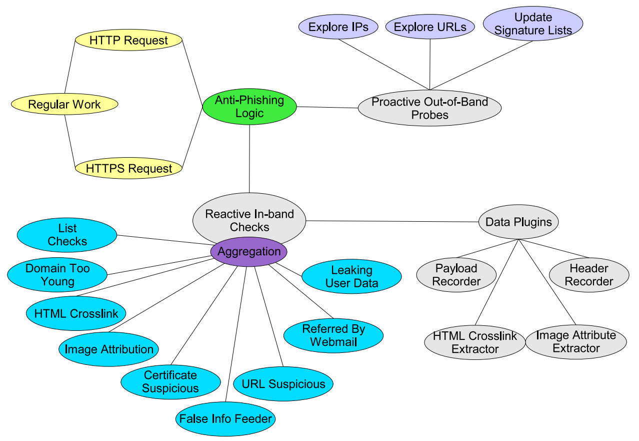

Fig. 1. Use case diagram showing all currently implemented checks, probes, and dataplugins.

A. Anti-Phishing Functionality

A key aspect of the PhishBouncer approach is its plugin framework, which provides a flexible way for applying custom adaptive and reactive logic to HTTP(S) streams. All anti-phishing logic is in fact implemented as a set of plugins. We divide plugins into three broad classes based on their role in the overall control flow and threading logic. *Dataplugins* are called on every HTTP request and associated response to perform analysis on header and payload data. *Checks* execute sequentially on HTTP requests entering the proxy and decide whether to accept the request, reject the request, or set a numeric value $0 < w < 100$ together with a typed choice (acceptPref, rejectPref) to indicate the confidence and choice selection. In contrast to Checks and Dataplugins which only execute reactively triggered by web browser requests, *Probes* allow us to embed proactive behavior into the proxy.

Figure 1 displays a use case diagram for the proxy. As part of performing regular work, a user may visit web sites and therefore generate HTTP and HTTPS requests. These requests are intercepted by the proxy and subjected to anti-phishing logic. The current implementation of PhishBouncer contains a set of nine anti-phishing checks and their supporting dataplugins. While many of these checks have been studied in the literature and variations have been implemented before, our work provides support for structured combination of various checks to form an overall effective defense.

The *Suspected by ImageAttribution* check and its associated data plugins look for image similarities between sites the user has previously registered with PhishBouncer and sites the user is trying to go to. If the images are similar but the domains are different, it is an indication that the user is trying to go to a phishing site that either displays back a local copy of the image or links to the real image. PhishBouncer keeps hashes of images for all registered sites and implements a hash-based comparison algorithm. SpoofGuard [2] implements a similar check but only compares images against a static set of fixed images for well-known domains.

The idea behind the *Domain Too Young* check is to flag sites which are hosted on very young DNS domains, as phishers frequently create mirror domains such as bancofbar1.com instead of bankofbar.com and move on to new domains (bancofbar11.com) as currently active ones get taken down by ISPs. During the Domain Too Young check, PhishBouncer performs a WHOIS lookup for outgoing TCP connections to DNS names and IP addresses. After retrieving the WHOIS record, it checks for the age of the domain name and prompts for user feedback if the age falls below a certain threshold. The WHOIS entries are cached locally in PhishBouncer to increase performance. One of the biggest challenges we encountered during implementation of this check is that there is no standardized protocol for finding the correct WHOIS server for a given domain name. We implemented heuristics and mappings similar to the WHOIS Unix command, which mitigated some of WHOIS server lookup problems. The next problem had to do with reliable parsing and data-extraction of creation time data returned by various WHOIS servers around the world. Since WHOIS requests are made in line with HTTP requests, the overhead of WHOIS lookups directly adds to the overall round trip HTTP request latency. We decided on a relatively small timeout of 200ms for timing out WHOIS requests, which may cut out a large number of slow WHOIS servers. Recent work in machine learning [3] has also used the age of domains as a feature to classify phishing emails.

The *HTML Crosslink* check looks at responses from non-registered sites and counts the number of links the page has to any of the registered sites (in particular image cross-links). A high number of cross-links is indicative of a phishing site.

The key idea behind the *False Info Feeder* check is that a phishing site is likely to indiscriminately accept user supplied data, since it has no way of knowing whether the user supplied data is valid or not. This can be used to detect the presence of phishing sites. For instance, PhishBouncer may decide to send a bogus user name and password combination to a site that has already been declared suspicious by previous checks. If the site does not reply with an error, it further increases the sites phishyness factor[2]. The overall idea of the False Info Feeder is similar to the main concept behind the PHONEY prototype [4], except that PhishBouncer doesn't rely on the assumption that the phishing attacks were initiated by email. Therefore, PhishBouncer covers are wider range of use cases, including phishing links communicated over IM systems.

The *Certificate Suspicious* check validates site certificates presented during SSL handshake and extends the typical usage by looking for Certification Authority (CA) consistency over time. The basic idea of this check is based on the observation that companies rarely change the CA used for signing their site certificates. We can use this fact to flag sites that present certificates signed by a different CA than expected. An associated probe proactively monitors registered sites to keep the set of expected CAs current.

The Certificate Suspicious check tries to assert the following conditions: 1) Certificate validity - certificate timestamp is in current time range. 2) Trusted CA - certificate is signed by a trusted CA. Currently, the list of the trusted CA is imported from that of the JRE. 3) Host Name of the server equals the CN field of the certificate. 4) The CA's DN field in the certificate matches that of the history data base. 5) Crypto signature algorithms run error-free.

For conditions 1 and 5, traffic to the site is automatically blocked when the check reports a problem, since they are clear signs of attack. For conditions 2, 3, and 4, the user will be prompted to either accept this certificate or to reject the certificate. Once the certificate is accepted, this certificate will be automatically accepted for that site until PhishBouncer gets restarted to increase user satisfaction. Likewise, once the certificate is rejected, it will be rejected until restart.

In addition to these checks, we have implemented many previously known and published algorithms as plugins, including the *URL Suspicious* check which tries to identify phishing sites by looking at the characteristics of the URL they present (puny-encoding [5], edit distance); the *Leaking User Data* check which searches through outgoing traffic for sensitive user information such as social security numbers; the *Phish Signature Match* check which compares URLS against a phishing signature feed; and the *Referred By Webmail* check which looks at the Referer header of HTTP requests and declares requests suspect (with low confidence) if they originate from a web-based email provider.

If an HTTP request emerges from the chain of checks together with a vector of check results, it will enter the Aggregation plugin. Aggregation of multiple check results plays an important role as we use it to increase true positives and decrease false positive rates via a multi-factor weight function. PhishBouncer combines the outcome of individual

[2]As an interesting extension, PhishBouncer could also send fake credit card numbers by implementing a credit card number generator that not only passes the rough number check but also guarantees that the number generated is not valid.

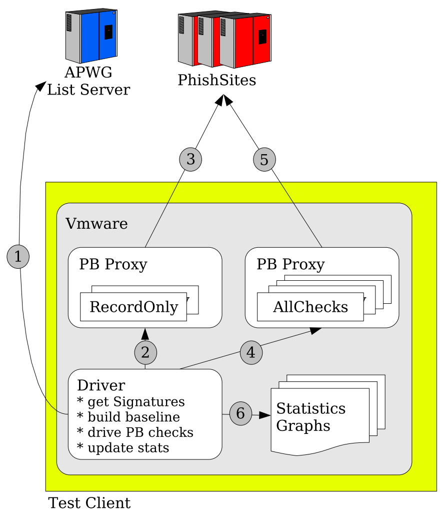

Fig. 2. Experimental setup for validating anti-phishing technologies.

checks in such a way that multiple weak suspicions have a multiplicative effect.

We use the following aggregation formula for computing the aggregate value V:

$$V = \sum_{t=1}^{t=3} O_t * T_t(w_{1..i}) \tag{1}$$

Each term T_t is an aggregation term over the distributions of i fired checks, where T_1 averages over single weights, T_2 averages over pair-wise combinations, and T_3 averages over three way combinations. O_t is a term weight between 0 and 1 specified via the administration console, and $0 < w_i < 100$ is the confidence value of an individual check with $w_i = 99$ expressing near certainty that the request goes to a phishing site.

In most of our experiments, we set up O_t to a value such that a single check with a high value of $w_i = 100$ does not cause a rejection of the request. We further calibrated the system to start rejecting requests as soon as two checks fire, given that their respective values $w_i, w_j > 1$.

It turns out that empirically determining a good set of w_i and O_t is non-trivial. It is probably better to define rules that take into account the number of fired checks or learn accurate values via statistical machine learning during off line training.

II. Experimental Validation

To evaluate the effectiveness of the attribute-based checks, we developed a test framework which can evaluate a generic anti-phishing technology against the latest live phishing sites[3] and performed evaluation by subjecting the checks to 1400 known phishing sites.

[3]We currently use the feed from APWG, but the framework can be plugged into other feeds as well.

A. Experimentation Setup

Figure 2 displays the main component used in the experimentation process together with an experimentation work flow.

In the first step, the automated driver obtains the latest set of URLs from a list service such as the APWG phish feed and builds a differential to the previously downloaded set so that only unseens URLs are processed.

For each new URL_X, the driver proceeds to start up a PhishBouncer proxy with all anti-phishing checks disabled and maximum logging enabled in step 2 (”Pb Proxy RecordOnly”) in Figure 2). Next, the driver starts a Firefox web browser configured to use the recordonly proxy and instructs Firefox to display the phish URL URL_X. The proxy records all headers and payloads (including Javascript code segments, images, HTTP headers, SSL handshakes) together with timing information for post-processing (3).

After building the baseline, the driver starts a proxy instance with all checks enabled (4), and again starts an associated Firefox instance to display URL_X (5). This time, multiple checks may indicate reject preferences for requests and all check results are logged. Note that is presently a limitation of this test harness that it cannot generate requests that simulate human user’s exploration of a web page and submission of data, e.g. a user logging into an account. This implies that checks that are based on user specific browsing pattern and data are not exercised in this test harness.

To conclude the cycle, the driver performs post-processing to extract metrics from the set of gathered log files and filter out invalid URLs. A URL is filtered if it a) is not live anymore because the phish site has been taken down already (resulting in 4XX and 5xx error codes) b) causes errors because other sites it depends on have changed c) causes errors induced by the archiving framework. The filtering is solely performed on the basis of log files from the recordonly proxy instance in order to minimize any impact of the anti-phishing checks on the selection process. The driver then proceeds to extract important statistics from the proxy log files, including check scores and results, processing latencies, and interesting check-specific metrics, e.g., the domain age. Finally, the driver calls the R [6] statistics package to group data, perform statistics, and plot results.

B. Preliminary Results

Figure 3 displays the check results as a bar plot. The y axis displays the percentage of runs (between 0 and 100), where a run consists of visiting a phishing URL. The x axis divides the outcome of a check into 4 categories: *accept* (*reject*) if a check clearly accepted (rejected) the URL without further checks and *accept_pref* (*reject_pref*) if a check didn’t (did) declare the URL suspicious. Each bar in a column represents a single check, and the bars within a column are ordered from left to right in correspondence with a top-down read of the check names from the agenda.

Since a single run may cause a large number of HTTP requests, this diagram aggregates the outcomes of individual

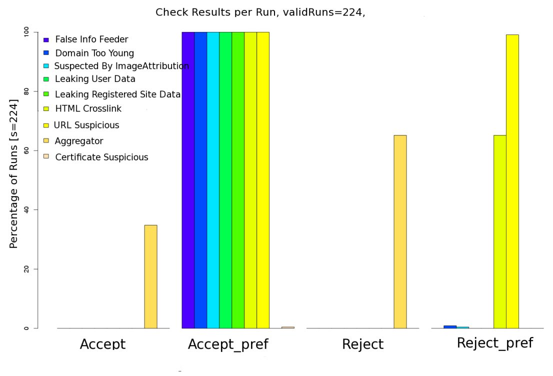

Fig. 3. Distribution of check results as a fraction of runs.

requests as follows. We count a run in the accept_pref (reject_pref) column of a check if that check flagged at least one request as accept_pref (reject_pref). Note that a run may be counted in both columns as shown for the URL Suspicious check. We perform a similar computation for the reject column of a check in that a run gets counted as rejected if the check rejected at least one request from that run. Because the Pb proxy terminates the run as soon as it hits a reject, we further compute the accept percentage as the total number of runs - the number of rejects.

The accept and reject columns are only populated by the Aggregator checks, which tells us that none of the definite checks (such as Certificate Suspicious or URL Blacklisted) fired. Furthermore, the Aggregator bars indicate that 65% of the URLs were rejected by the Aggregator.

The accept_pref and reject_pref columns allow us to drill down and decompose the aggregated decision into individual base check results. The main reason for rejection of URLs seems to stem from reject_pref values from the URL Suspicious and HTML Crosslink checks, together with a small number contributed by the DomainTooYoung Suspected By ImageAttribution checks.

The False Info Feeder, Leaking User Data, and Leaking Registered Sites Data checks operate on HTTP POST requests that submit user data to remote websites. As pointed out earlier, we currently don't simulate such behavior in the automated test framework, which is why these checks only have accept_pref bars.

None of the phishing URLs were HTTPS based. However, some caused generation of HTTPS requests to display images, which explain a small accept_pref bar for the Certificate Suspicious check.

A number of comment about these observations are in order. First, we realize that the size of the data set described in this paper is small and more extensive testing is needed to establish stronger confidence on the effectiveness claims of attribute checks. Furthermore, since many of the reported phishing URLs from the APWG list were not live, the remaining live sites may represent a skewed distribution and might not be representative of the set of unknown phishing sites that have not been reported yet. Another interesting observation is that the APWG list goes through spurs of URL permutations such that in a short amount of time, a very large amount of similar looking URLs may get reported, e.g., http://www.foobar1.com, http://www.foobar2.com etc. This may skew the results in Figure 3 to favor checks that are better at catching URL permutation attacks over other checks. Last, the weights on the individual checks and the aggregation formula have a large impact on this result. Although we did not calibrate those parameters directly for the dataset at hand, it is unclear whether they need to be adjusted for future datasets and how to best optimize them for the current dataset.

1) False Positive Rate: For determining the false positive rate, we utilized the previously described experimentation setup, except that we replaced the APWG signature feed with a static list of 23 commonly used commercial bank and news sites. The results were surprisingly good in that PhishBouncer didn't reject any of the 23 URLs. That said, we received multiple incident reports from users indicating that PhishBouncer had blocked access to legitimate sites. Our plans are to extend the false positive test coverage to a larger more representative set of white-listed websites and domains and to allow manual override of **all** block decisions while adding a reporting mechanism for gathering and evaluating override occurances.

2) Performance Overhead: We measured the overhead of the HTTPS interception in a lab environment and through field testing. Experimental data shows that the HTTPS proxying introduces an median overhead of less than 25%, which is well within the interquartile range of normal behavior and therefore does not noticably impact the use web surfing experience.

III. Conclusion

The preliminary experimentation results show a 65% rejection rate for zero-day attacks using only 4 types of checks. This implies that using attribute-based anti-phishing checks can provide a viable defense against phishing, complementing signature and blocklist based defenses. End users, security practitioners ,and anti-phishing researchers can benefit from using these algorithms. Toward that end, we have made the proxy together with implementation of the attribute-based checks and automated testing framework available as open-source software at http://www.phishbouncer.com.

References

[1] "Apwg phishing activity trends february 2007." [Online]. Available: http://www.antiphishing.org/reports
[2] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. C. Mitchell, "Client-side defense against web-based identity theft," in *11th Annual Network and Distributed System Security Symposium (NDSS '04)*, 2004.
[3] I. Fetter, N. Sadeh, and A. Tomasic, "Learning to detect phishing emails," in *Proceedings of the 16th International Conference on World Wide Web*, 2007.
[4] M. Chandresekaran and R. Chinchani, "Phoney: Mimicking user response to detect phishing attacks," in *World Of Wireless, Mobile and Multimedia Networks*, 2006.
[5] "Punycode: A bootstring encoding of unicode for internationalized domain names in applications (idna)," http://tools.ietf.org/html/rfc3492.
[6] "The r project homepage," http://www.r-project.org/.