Poster: Lightweight Content-based Phishing Detection

Calvin Ardi*, John Heidemann†
*: Student †: Faculty
USC/Information Sciences Institute
Marina del Rey, California 90292
{calvin, johnh}@isi.edu

I. Introduction

Increasing use of Internet banking and shopping by a broad spectrum of users results in greater potential profits from phishing attacks. Phish are fake websites that masquerade as legitimate sites, to trick unsuspecting users into sharing sensitive information: credentials, passwords, financial information, or other personal information that can enable fraud. This threat is especially dire for financial services and sites involving online payment: an attacker can use stolen credentials to steal money or make fraudulent transactions.

Most browsers today detect potential phishing with URL blacklists such as the Google Safe Browsing API, PhishTank [1], Is It Phishing [2] service, and the Netcraft toolbar [3]. The browser checks each website a web user visits against a list of known bad sites that is typically cached locally and refreshed regularly. While effective at stopping previously known threats, blacklists must react to new threats as they are discovered, leaving an inevitable period of vulnerability where users are vulnerable. Attackers exploit this gap by changing URLs for phishing sites frequently.

Alternatively, whitelists can identify pre-determined websites as "known-good". Whitelists thus avoid the race to identify and add new phishing sites, but have their own delays in approving new sites, and by definition prohibits (or strongly discourages) use of sites off the list. This delay makes them too limited for many users.

Our goal is proactive detection of phishing websites with neither the delay of blacklist identification nor the strict constraints of whitelists. Our approach is to list known phishing targets, index the content at their *correct* sites, then look for this content to appear at *incorrect* sites to signal a phishing site. While prior work has visually compared good website layouts with potential phishing sites [4], we focus on the content itself. Our insight is that cryptographic hashing of page contents allows efficient bulk identification of content reuse at phishing sites.

Our contribution is to build a system to detect phish by comparing hashes of visited websites to the hashes of the original, known good, legitimate website. We implement our approach in the form of a browser extension in Google Chrome, and show that our algorithms detect a majority of phish, even with minimal countermeasures to page obfuscation. A small number of alpha users have been using the browser extension without issues for several weeks, and we will be releasing our extension and source code upon publication.

We thank Danyong Zhao for his initial exploration in developing a browser extension. This material is based upon work supported by Department of Homeland Security Science and Technology Directorate, Cyber Security Division, via SPAWAR Systems Center Pacific under Contract No. N66001-13-C-3001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of SSC-Pacific.

II. Methodology

Our system detects if a visited website is suspected phish by comparing the content of an uncertain site against content from likely phishing targets. We build a list of likely targets of phishing (the official websites of financial and e-commerce sites) based on prior phishing targets and user interest. We regularly crawl these good sites and build an index of cryptographic hashes of their content, broken up into chunks (paragraphs of text or potentially individual graphic items or other content). Upon visiting an unknown page, we hash it in the same way and efficiently check for similarity. This methodology builds upon our prior work of detecting content reuse over large portions of the web [5].

A. Identifying Phishable Content

We define phishable content as the web pages and contents of sites that are likely phishing targets. We build a set of hashes of that content as our *phishable content set*. We store this set at the client and use it to detect potential phishing sites.

After identifying targets that are likely to be candidates for phish, we crawl and chunk the front page of each candidate delimited by `<p>` and `<div>`. We then hash each chunk using SHA-1 [6] and add the hash to our phishable content set. (We ignore extremely short chunks.) We also maintain a whitelist of sites allowed to host this content. In our current use, both sets are relatively small and is stored directly in the client. Efficient techniques such as Bloom filters allow very large sets to be compressed to fixed-size storage and compared very efficiently [7].

Identifying Phishable Content

1) Crawl the front page of phishing candidate targets
2) Crawler chunks each candidate front page delimited by `<p>` and `<div>`, and hash each chunk
3) Store the hashes in the phishable content set

B. Detecting Phishable Content in the Wrong Place

We test an unknown site to detect a suspected phish by testing its content against our set of phishable content. Similar to the identification phase, we chunk the visited webpage and hash each chunk. We then compare the number of chunks that match the phishable content set. If the number is greater than a threshold, we flag the webpage as suspected phish. We implement detection in a browser extension for Google Chrome and display an alert overlay when a visited webpage is a suspected phishing site (Figure 1).

Detecting Phishable Content in the Wrong Place

1) User visits a webpage
2) Browser extension chunks the visited page delimited by `<p>` and `<div>`, and hashes each chunk
3) Extension performs the intersection of the chunks of the visited page and the phishable content set

4) If the size of the union set exceeds a threshold, the page is detected as a suspected phish, and we alert the user

III. Effectiveness of Phishing Detection

A. Evaluation of Detection Mechanism

We now evaluate the core algorithms used in the plugin. Since we do not have access to a large source of spam, we evaluate our algorithm by targeting PayPal phishing. We build a set of known good sites from recent and older home pages of PayPal U.S., U.K., and France (Sep. 2014, plus Jan. 2012 to Aug. 2013 from archive.org). Because these sites use modern HTML, we chunk each page and create a phishable content dataset of 311 distinct chunks longer than 100 characters. We also apply whitespace normalization, mapping all sequences of whitespace to a single space.

We then create a corpus of likely phish drawn from a stream of 1813 suspected and available phish drawn from from PhishTank [1] over two days, a crowd-sourced anti-phishing site. Since the lifetime of a phish is short, we automatically rip the top page for each site. We compare each suspected phish against our known good dataset and a detection threshold of greater than one chunk.

To evaluate ground truth, we manually examine the corpus and identify 132 (of the 1813) as PayPal phish attempts. We further identify 90 of the remaining sites as phish utilizing text content from PayPal. Our mechanism detects 48 (53.3%) pages that pass the detection threshold: 36 are direct rips detected with no normalization applied, and an additional 12 are detected with whitespace normalization. Table I classifies the type of techniques each phishing site uses and our detection rates.

We have not yet hardened our approach against adversarial techniques, so we miss about 47% of the phish. For example, some targets construct their sites purely from images, encode their phish using escaped JavaScript, or construct a original phish without directly ripping from the original site. Since a phishing site *must* look like the original, in principle these techniques could be countered by replay of JavaScript in a sandbox, image recognition, and other normalization. These counter-countermeasures are complementary to our approach and can be used in addition to our detection mechanism; its implementation and effectiveness are left for future work.

Without addressing adversarial techniques, our approach has a fairly high false negative rate with a sensitivity of 53.3%. However, our targeted dataset means no false positives and a specificity of 100%.

B. Experiences in Real-World Usage

We have been using the browser extension intermittently since March 1, 2015, and continuously since March 31, 2015. So far the extension works reasonably well, detecting the known phish we use for testing without affecting the speed of normal browsing operations. We have not yet seen any real phish, nor any false positives.

IV. Conclusion

Our browser extension will bring a usable and efficient means of detecting phish that doesn't rely on the continuous cat-and-mouse chase of updating URL blacklists. We see our approach as a seamless augmentation to existing anti-phishing techniques in current browsers. We will be releasing our extension and source code upon publication, and intend on improving our extension's performance and capabilities.

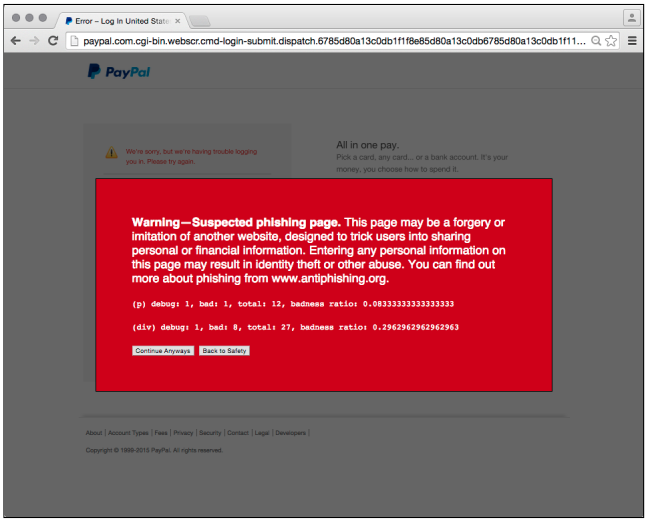

Fig. 1. Alert overlay displayed on PayPal phish.

TABLE I. Classification and Detection of PayPal Phish

Description	$\|P\|$	%	
Candidates	1813		
Other (Non-PayPal)	1681		
PayPal (image-based site, *removed*)	42		
PayPal	90	100.0	FP = 0
Direct rips (*detected*)	36	40.0	
`<p>`	12		TP = 48
`<p>` and `<div>`	36		
Whitespace norm. (*detected*)	12	13.3	
JavaScript obfuscation	7	7.8	FN = 42
Custom-styled phish with minor PayPal content	35	38.9	

References

[1] OpenDNS, "PhishTank - Join the fight against phishing," Available at: http://www.phishtank.com, 2015.

[2] V. Retro, "isitPhishing - Anti phishing tools and informations," Available at: http://www.isitphishing.org/, 2015.

[3] N. Ltd., "Netcraft extension - phishing protection and site reports," Available at: http://toolbar.netcraft.com/, 2015.

[4] W. Zhang, H. Lu, B. Xu, and H. Yang, "Web phishing detection based on page spatial layout similarity," *Informatica*, vol. 37, no. 3, pp. 231–244, 2013.

[5] C. Ardi and J. Heidemann, "Web-scale content reuse detection (extended)," USC/Information Sciences Institute, Tech. Rep. ISI-TR-692, June 2014.

[6] National Institute of Standards and Technology, "Secure hash standard (SHS)," National Institute of Science and Technology, Federal Information Processing Standard (FIPS) 180-3, Oct. 2008. [Online]. Available: http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[7] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970. [Online]. Available: http://doi.acm.org/10.1145/362686.362692