AuntieTuna: Personalized Content-based Phishing Detection

Calvin Ardi   John Heidemann
USC/Information Sciences Institute / {calvin, johnh}@isi.edu

***Abstract*—Phishing sites masquerade as copies of legitimate sites ("targets") to fool people into sharing sensitive information that can then be used for fraud. Current phishing defenses can be ineffective, with training ignored, blacklists of discovered, bad sites too slow to pick up new threats, and whitelists of known-good sites too limiting. We have developed a new technique that automatically builds personalized lists of target sites (candidates that may be copied by phish) and then tests sites as a user browses them. Our approach uses cryptographic hashing of the browser's Document Object Model (DOM) of each page, providing a zero false positive rate and identifying more than half detectable phish in a controlled study. Since each user develops a customized list of target sites, our approach presents a diverse defense against phishers. We have prototyped our approach as a Chrome browser plugin called *AuntieTuna*, emphasizing usability through automated and simple manual addition of target sites and clean reports of potential phish that include context about the targeted site. AuntieTuna does not slow web browsing time and presents alerts on phishing pages before users can divulge information. Our plugin has been used by a few users for months and is open-source.**

I. Introduction

Phish are fake websites that masquerade as legitimate sites, with the goal of tricking unsuspecting visitors to sharing sensitive information: their credentials, passwords, financial or other personal information (recently surveyed [16]). In phishing, an adversary constructs a *phishing site* from *target content* drawn from a legitimate service used by the user. The phishing site fools the user (as a Trojan horse) into disclosing information that can then be exploited for identity theft, fraud, and to compromise other services.

Phishing is an increasing threat, with widespread opportunities as general public makes extensive use of the Internet for banking and electronic commerce. This threat is especially dire for financial services and sites with online payment and their users: an attacker can use stolen credentials to steal money or make fraudulent transactions. Sophisticated attacks also target specific individuals in *spear phishing* attempts, customizing e-mail with personal information to draw individuals to specific Trojan-horse websites.

Phishing is sometimes seen as a problem of education and experience, raising the question: "why can't people just stop clicking on the bad links?" While studies show training can help [17], training is expensive and time-intensive, and other studies show training provides more mixed benefits [5]. Moreover, the user-specific content in spear phishing exploits social pressures to encourage targets to set aside training and click. Even with training, ideally technical methods for anti-phishing would assist users, both naïve and trained.

There are two classes of technical methods to intercept phishing attempts. Most browsers today detect potential phishing with URL blacklists such as the Google Safe Browsing API, PhishTank [22], Is It Phishing [23] service, and the Netcraft toolbar [21]. 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. Moreover, while blacklists may protect against common phishing sites, they are unlikely to track "pop-up" sites used for spear-phishing against a small number of targeted victims.

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 and personalized detection of phishing websites. Our mechanism provides *proactive* in-browser testing of visited websites against likely phishing content, providing rapid defense with neither the delay of blacklist identification nor the strict constraints of whitelists. Each user can *personalize* the sites they visit and identify target content that might be used in phishing. Personalization customizes defenses and generates uncertainty in attackers, increasing protection against targeted, user-specific sites and spear phishing. Personalization can augment shared, centralized lists.

In this paper we introduce *AuntieTuna*, a web browser plugin that provides anti-phishing alerts as a user browses. Our approach includes a usable and simple mechanism for users to identify and customize protection against their own target sites. Our system indexes the target site's content and watches for this content to appear at *incorrect* sites as a sign of a active phishing. While prior work has visually compared good website layouts with potential phishing sites [27], we focus on the content itself using cryptographic hashing. Our insight is that cryptographic hashing of page contents allows *precise and efficient* bulk identification of content reuse at phishing sites.

The contribution of this paper is to show that our precise phishing detection using cryptographic hashing and user-personalized lists is both usable and effective. We emphasize usability through automated and simple manual addition of target sites and clean reports of potential phish that include context about the targeted site. Since each user develops a customized list of target sites, our approach presents a diverse defense against phishers. Our approach is precise, with zero false positives. We show that our algorithms detect a majority of phish, and is robust to several countermeasures, although it can be defeated by techniques such as a phishing site using only new images. AuntieTuna does not slow web browsing time and presents alerts on phishing pages before users can divulge information. A small number of alpha users have been using the browser extension, and we have released our extension and source code at https://ant.isi.edu/software/antiphish.

USEC '16, 21 February 2016, San Diego, CA, USA
Copyright 2016 Internet Society, ISBN 1-891562-42-8
http://dx.doi.org/10.14722/usec.2016.23012

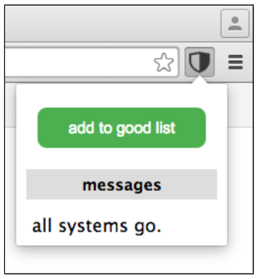
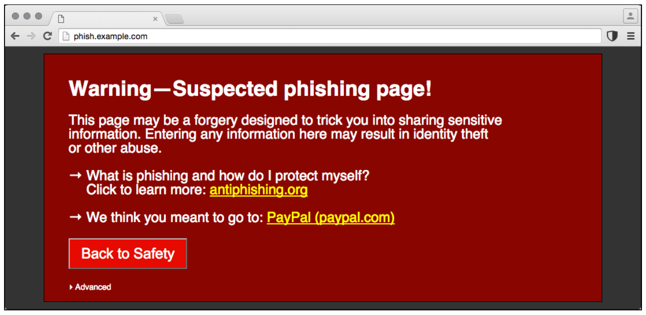


Fig. 1. Users click the “Personalize Button” on websites to add to whitelist of known-good sites.



Warning—Suspected phishing page!

This page may be a forgery designed to trick you into sharing sensitive information. Entering any information here may result in identity theft or other abuse.

→ What is phishing and how do I protect myself? Click to learn more: antiphishing.org

→ We think you meant to go to: PayPal (paypal.com)

Back to Safety

▸ Advanced

Fig. 2. Example of actively preventing a user from accessing a phishing site.

II. Related Work

Given the importance of phishing, many anti-phishing solutions have been proposed. We build on prior experience in phish detection, and anti-phish user interfaces and education.

1) Automating Phish Detection: There are many different approaches to detecting phish. Anti-phishing blacklists, page heuristics, or a combination of both are used in browser toolbars [22], [23], [21], but aren't always effective against phishing attacks [26], performing poorly even when blacklists were kept up-to-date [28]. Our plugin proactively detects phish using target content from known good sites, thus avoiding the delay in updating and retrieving blacklists.

Machine learning can also used to detect phish. By converting a website's content [14] or URL and domain properties [20] into a set of features or feature vectors, machine learning can look for websites that are similar, but have anomalous properties, such as “right” content in the “wrong” place. Computer vision techniques [2] can also be used to visually match the images on visited webpages with the originals. While these techniques can detect new phish, their approximate matching risks many false positives, and their high computational requirements make them difficult to run on clients. We instead employ precise content matching using hashing to avoid false positives, and to provide lightweight detection that can run in a client's browser without centralized support.

Other approaches measure the similarity of phish and original sites by looking at their content and structure. Similarities can be computed based on the website's visual features (text content, styles and layout) [19], or object positioning in their Document Object Model (DOM) trees [24], [27]. CANTINA [29] sends signatures based on the highest ranked words from the page's content through search engines and assumes valid content will be highly ranked in the results. Each of these approaches use approximate matching, while we instead apply cryptographic hashing to avoid false positives when detecting phish that reuse content from the original website.

Although not for phishing, CodeShield uses personalized whitelists to identify good PC-based applications [11]. Users must verify newly installed applications, and the process requires multiple steps to encourage careful review. We too apply personalized lists to antiphishing detection, but we emphasize fully automated or easy manual addition to the whitelist.

2) Anti-Phishing User Interfaces: User interfaces in anti-phishing tools play an important role in determining whether a user clicks on phish after it has been detected. There is the need for clear, non-subtle visual indicators of security problems [10]. Zhang et al. [28] found that while some user interfaces used colored indicators to signal if a website was legitimate or phish, they also found a lack of meaningful user interaction once a warning has been presented. Egelman et al. [8] found that most users responded positively when active active warnings interrupt the user, prevent clear, recommended actions, and are not easily closeable. Inspired by this work, our active warnings follow these guidelines, interrupting the page and providing information on our choice and education.

Other approaches seek to prevent information disclosure by focusing on the site's login page. Dhamija and Tygar [7] propose using a memorable “visual hash” as a prominent graphical indicator that the site being accessed is secure and trusted. Google's Password Alert [13] binds together the known-good website and the user's login details; if the password is reused or entered in somewhere else, the user is warned about phish and asked to change their password. We instead focus on phish website detection; but our approach could be used with these alternatives.

3) The Role of User Education: Multiple studies explore why users are susceptible to phish, and user education against phishing attacks. For example, they encourage users to recognize indicators such as incorrect URLs or broken locks (TLS) that are *around* the main content that indicate something is amiss.

Herley et al. [15] found the mental costs in frequent evaluation of such indicators exceeds its benefits; users often perceive the consequences of getting phished as low and ignore warnings. Additional studies [17], [18] showed that anti-phishing training for users was effective when provided immediately as the user clicks on phish in email and when done periodically. We follow these studies' recommendations by working silently without any requisite indicators and intervening with an active alert only when we detect a phishing website. We also point the user to resources where they can learn more about phishing and how to avoid falling victim to phishing attacks.

III. Design for User-customizable Anti-Phishing

Our anti-phishing system consists of three components: a **browser-plugin** watches websites a user tries to browse (§ III-D). Using our **detection algorithms** (§ III-B), it compares each new website against a *list of target content* by comparing cryptographic hashes: a detected phish will have a match in the content list and not be in a *whitelist of known-good sites*. Finally, we allow users to **personalize** the list of target content (§ III-A), customizing a common list of well known phishing targets. We describe these approaches below and in § III-C highlight our choices that optimize usability.

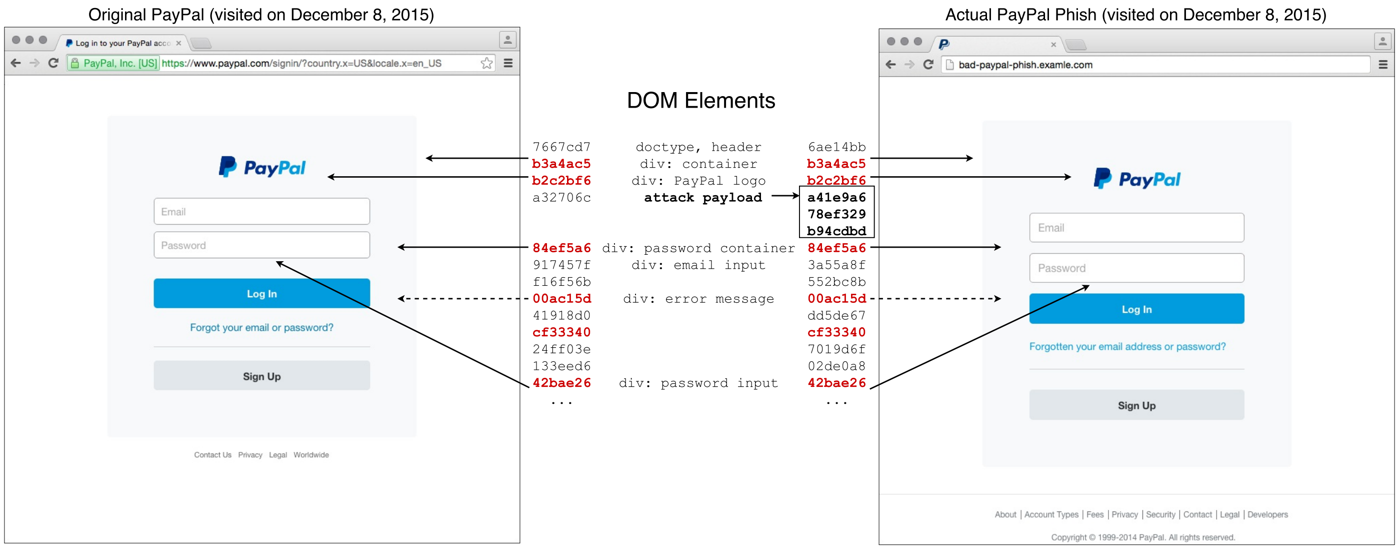

Original PayPal (visited on December 8, 2015)
Actual PayPal Phish (visited on December 8, 2015)
DOM Elements
7667cd7 doctype, header 6ae14bb
b3a4ac5 div: container b3a4ac5
b2c2bf6 div: PayPal logo b2c2bf6
a32706c attack payload a41e9a6
78ef329
b94cdbd
84ef5a6 div: password container 84ef5a6
917457f div: email input 3a55a8f
f16f56b 552bc8b
00ac15d div: error message 00ac15d
41918d0 dd5de67
cf33340 cf33340
24ff03e 7019d6f
133eed6 02de0a8
42bae26 div: password input 42bae26
...
PayPal
Email
Password
Log In
Forgot your email or password?
Forgotten your email address or password?
Sign Up
https://www.paypal.com/signin/?country.x=US&locale.x=en_US
bad-paypal-phish.examle.com

Fig. 3. Detecting a phishing attempt against PayPal. The known good site site (left) is visually similar to the phish (right), and common elements are identified by identical hashes of DOM elements (red values in the middle columns).

A. Identifying and Personalizing Target Content

A central goal of our approach is easy-to-use, per-user customization. Here we describe how and what information is collected, and in § III-C we discuss usability.

Detection is based on looking for target content in unexpected places. Users identify both target content and its expected locations by marking sites that may be targets for phishing using a simple web button (Fig. 1). This button adds that site to the whitelist of known-good sites, and adds hashes that identify the content of that page to the list of target content. This approach is analogous to public key pinning [9], and allows each user to build a custom list of sites they trust. (In § III-A we show how even this button can be automated.)

Once a site is marked as known-good, its content may evolve over time. We update our the content for known-good targets by *opportunisticly rehashing* these pages when a user revisits their URLs after some time.

We choose to build and store the whitelist and target content in each client, distributing detection and avoiding any centralized infrastructure. (Some blacklists or whitelists, like Google's Safe Browsing API or an HTTP proxy depend on centralized infrastructure and so require global network connectivity and infrastructure managed by a third-party.)

We expect to draw on both centralized and per-user whitelists and target content. Organizations may distribute target content lists, either generated centrally or aggregated from users.

But we expect user-customization to help build robust resistance to phishing in two ways. First, some sites offer user-specific "skins". For example, users an indicate a preferred background or color scheme in Google and Yahoo's websites. By selecting these *specific* versions of these popular sites users tune anti-phishing to their profiles. Second, individuals often access smaller sites that are specific to their behavior, yet are vulnerable sources to spear phishing. For example, a company may have an public-facing internal portal that requires authentication. By making each user's defenses more diverse and unique, we avoid a "monoculture" of anti-phishing filtering [12], decreasing the effectiveness of bulk attacks.

We augment manual identification of pages with a fully automatic approach: every time a user agrees to save the password for a web page, we automatically mark that site as a phishing target. This approach leverages the existing indication of user trust (save my password) to provide a form of Trust On First Use [25]. We are in the process of integrating this method into our system; when deployed, it will provide protection without *any* need for user interaction (the button can be eliminated).

B. Processing Pages: Hashing and Detection

Our process of identifying target content and matching it against new pages to detect phishing uses cryptographic hashing. We first describe how this process is used to add a known good page to target content and the whitelist, then how it is used to check unknown content for potential phishing. We have explored the use of hashing previously to detect plagiarism and content duplication for advertising on the web [3]; here we consider how it can be used specifically for anti-phishing.

1) Processing a Known-Good Page: When a web page is identified as known-good (as described previously, § III-A), we must record the content and the URL of that website. We place the URL on the whitelist. To track the content, to process a given web page we walk the page's DOM representation in the browser, breaking it into "chunks" delimited by `<p>` and `<div>` tags. (Other delimiters are possible, but we found these to be most effective). We remember the contents of each chunk 25 or more characters in length by computing its cryptographic hash with SHA-256 [1] (we filter out common, small-length chunks to avoid affecting our results). We then save the hashes on the client's local storage and add the URL to 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 [4].

The left part of Fig. 3 shows a known good page, with the PayPal login page taken on 2015-12-08 as an example. The truncated hashes of each DOM element are listed in the middle column (7667cd7, b3a4ac5, etc.).

Target content of some sites will evolve over time. *Opportunistic recrawl* keeps our record of those pages fresh: when a user re-accesses a page in the whitelist after some time, we use that opportunity to refresh our hashes of their content.

2) Processing Unknown Content: We process an unknown page for potential phishing in the same way: we walk the DOM, breaking it into chunks and computing the hash of each chunk. We then compare the number of chunks that match the list of target content. If the number of matches is greater than a threshold, we flag the webpage as suspected phish and actively prevent the user from accessing it (Fig. 2).

The *right* page in Fig. 3 shows an actual PayPal phishing page we copied on 2015-12-08 (the URL has been obscured for privacy; the phish is no longer accessible as of 2015-12-10). Visually, the sites are identical to mislead a user. The red hashes in the middle indicate that this similarity was accomplished by copying graphical and text elements. In fact, not all of the duplication is visible—we detect duplication in a hidden error message `div` (indicated with dotted arrows). This duplication supports our automated detection.

Detection of a phish results in an overlay that obscures the content (Fig. 2). The alert encourages users to back-off and gives links to web pages that can help them learn about phishing and avoid falling victim to phish. We also provide users with a link to the "most similar trusted page" as method of explaining why we distrust this page. We also allow an escape mechanism to handle false positives, but presented in a manner to discourage and caution its use (initially hidden under the "Advanced" option).

C. Design Choices for Usability

Our approach to maximizing the usability of our anti-phishing methods is based on minimizing user interaction and no-knobs ("hands free") use. We adopt these goals based on studies which show that users reject security advice when it poses too great of a burden relative to its perceived benefits [15], and the need for clear, non-subtle visual indicators of security problems [10]. These goals reflect in four design choices: full automation of building user-customized lists of target content, minimal controls for optional user additions, suppression of untrusted content, and some explanation and reasoning for that suppression.

First, we can fully automate user customization by integrating identification of target content with password storage, an existing method of managing trust. Although we optionally allow users to manually flag pages as target content, and organizations to distribute centralized lists of trusted sites, this automation customizes phishing defense for each user with no explicit effort. In addition, our manual interface is very simple: use one button to add a site (Fig. 1).

We actively suppress the visited webpage if it is suspected phish. Prior studies have shown that users often ignore passive warnings [8] and continue through to dangerous content. We explicitly choose *not* to redirect the user to their intended website automatically, so as to discourage users from becoming complacent using phishing links.

Finally, rather than treat AuntieTuna as a black box, we provide the user some background about why we found the candidate site as a phish by including a link to the "closest trusted site". We provide this link as content, not as an assistance to redirect, and accompany it with links to information about phishing.

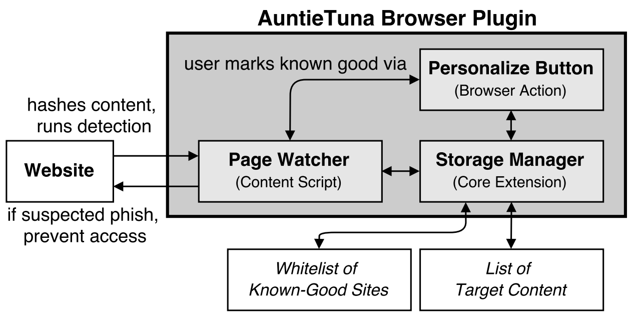


Fig. 4. Implementation diagram of the AuntieTuna anti-phishing plugin.

D. Implementation of Anti-Phishing in AuntieTuna

Our plugin is implemented as an extension to the Google Chrome browser, written in JavaScript and using only the Chrome APIs. This section summarizes our implementation choices to operate with the security model for Chrome plug-ins [6]. We expect that our approach can port to other browsers.

AuntieTuna consists of three components (Fig. 4): the Personalize Button, Page Watcher, and Storage Manager. In the Chrome model, these components run as a Browser Action, Content Script, and Core Extension, respectively.

1) Page Processing Workflow: Processing pages begins with users personalizing their list of target content by using the Personalize Button (located on the browser toolbar, Fig. 1) on known-good websites. Pressing the button signals to the Page Watcher to mark as known good and chunk the current webpage. The Storage Manager stores the resulting hashes and site URL in the list of target content and whitelist of known-good sites, respectively. The appended list of target content is ready for use immediately.

The Page Watcher runs continuously in the background, watching for and processing unknown pages not found in the whitelist of known-good sites. When the page has rendered, the Page Watcher processes the page as described in § III-B and, if the page is suspected fish, injects an overlay (Fig. 2) on the current page to prevent the user from accessing it.

2) Platform-Specific Customizations: Implementing our browser plugin required changes to our discovery and detection methodology from our prior work [3]. Chrome's security model prevents our extension from accessing the raw underlying HTML of sites, but it does allow access to the parsed version of the page in the form of the page's document object model (DOM). Because the DOM is the processed (rendered) version of the underlying HTML, it can be modified by scripts on the page or other concurrent extensions, potentially reducing the accuracy of our phish detection mechanism. (We discuss this problem and possible countermeasures in § IV-B.) Additionally, the rendered DOM is browser-specific. Thus, the hashes in a given user's phishing target content may not apply to users of other browsers.

We generate and store all hashes and lists in the client browser, making our methodology completely self-contained, without dependence on outside infrastructure or processing, Our approach runs after page render time, imposing *no* increase in page render time. However, processing time provides a gap where users are briefly un-alerted about a potential phish attempt. In § IV-C we show that we are faster than user reaction time on PC-class hardware, but this gap will be larger on lower-end hardware such as tablets or mobile phones.

We can reduce classification time of unknown pages time by using Bloom filters to speed comparison of the contents of a new page against our list of target content. We can eliminate false positives that occur with Bloom filters with a two-tiered search: if a hash of some content chunk is "found" in the Bloom filter, do another search in the full list of target content. Since we expect most searches to return negative, the amortized cost of doing a full search is sufficiently negligible to maintain a zero false-positive detection rate. We have not yet implemented this optimization.

IV. Effectiveness of Phishing Detection

To evaluate AuntieTuna, we consider its effectiveness today and in the face of potential countermeasures. We also examine its effects on browser performance and in our usage to date.

A. Evaluation of Phish Detection Accuracy

We now evaluate the effectiveness of the core algorithms of AuntieTuna. This is our first evaluation of DOM-based hashing, although it builds on our prior work evaluating duplication of HTML [3]. Since we do not have access to a large source of spam, we approximate this system as follows. We target PayPal phishing, and fill our target content list with current and recent PayPal U.S., U.K., and French home pages (Sept. 2014, plus Jan. 2012 to Aug. 2013) loaded from archive.org. We gather six variations on these three web pages, resulting in a target content list containing 311 distinct chunks longer than 25 characters.

We test AuntieTuna against a *suspected phish stream* of 2374 URLs drawn from from PhishTank [22] over 2 days (2014-09-24 and 2014-09-25). PhishTank is a crowd-sourced anti-phishing service. Since the lifetime of a phish is short, we automatically rip the target of each suspected phishing link. We compare each suspected phish against our target content list with our algorithm (§ III-B) with a detection threshold of one or more non-trivial chunks.

To evaluate ground truth, we manually examine the suspected phish stream and identify 124 (of the 1888) as PayPal phish attempts. We further identify 85 of the remaining sites as phish utilizing content from PayPal. Our mechanism detects 50 (58.8%) pages that pass the detection threshold: 42 are direct rips detected with no normalization applied, and an additional 7 are detected with whitespace normalization. Table I classifies the type of techniques each phishing site uses.

Without taking steps to defeat countermeasures, our approach has a fairly high false negative rate with a sensitivity of 58.8%. However, our targeted dataset has zero false positives and a specificity of 100%. This experiment suggests our approach is a valuable additional technical method to automatically block phishing attempts, at least against our sample. Evaluation against more diverse phished sites and use by more users is important future work. We next discuss hardening our approach to countermeasures.

B. Resisting Potential Countermeasures

While most phish copy much of the original site, other phish use different techniques to attack their targets, sometimes deliberately obscuring the source of their content. We discuss how these countermeasures affect the accuracy of our phish detection, and strategies to work around them.

All phish are constrained by the requirement that they must *look* very similar to the original. Most simply copy content from the original, prompting our approach. However, others obscure that content. A fair number of phish (39 of the 124 PayPal-appearing phish) replace the original content with images. Our approach cannot see through this concealment and we exclude these from our list of PayPal phish that are potentially detectable by our method. Fortunately, such sites can be obvious (for example, text is not selectable, or fonts vary by platform), and are subject to image analysis.

TABLE I. Classification of phish in two days of PhishTank reports, based on detection against PayPal. Sensitivity = 58.8%, Specificity = 100%.

Description	Num. Pages	%	
Candidates	2374		
Unavailable	486		
Ripped	1888		
Other	1764		TN = 1764
PayPal (image-based, *removed*)	39		
PayPal	85	100.0	FP = 0
Successfully detected	50	58.8	TP = 50
Direct rips	35		
Whitespace norm.	8		
JavaScript obfuscation	7		
Custom-styled with minor PayPal content	35	41.2	FN = 35

We next focus on the 85 potentially detectable (non-image-based) PayPal phish: of these we detect 50 phish (58.8%).

Sites can vary the original site's HTML slightly, replacing whitespace or making other changes that do not affect the visual result. The DOM passes some variations through, so we normalize whitespace as part of our processing, detecting 8 (9.4%) sites that we would otherwise miss. A phisher willing to mutate every element will evade our approach, however, we argue that such a phish would also appear suspicious (due to misspellings or awkward phrasings) and is more work to generate than cut-and-paste.

More challenging are sites that generate or obfuscate content dynamically with JavaScript to elude web crawlers that look for and process static HTML only. Because we parse the DOM after any JavaScript has run, we can see through this obfuscation. Manual identification showed 7 suspected PayPal phish (8.2% of detectable phish) that used JavaScript that we find in DOM-based analysis but not in HTML alone: we found all 7 of them.

A phisher could use use homographs (look-alike characters) in ASCII or Unicode (e.g., Greek ρ for "P") to spoof the original. Our approach cannot currently see through these techniques, although we could potentially normalize characters by shape just as we normalize whitespace.

Finally, we see a fair number (35 pages, 41%) of potentially detectable phish construct an original phishing site using only a small mount of content taken from the original site. We miss these phish, although we expect their deviation from target content makes them less believable.

We conclude that we miss a number of phish that use images or copy minimally from the target, however we detect more than half of phish with no false positives, thus providing a useful service. Wide use of our approach will of course cause phishers to move to other types of attacks or target the thresholds our tool uses. Personalization makes such coutermeasures difficult, but we would consider "raising the bar" by rip-and-copy attacks a partial victory.

C. Browser Performance with AuntieTuna

A concern with any plugin is that it slows the browsing experience, so we next examine the computation performed by our plugin. AuntieTuna introduces *zero increase* in page render time because we process the page only after it has finished rendering. Thus the performance of AuntieTuna is *not* about a

TABLE II. PAGE RENDER AND AUNTIETUNA EXECUTION TIMES

Website	Page Render ms (σ ms)	AuntieTuna ms (σ ms)
google.com	327 (15)	144 (6)
paypal.com	349 (5)	20 (2)
nytimes.com	5316 (1632)	167 (6)
en.wikipedia.org	321 (7)	75 (3)

"slower web", but instead about a potential gap between when the page is visible and when we detect it as phish.

We run our benchmarks on four sites using Google Chrome (v47.0.2526.80, 64-bit, 2015-12-08) with the list of target content from § IV-A. We test on a PC running OS X 10.10.5 with an Intel i7-2760QM processor and 8 GB of memory.

In Table II we report mean and SD page render and AuntieTuna execution times taken from five runs. We find that our plugin's execution time ranges between 20–167 ms on each page. There is quite a bit of variation depending on the complexity of the page. Only for the highly-optimized Google home page does scan time approach page render time; in other cases it is small in relative to page render time. However, we again emphasize that analysis happens *after* rendering and *in parallel* with viewing, so user browsing is not affected.

A more serious concern is if we can put up a warning fast enough, before a user divulges private information, since they will see content while we process the page. Our longest scanning time is 167 ms; while this one-sixth of a second will be noticeable, we believe there are few users who could enter their information and click submit in this short amount of time.

A great deal of web use today occurs on less powerful hardware with tablet computers or mobile phones. We have not evaluated our plugin on these devices. While our plugin will be slower on slower computers, user data entry will also be slower. Porting our plugin to mobile devices is future work.

We conclude that AuntieTuna has no effect on web browsing performance on reasonably powerful hardware, and it runs fast enough to protect users.

D. Experiences in Real-World Usage

We have been using the browser extension continuously since March 31, 2015. So far the extension works reasonably well, detecting the known phish we use for testing without noticeably affecting the speed of normal browsing operations. We have not yet seen any real phish, nor any false positives. A larger and formal user study remains as future work.

V. CONCLUSIONS

This paper has described a new approach to phish detection and its realization in *AuntieTuna*, a Chrome browser plugin. We described our design decisions to make our approach easy to use, with automatic or simple manual addition of targets and clean reports of potential phish. We have shown our approach is precise (no false positives), that it detects a majority of phish in controlled experiments, and that it does not affect browsing speed and it presents alerts before users can divulge information. We have released our extension and source code on our website at https://ant.isi.edu/software/antiphish.

Acknowledgments: We thank Danyong Zhao for initial exploration in developing a browser plugin. Research by Calvin Ardi and John Heidemann in this paper is partially sponsored by the Department of Homeland Security (DHS) Science and Technology Directorate, HSARPA, Cyber Security Division, via SPAWAR Systems Center Pacific under Contract N66001-13-C-3001, and via BAA 11-01-RIKA and Air Force Research Laboratory, Information Directorate under agreement numbers FA8750-12-2-0344 and FA8750-15-2-0224. The U.S. Government is authorized to make reprints for Governmental purposes notwithstanding any copyright.The views contained herein are those of the authors and do not necessarily represent those of DHS or the U.S. Government.

REFERENCES

[1] D. E. 3rd and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)," RFC 6234 (Informational), Internet Engineering Task Force, May 2011.

[2] S. Afroz and R. Greenstadt, "Phishzoo: Detecting phishing websites by looking at them," in *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*. IEEE, 2011.

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

[4] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, Jul. 1970.

[5] D. D. Caputo *et al.*, "Going spear phishing: Exploring embedded training and awareness," *IEEE Signal Processing Magazine*, vol. 12, no. 1, Jan. 2014.

[6] N. Carlini *et al.*, "An evaluation of the Google Chrome extension security architecture," in *USENIX Security Symposium*, 2012.

[7] R. Dhamija and J. D. Tygar, "The battle against phishing: Dynamic security skins," in *Proceedings of the 2005 Symposium on Usable Privacy and Security*, ser. SOUPS '05. ACM, 2005.

[8] S. Egelman *et al.*, "You've been warned: An empirical study of the effectiveness of web browser phishing warnings," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '08. ACM, 2008.

[9] C. Evans *et al.*, "Operation of anycast services," Internet Request For Comments, RFC 7469, Dec. 2015.

[10] S. Furnell, "Phishing: can we spot the signs?" *Computer Fraud & Security*, vol. 2007, no. 3, 2007.

[11] C. Gates *et al.*, "Codeshield: Towards personalized application whitelisting," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12.

[12] D. Geer *et al.*, "Cyberinsecurity: The cost of monopoly: How the dominance of Microsoft's products poses a risk to security," Computer and Communications Industry Association, Tech. Rep., Sept. 24 2003.

[13] Google, "Password Alert," Apr. 2015.

[14] R. Gowtham and I. Krishnamurthi, "Phishtackle–a web services architecture for anti-phishing," *Cluster Computing*, vol. 17, no. 3, Sep. 2014.

[15] C. Herley, "So long, and no thanks for the externalities: The rational rejection of security advice by users," in *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*, ser. NSPW '09.

[16] J. Hong, "The state of phishing attacks," *Communications of the ACM*, vol. 55, no. 1, Jan. 2012.

[17] P. Kumaraguru *et al.*, "School of phish: A real-world evaluation of anti-phishing training," in *Proceedings of the 5th Symposum on Usable Privcay and Security*. Mountain View, CA, USA: ACM, Jul. 2009.

[18] ——, "Teaching johnny not to fall for phish," *ACM Trans. Internet Technol.*, vol. 10, no. 2, Jun. 2010.

[19] W. Liu *et al.*, "An antiphishing strategy based on visual similarity assessment," *Internet Computing, IEEE*, vol. 10, no. 2, 2006.

[20] J. Ma *et al.*, "Beyond blacklists: Learning to detect malicious web sites from suspicious urls," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. ACM, 2009.

[21] Netcraft Ltd., "Netcraft extension: Phishing protection and site reports," Available at: http://toolbar.netcraft.com/, 2015.

[22] OpenDNS, "PhishTank," Available at: http://www.phishtank.com, 2015.

[23] V. Retro, "isitPhishing—anti phishing tools and informations," Available at: http://www.isitphishing.org/, 2015.

[24] A. P. Rosiello *et al.*, "A layout-similarity-based approach for detecting phishing pages," in *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, Sept 2007.

[25] D. Wendlandt *et al.*, "Perspectives: Improving SSH-style host authentication with multi-path probing," in *Proceedings of the USENIX Conference Proceedings*. USENIX, Jun. 2008.

[26] M. Wu *et al.*, "Do security toolbars actually prevent phishing attacks?" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '06. ACM, 2006.

[27] W. Zhang *et al.*, "Web phishing detection based on page spatial layout similarity," *Informatica*, vol. 37, no. 3, 2013.

[28] Y. Zhang *et al.*, "Phinding Phish: Evaluating Anti-Phishing Tools," in *Proceedings of the 14th Annual Network and Distributed System Security Symposium*, ser. NDSS '07.

[29] ——, "Cantina: A content-based approach to detecting phishing web sites," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07.