PhishSafe: Leveraging Modern JavaScript API's for Transparent and Robust Protection

Bastian Braun
ISL, University of Passau
Passau, Germany
bb@sec.uni-passau.de

Martin Johns
SAP Research
Karlsruhe, Germany
martin.johns@sap.com

Johannes Koestler
University of Passau
Passau, Germany
koestler@fim.uni-passau.de

Joachim Posegga
ISL, University of Passau
Passau, Germany
jp@sec.uni-passau.de

ABSTRACT

The term "phishing" describes a class of social engineering attacks on authentication systems, that aim to steal the victim's authentication credential, e.g., the username and password. The severity of phishing is recognized since the mid-1990's and a considerable amount of attention has been devoted to the topic. However, currently deployed or proposed countermeasures are either incomplete, cumbersome for the user, or incompatible with standard browser technology. In this paper, we show how modern JavaScript API's can be utilized to build PhishSafe, a robust authentication scheme, that is immune against phishing attacks, easily deployable using the current browser generation, and requires little change in the end-user's interaction with the application. We evaluate the implementation and find that it is applicable to web applications with low efforts and causes no tangible overhead.

Categories and Subject Descriptors

K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection

Keywords

Web Security; Phishing; Protection

1. INTRODUCTION

From a security point of view, passwords are a terrible choice for authentication. They are easily stolen. Often, they are easy to guess, due to the fact that they were chosen in a fashion that allows the user to remember them (e.g., names of pets, children, or cars). And they are frequently reused, causing the compromise of one server to probably affect several independent applications as well.

However, it is an unrealistic assumption, that we will reach a situation, in which password-based authentication looses its significance, even in the presence of well designed password-less techniques, such as client-side SSL authentication, and promising new developments, such as Mozilla Persona [30].

Unlike all alternatives, the user's requirements to utilize password authentication are extremely light-weight: All she needs to logon, is to remember her username and password. Password-less authentication systems either require preconfigured state on the device, such as installed client-side certificates, the presence of specific hardware, such as smart card readers, or the possession of additional items, e.g., a cell phone to obtain out-of-band credentials [15].

This characteristic of password authentication is even amplified in the presence of web applications: The only remaining software requirement is, that on the utilized computer a web browser is installed, something that can be taken for granted since several years. Hence, no matter in which situation a user is, as long as she remembers her password and has a networked device with a web browser at her disposal, she is able to access her applications. No other system for networked applications offers similar properties. It can even be argued that the ease of password authentication was one of the success-factors of the web.

However, the passwords' strength – their ease of use – is also their biggest weakness: As easily they are entered, as easily they are stolen, in case that a used password field is actually under the control of the attacker.

In variants, this class of attack, known under the term *phishing*, is probably as old as the discipline of password authentication itself, having its roots in social engineering attacks [29]. The severity of phishing is recognized since the mid-1990's and a considerable amount of attention has been devoted to the topic. However, as we will show in Section 2, currently deployed or proposed countermeasures are either incomplete, cumbersome for the user, or incompatible with standard browser technology.

In this paper, we present *PhishSafe*, a light-weight approach that provides robust security guarantees, even in case that the user's password was successfully stolen. The core of our approach is a transparent browser-personalization pro-

CODASPY'14, March 3–5, 2014, San Antonio, Texas, USA.
Copyright 2014 ACM 978-1-4503-2278-2/14/03 ...$15.00.
http://dx.doi.org/10.1145/2557547.2557553.

cess, that is invisible to the user. This way, unlike the majority of existing anti-phishing approaches, *PhishSafe* does not burden the user with altered authentication interaction or additional burdens, such as recognizing security indicators or visual authenticity clues. On the contrary: As long as a user predominately uses only a single browser, she won't notice a difference to the currently established, insecure scheme.

Paper Organization

The remainder of this paper is structured as follows: The next section provides an in-depth discussion of phishing attack vectors, current solutions and their shortcomings, and the user as the weakest link in phishing protection as well as a definition of the attacker models considered in this paper. Section 3 presents the concept of PhishSafe, our authentication scheme to overcome phishing attacks. Section 4 describes the implementation of PhishSafe and provides technical details. The evaluation is given in Section 5. Section 6 discusses related work before Section 7 concludes.

2. PHISHING ATTACKS

While phishing attacks have a long history, phishing activity has not decreased over time (see Fig. 2). The attackers' strategy, however, has changed to counter the anti-phishing means in use, for instance, phishing sites move faster to prevent blacklisting (see Fig. 1). In this section, we describe modern phishing attack methods, model the attackers' capabilities, evaluate proposed anti-phishing solutions, and analyze why those solutions have not significantly reduced phishing activities.

2.1 Attack Method

The Anti-Phishing Working Group (APWG)[1] states that phishing schemes use "spoofed e-mails purporting to be from legitimate businesses and agencies, designed to lead consumers to counterfeit websites that trick recipients into divulging financial data such as usernames and passwords." [45] Attackers usually send emails or personal instant messages and put pressure on the recipients to perform actions intended by the attacker. For instance, recipients are told that their email quota is reached, their credit card is disabled, or an invoice has not been paid. Usually, to increase the pressure and omit a reconsideration, immediate steps are allegedly necessary. These steps require logging into an account on a website. In this scenario, attackers know the target business. All they need to do is copy the public design of the website and send bulk emails. In order to educate customers, anti-phishing campaigns published rules of conduct. For example, users are advised to not click on links embedded in emails if the link does not include the expected domain of the (seeming) sender. As another rule of thumb, reliable emails contain the recipient's name and maybe other personal information which is supposedly not known to a phisher.

The following attack vectors emerged in the past and illustrate the ongoing arms race between phishers and the anti-phishing community.

2.1.1 Concealing the Target Domain

In order to answer the anti-phishing suggestions, phishers took measures to make embedded links look familiar to the user. These measures range from open redirects on the target website, over URLs featuring a target domain prefix and URL shorteners hiding the target, up to malicious relying parties in single sign-on protocols.

Open Redirects First of all, a phisher's chance is considerably higher if the link the victim is supposed to click appears to belong to the expected domain. In that sense, an attacker can succeed if he finds an open redirect function on the target web application. Web applications redirect their users for several reasons: when a requested web page is not found (HTTP 404), users are redirected to a landing page that explains what happened. Webmail providers redirect their customers via 'de-referrers' to avoid that the actual URL of the read email appears as a part of the subsequent request to the foreign domain (in the `Referrer` header). Open redirects do not sanitize their input, i.e., the redirect target and source. Given that the attacker prepared a phishing site for `example.com` that has an open redirect, he can send out emails asking users to click on `https://www.example.com/redirect?target=example-attack.com`. A better masking is possible by URL encoding the target parameter. Finally, the victim sees an `https` link to the expected domain and can eventually check the SSL lock on `https://www.example-attack.com` but is attacked, though.

Confusing URLs Second, it is often sufficient to make the URL appear innocent at a first glance.

Non-expert users can hardly distinguish between the host, domain, and path elements of a URL. Phishers exploit this weakness crafting links like `https://www.example.com.attacker-domain.com` which seem to contain the expect domain name `example.com`. Similar approaches include typos in the URL, e.g. `https://www.gooogle.com`.

A more sophisticated attack is known as international domain name (IDN) homograph attack [14]. This attack makes use of so-called homographs, characters from non-latin alphabets that are indistinguishable for humans but interpreted by browsers as different symbols.

URL Shorteners The emerging trend towards URL shorteners, that save characters on Twitter and prevent line breaks in emails, makes people familiar with short URLs and redirects to unpredictable URLs. Attackers exploit that people are more used to click on links from `bit.ly`, `tinyurl.com`, `is.gd`, or `goo.gl` than on links containing unknown domains. If the target website looks convincing enough, the user's focus is caught on the content [52].

Malicious Relying Parties Single sign-on (SSO) protocols require the user to log in once with her identity provider to obtain access to all related accounts. If the user first visits a relying party, she is redirected to her identity provider. A malicious relying party can redirect the user to a phishing identity provider to request the user's credentials.

2.1.2 Spear Phishing

Spear Phishing denotes a particular phishing attack vector that targets a set of victims the attacker has information about. While this attack is restricted to those users the attacker could gain knowledge about, it can still hit thousands of users. The first step in a common scenario is a data leak of a company's customer database. In most cases, there is a laxer security policy in place if the database does not contain critical data like passwords, social security numbers, or credit card information. Using the obtained data, how-

[1]`http://www.apwg.org/`

ever, an attacker can address his victims personally including the name, correct email address, and account number which used to be an indicator of a benign message.

2.1.3 *Browser-less Phishing*

A phisher can circumvent browser-based countermeasures if the user does not use her browser to follow his instructions. As a matter of fact, users regularly experience that colleagues or friends quickly ask for information by email or instant messenger. Phishers convey the notion of this scenario to make their victims reply with the credentials.

2.2 Attacker Models

In order to estimate a phishing attacker's capabilities, we define two attackers. These attackers define the scope of our work, i.e., we present existing approaches against these kinds of attackers in Section 2.3 and propose PhishSafe, our countermeasure, in Section 3.

We consider a *phishing attacker* as a remote web participant. He is able to set up websites and email accounts, can send emails and messages via instant messengers (IMs). He can obtain valid SSL certificates for his domains. We do not assume timing constraints, i.e., he can react immediately on any input at all time.

Moreover, we consider an *XSS attacker*. He has all capabilities of the phishing attacker but can also inject JavaScript code into vulnerable web pages.

Neither of both has control over the user's platform nor over the network. We neglect browser vulnerabilities and respective exploits. Also, they can not break cryptography.

2.3 Current Solutions

Several approaches have been applied so far to mitigate phishing attacks. In this section, we name them and explain their strengths and weaknesses. We find that they are either incomplete, cumbersome for the user, or incompatible with standard browser technology.

2.3.1 *Incomplete Countermeasures*

One class of countermeasures suffers from incompleteness in terms of false positives and false negatives, i.e., they do not protect against phishing on some sites and prevent access to genuine sites suspected to phishing.

Browser vendors, e.g. Microsoft[2] and Google[3], as well as third parties, e.g. PhishTank[4], provide lists of malicious and genuine websites. Browsers query their list upon accessing a website and check whether this site is known for phishing. The blacklists suffer from a window of vulnerability between the setup of a phishing site and its listing [43]. This window can be decreased by real-time queries towards the list providers for each unknown domain. The additional online query slows down page loading and reveals almost the complete browsing history to the list providers. List providers went over to classify websites automatically to capture phishing sites earlier [51], however, at the expense of accuracy, i.e., more false positives and false negatives [25]. The extraction of features from phishing sites provoked an arms race between phishers, who have a financial interest in passing those filters, and the blacklist providers. Among other features, phishers reduce the uptimes of their sites (see Figure 1) to make the blacklists come to nothing. The trend lasts and led to an average uptime of one day in 2012 leaving only very short reaction time to blacklist providers [44].

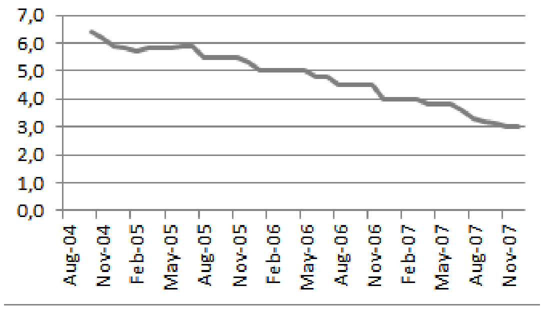


Figure 1: The Average Online Time of Phishing Sites in Days Between Oct '04 and Dec '07, the Time of Acquisition by the APWG, *src: Regular APWG Phishing Attack Trends Reports [46]*

2.3.2 *Countermeasures Cumbersome for the User*

Another class of approaches makes use of the increasing propagation of mobile devices. Users need to enter a second credential that is either received or generated by their mobile device in order to login or perform critical actions. This breaks their ongoing workflow as they need to switch to a different device. Example implementations include Google Authenticator [15] and one-time passwords sent to cell phones. Beside the fact that malware now also targets mobile devices to intercept received tokens [8], both approaches can not help against our attacker models (see above) because the attacker only needs to wait for the victim to enter her credentials and relay all gathered user data to the actual web application in real time. This way, the user serves as an oracle that provides the needed information. In this scenario, the attacker plays the role of a man in the middle without manipulation on the network layer.

Client-side SSL aims at replacing username/password-based logins. Though SSL could overcome most of currently known weaknesses in knowledge-based authentication, it has not become popular probably due to its setup complexity for non-expert users. Finally, SSL certificates are hardly portable. A user can login to web accounts from every device using an off-the-shelf browser and her password. It is rather difficult to store, carry, and use an SSL certificate securely on an untrusted computer.

2.3.3 *Countermeasures Incompatible with Standard Browser Technology*

A family of approaches extends the user's browser [21, 39, 56, 6, 55, 53, 37, 17, 47, 3, 38, 27, 2, 57, 41] (see Section 6 for details). Browser extensions and toolbars share a number of drawbacks:

- They provide no protection by default but only protect risk-aware users after installation.
- They are inherently incompatible with standard browser technology and can only protect users of supported browsers while porting them to other browsers is hard. [35]

[2]`http://windows.microsoft.com/en-US/windows-vista/Phishing-Filter-frequently-asked-questions`
[3]`https://support.google.com/chrome/answer/99020?hl=en`
[4]`http://www.phishtank.com/`

- The majority of browser-based solutions aims at detecting phishing websites while accessed. However, most users ignore issued warnings and more rely on the web content to estimate a website's authenticity. [52]
- Browser toolbars, that classify websites into phishing and harmless, are susceptible to false positives and false negatives, i.e. letting phishing sites pass while warning of genuine sites. Case studies showed that a high detection rate often comes with a high false positive rate. [59]
- Phishing is a particular problem on mobile devices while existing approaches are hard to port because of the limited screen size. [12]
- Phishers can evade most browser-based protection approaches by asking victims to reply by email to their inquiry.

2.3.4 Summary

We can conclude that the existing approaches still leave room for phishing attacks. None of the current solutions offers thorough protection for all users. The volatile number of active phishing sites reflect the ongoing arms race between phishers and anti-phishing blacklist providers (see Figure 2). The more stable number of phishing campaigns shows the unabated activity of phishers over a long period. Matters are complicated by more targeted spear phishing attacks which are harder to detect by generic features than common large-scale attacks.

Emerging consumer-oriented SSO protocols like Mozilla Persona [30], OpenID [36], and OAuth [33] decrease the user's attack surface. Nevertheless, they still require user logins with the identity provider and, thus, cannot remedy phishing attacks. SAML [24] and Shibboleth [20] target business environments and require a higher level of coordination between participants, thus, are more suitable for closed application scenarios. We provide more details in Section 6.3.

2.4 The Weakest Link: The User

After analyzing existing countermeasures and modern attack vectors, we identify the user as the weakest link. We find that phishing attacks abuse the user's misconception concerning her communication partner in the World Wide Web. Transferred to the physical world, a phishing attacker would set up a storefront that looks familiar to many people. In the virtual world of the World Wide Web, the attacker can succeed much easier for several reasons: First of all, the user has no personal reference point in terms of location. Informally speaking, she does not know where she actually is. Most users are not familiar with domains and URLs, and even if they were, they could still be misled by exploiting weaknesses in the Domain Name System (DNS spoofing, pharming). The international domain name (IDN) homograph attack [14] even deceived skilled security experts.

Second, users learned to assess a person's trustworthiness. While this assessment can be manipulated, there is hardly any natural feeling of trustworthiness with respect to programs and machines nor do reliable indicators help. Existing approaches focus on proving an email's (e.g. DKIM [4], SenderID [28]) or a website's (e.g. https) trustworthiness but not the opposite, i.e. in an attack scenario, they do not provide any helpful hint. Teaching users to check SSL indicators inspired phishing attackers to spoof those indicators or obtain valid certificates for similar domains, e.g. `gooogle.com`. Such indicators are missing on most mobile devices due to the limited screen size [32]. The opposite approach – warning users instead of indicating trustworthiness – made users being annoyed and ignore such warnings [9], because users want to make things happen and not think about security, so they do whatever is asked for in even unusual emails [7]. Attackers increase their chances by threatening their victims, for example, announcing bad consequences like blocking an email account or disabling the credit card. This strategy prevents that users contemplate on the message's reliability.

Third, automation allows large-scale attacks making the efforts worthwhile. The intention to classify phishing attempts led to an arms race meaning that attacks evolve and require new features to detect phishing [13].

To sum up, we conclude that the user must not play a decisive role in phishing protection nor can the user behavior be supposed to change. An algorithmic approach is needed to rule out phishing attacks.

3. PHISHSAFE

In this section, we describe the idea of our authentication scheme, named PhishSafe, that avoids the drawbacks identified in Sec. 2. Section 4 gives details of the implementation.

3.1 Design Goals

Following the lessons learned from previous approaches and current phishing techniques (see Sec. 2), we phrase the following design goals for PhishSafe: It

- sidesteps the arms race between phishers and the anti-phishing community,
- reduces reliance on the user,
- avoids dependence on the browser's interface,
- waives the need for additional devices and the installation of protective tools, and
- withstands the attackers defined in Section 2.2.

Our design goals are in parts inspired by Parno et al. [34] (see Section 6). In the remainder of this section, we motivate our design goals in more detail.

Sidestep the arms race between phishers and the anti-phishing community It is important to quit the arms race with financially motivated phishers that are always one step ahead. The anti-phishing community can only react on new phishing techniques while phishers update their features again.

Reduce reliance on the user We showed in Section 2.4 that the user is the weakest link in phishing scenarios. Hence, a reliable countermeasure must not rely on the user. Instead, it must tolerate that the user can be tricked and gives away all credentials she knows.

Avoid dependence on the browser's interface Approaches relying on the browser's interface either require the installation of additional software (e.g., toolbars or extension, thus, excluding users of not supported browsers or platforms) or can be spoofed using JavaScript or a favicon (e.g., simulating an SSL lock symbol). The interface is even hidden on mobile devices due to the limited screen size.

Waive the need for additional devices and the installation of protective tools The need for second devices makes processes more complex and requires considerable changes of the used logon procedure. Those devices must be always at hand, secure, and have a direct connection to the browser to transfer control. Obtaining passcodes

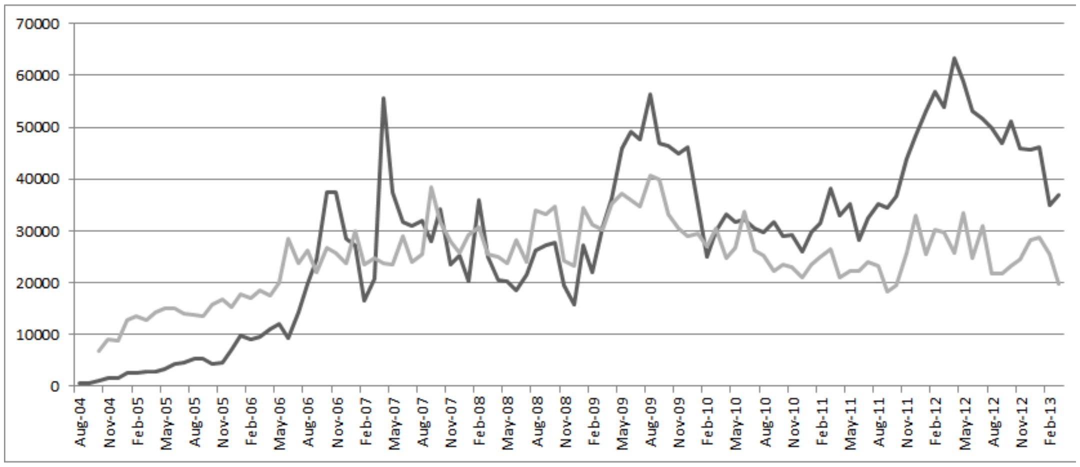


Figure 2: Phishing Statistics Since Aug. 2004 in Terms of Active Phishing Sites (dark grey) and Email Phishing Campaigns (pale grey), ***src: Anti-Phishing Working Group (APWG) Reports [46]***

from a second device is not an option because these can be phished and exploited.

The usage of protective software is always limited to risk-aware users utilizing a supported platform.

Withstand the attackers defined in Section 2.2 We modeled the attackers according to realistic assumptions. So, a reliable approach must provide protection against their attacks.

3.2 High-Level Overview

The main idea of PhishSafe is to release the user from responsibility: she neither needs to perform special actions nor check security indicators nor keep a secret other than her password. Instead, she will use a second factor she does not know and, thus, can not disclose to a phisher. This factor is stored in her browser and attached to logins towards the genuine web application. The web application prohibits logins without proper second factor authentication. An attacker luring his victim on a phishing site can obtain her password but not the second factor credential. However, the password alone is not enough to login. The second factor is established during account setup and, if necessary, restored after visiting a URL sent by email.

3.3 Detailed Authentication Process

As emphasized above, the authentication scheme implements two-factor authentication without the user knowing about it. In order to apply our authentication scheme, the website stores a secret token in the persistent web storage of the user's browser (see Sec. 3.4 for details). The user does not have to be aware of this token nor does she have to care. The important point is that this token is subject to the same-origin policy (SOP) [58] and not accessible to web applications on foreign domains.

When the user accesses the login page, a challenge string is invisibly included in the HTML form beside the username and password input fields. The page also embeds JavaScript code that computes the second factor credential from the challenge and the secret browser token using an HMAC function [23]. The second factor is then appended to the HTML form and transmitted to the web application together with the username and password. The web application verifies the second factor by performing the same computation that happened in the browser. It denies access to the user account if the verification fails.

A phisher could lure the user into visiting his prepared page. Given that the user does not detect the attack, she enters her username and password and sends them to the attacker's site. Then, the attacker tries to log into the user's account exploiting the phished credentials. The web application, however, denies access because the necessary browser token is not available to compute the valid second factor.

There are scenarios where a browser is not only used by one user but at least two where both have an account on the same web application respectively, e.g., a family sharing one laptop (and OS account) or tablet PC. In this case, they would share the same browser token. This is also true for guests accessing the web application via this browser just once. We prevent such unintended sharing of the browser token by assigning it to the respective user account in the browser's storage, i.e., the second factor can only be computed if a browser token associated with the given username is found.

3.4 Browser Enrollment

The idea how PhishSafe proceeds has been described above. What remains is PhishSafe's bootstrapping, i.e., the process that establishes the token in the user's browser. There are two options when the token is stored: during account setup or, afterwards, whenever the user logs in from a previously unknown browser.

The web application can set a token during the registration process unless the user opts out, e.g., because she uses a friend's device. After the user chooses username and password, the token is stored in the browser's web storage.

3.4.1 Restoring the Browser Token

We assume that account information includes the user's email address and leverage this as a second channel for token installation. Given that the user uses more than one device to access the web application, changes her browser, reinstalls her operating system or firmware, or just deletes the browser's web storage for privacy reasons, she needs an opportunity to restore her browser token. The password

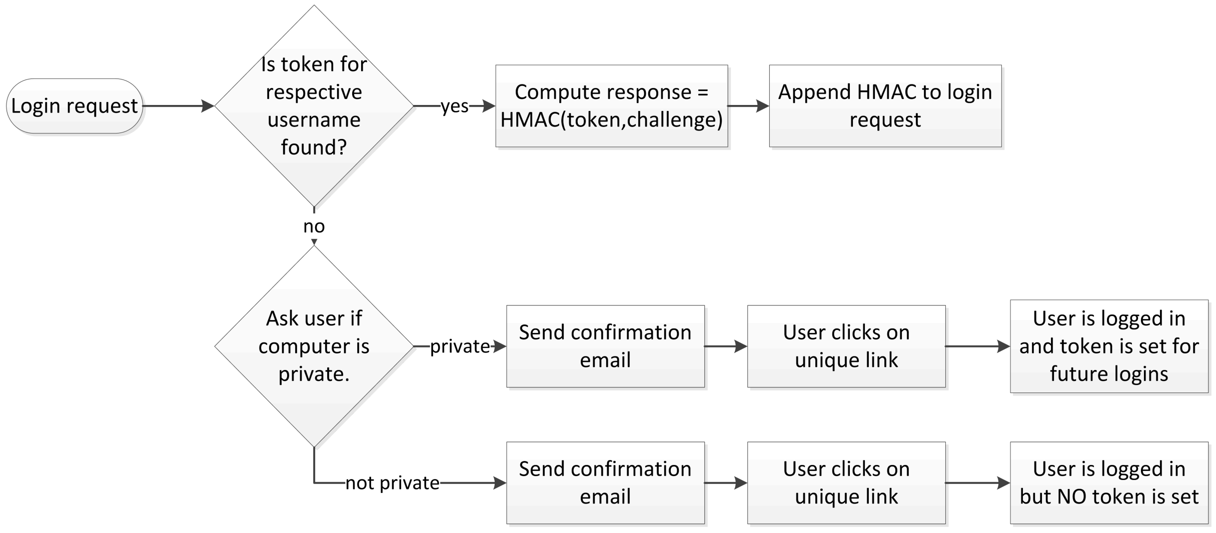

Login request
Is token for respective username found?
yes
Compute response = HMAC(token,challenge)
Append HMAC to login request
no
Ask user if computer is private.
private
Send confirmation email
User clicks on unique link
User is logged in and token is set for future logins
not private
Send confirmation email
User clicks on unique link
User is logged in but NO token is set

Figure 3: Authentication Token Logic

alone is insufficient because the attacker can learn it and so use it to equip his browser with a valid token.

Usual second authentication tokens are not sufficient, either. Examples for this class are apps or devices that issue two-step verification numbers, e.g. Google Authenticator [15] and RSA SecurID [10], as well as one-time passcodes sent to the cell phone or by email. A phishing attacker could lure the victim on his page and at the same time request the original login page. When the victim provides her password, he forwards it and is prompted with an input field for the second authentication step. Then, he leads his victim to believe that her browser token needs to be reset and requests the same authentication credential that he is supposed to enter. Finally, he only needs to forward the user's second factor credential to finally own the password and the browser token. Note that this attack even works with passcodes sent to the user's cell phone because the application indeed sends such a code to the user (upon the attacker's request). We believe that receiving the code makes the actual phishing attack even more credible. The attacker acts as a man in the middle.

Our authentication scheme uses complete URLs that must be clicked (or copied and pasted to the browser) by the user. When the user enters her username and password on the login page but no respective browser token is found, the web application sends a confirmation email to her account. The email contains a unique URL that must be accessed within the same session context as the login request. In the attack scenario described above, the attacker's login attempt triggers the email confirmation. However, if the user clicks on the provided link, she accesses the real web application but not the phishing page. At that point, the attack becomes detectable for the web application because the session context does not match. The attacker never obtains the necessary input to obtain a valid browser token.

3.4.2 *One-Time Account Access*

Finally, the user might use a public computer to access the web application. So, a persistent credential in the browser's web storage is not appropriate. For this reason, PhishSafe also provides one-time access. The only difference to the above described browser token reset is that no token is stored. After the user enters her username and password and no browser token for this username is found, she is asked if she is using a public computer or if she trusts all users of this computer and uses it regularly. In both cases, the web application sends an email with a unique URL. However, if the user requests the URL from a public computer, no browser token is set and access to the account is granted only once. The token handling logic is given in Fig. 3.

3.5 Protection against the XSS Attacker

The authentication process described above perfectly protects against the phishing attacker (see Sec. 2.2). The XSS attacker, however, could inject JavaScript code that is executed within the same domain context as the web application. This allows him to read the browser's web storage and obtain the secret browser token. For this reason, we move the token and all related computations to a secure subdomain. Given that the actual web application runs on `www.example.com`, a subdomain, e.g., `auth.example.com`, is responsible to handle and store the secure browser token. This subdomain only contains static JavaScript dedicated to this task and nothing else. Based on this, we consider well audited and XSS-free code to be feasible. An HTML document served from the subdomain and embedded into the main web application as an invisible iframe contains the JavaScript code. This way, we leverage the guarantees provided by the same-origin policy [58] and the postMessage API [50] to prohibit access by the XSS attacker to the browser token while enabling controlled interaction between the web application and the secure subdomain.

Hence, the challenge appended to the login form is submitted to the secure subdomain via JavaScript and the postMessage API. The code of the subdomain computes the HMAC of the challenge using the browser token. The HMAC is then sent back to the original document and attached to the subsequent login request (see Fig. 4). Please note that all this communication happens within the browser.

4. IMPLEMENTATION

The implementation of our proposed authentication scheme comprises three interacting components: the *TokenManager* handles the browser token in the secure subdomain, the *Authenticator* assembles necessary input for the login, and the server-side *AccountManager* fits into legacy or new web applications in order to handle browser challenges and responses.

4.1 Client-side Components

We first describe the client-side components, the TokenManager that is loaded in the iframe from the secure subdomain and the Authenticator that delivers the web application's challenge to the TokenManager and appends the retrieved response to the HTML login form.

4.1.1 The TokenManager

The TokenManager implements the necessary functions to fetch the browser token, compute the HMAC of the token and the web application's challenge, and return an error message if no token is found. It runs in the domain context of a secure subdomain. We use the CryptoJS[5] library as an implementation of the cryptographic functions.

The TokenManager uses the browser's `localStorage` part of the web storage [18]. Web storage is supported by all major browsers on mobile and desktop platforms which makes our authentication scheme platform and browser independent. The localStorage is persistent, i.e., it is not cleared on a regular basis as the `sessionStorage` is. The storage is limited in size per origin between 5 and 25 Mbytes depending on the browser which, however, is far more than necessary for our purposes. Web storage is meant for pairs of identifiers and values where both must be strings. More complex data structures can be stored as JSON objects [5] that are easily converted to string and back. The TokenManager uses JSON to store the username and the associated browser token.

The communication interface of the TokenManager is restricted to a function that expects a challenge and a username as input and provides an HMAC as the output (see Fig. 4). The Authenticator's direct access to the subdomain's localStorage is prohibited by the same-origin policy [58]. So, the Authenticator needs to use the JavaScript `postMessage` API [50] that enables two web documents in a browser to communicate across origin boundaries in a secure manner. A `postMessage(msg, target)` call expects a message string and the target origin as parameters. The receiving document needs to register an event handler to receive a message. The triggered event comes with additional metadata provided by the browser, e.g., the origin of the sender. This allows the receiver,i.e., the TokenManager, to carefully check the sender's authenticity.

4.1.2 The Authenticator

The HTML login form contains two additional hidden fields for PhishSafe: `AuthChallenge` and `AuthResponse`. The first contains the web application's challenge, the second is initially blank. The Authenticator reads the challenge and the user's username from the input field. It passes both arguments to the TokenManager and reads back the answer. The answer either contains the computed HMAC or an error. In the success case, the Authenticator rewrites the login form's `AuthResponse` field to append the response. It prompts the user if the TokenManager reported that no browser token was found (see Fig. 5).

[5]`http://code.google.com/p/crypto-js/`

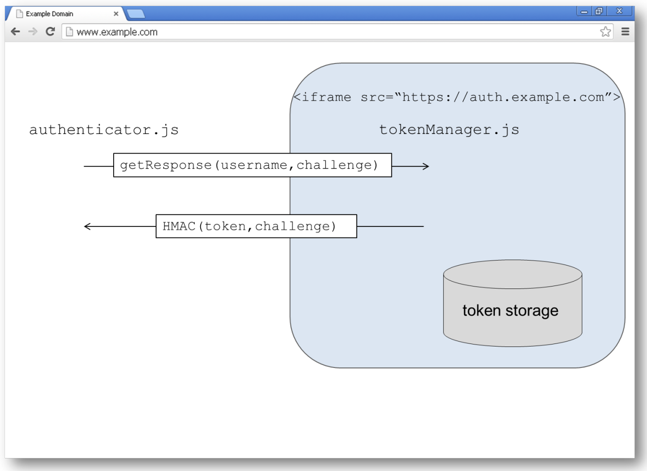
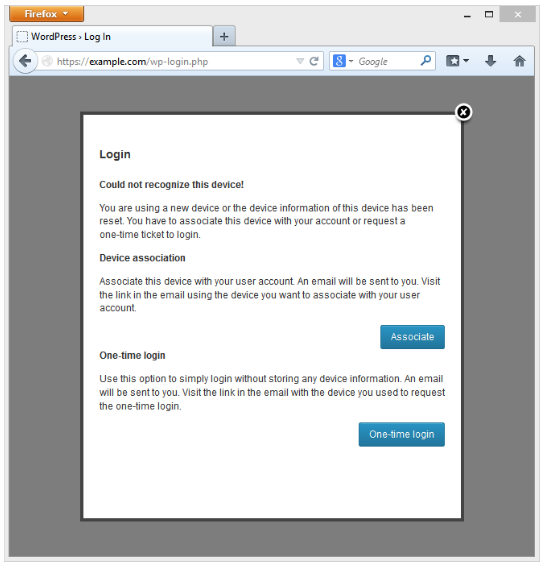


Figure 4: Domain-Isolated Token Storage



Login

Could not recognize this device!

You are using a new device or the device information of this device has been reset. You have to associate this device with your account or request a one-time ticket to login.

Device association

Associate this device with your user account. An email will be sent to you. Visit the link in the email using the device you want to associate with your user account.

Associate

One-time login

Use this option to simply login without storing any device information. An email will be sent to you. Visit the link in the email with the device you used to request the one-time login.

One-time login

Figure 5: User Prompt If No Browser Token Is Found

4.2 Server-side Component

The server-side part of PhishSafe consists of a single component, the AccountManager.

4.2.1 The AccountManager

The AccountManager implements the server-side part of our authentication scheme. We equipped WordPress with the AccountManager as a plugin. The integration required only reasonable efforts and no changes of the application code due to the modular architecture of WordPress together with the hooking feature. We consider the integration into modular or new web applications as an easy task while necessary efforts might be bigger for non-modular legacy applications.

The AccountManager issues the user's browser token and adds the invisible iframe and the Authenticator to the web application's login page. It generates a new challenge for every user login, adds the `AuthChallenge` and `AuthResponse` fields to the HTML login form, and checks incoming login requests for valid responses.

The XSS attacker could inject a payload that reads the user's username, password, and the returned HMAC for authentication. Having all this information, he can log into the web application without a valid browser token. For this reason, the AccountManager sets a cookie in the user's browser. This cookie has a random value that is saved by the AccountManager together with the user's current challenge. Only login requests that carry this cookie and the valid response are processed. The cookie has the `HttpOnly` and the `Secure` flags set to prevent it from being read by the attacker's payload or during plain `http` transport.

4.2.2 Security Configuration

Though it is not part of our attacker model, we leverage two more modern security features to shelter from SSL stripping [26] and pharming [49] attacks. A man-in-the-middle attacker performing an SSL stripping attack could prevent the user's browser from using `https` and then read transmitted information or even inject code into the document loaded in the invisible iframe. A pharming attacker could serve own code on behalf of the abused web application and so bypass the same-origin policy. Both attackers could hijack the secure browser token.

To overcome these attacks, the AccountManager adds HTTP Strict Transport Security (HSTS) [19] and Public Key Pinning (PKP) [11] policy headers to the web application's HTTP responses. HSTS makes sure that the browser only contacts the website using `https`. There is an inherent bootstrapping problem before the first request, i.e., the website must ensure that the browser eventually receives the HSTS header. Google Chromium and Mozilla Firefox overcome this problem using a static list of pre-defined domains[6]. The PKP policy prevents that a pharming attacker presents his own certificate. After receiving the policy header, the browser only accepts SSL certificates with the pinned public key.

5. EVALUATION

We evaluate the security properties of our authentication scheme and validate our design goals from Section 3.1.

5.1 Security Evaluation

We first explain how far PhishSafe protects users against the attackers defined in Section 2.2. Then, we give details of further security properties, and finally, we identify open issues of the proposed scheme.

5.1.1 The Phishing Attacker

The phishing attacker counterfeits the design of a web application in scope and lures victims. The latter can be done either by email or by links and ads on other websites. In any case, our proposed authentication scheme does not prevent a victim from accessing the phishing page. However, the information the phisher can obtain is not sufficient to abuse the user's account because the web application denies access if no valid response is appended to the login request.

5.1.2 The XSS Attacker

The XSS attacker differs from the phishing attacker in his ability to execute JavaScript code on vulnerable domains. For instance, the XSS attacker could perform a reflected XSS attack by sending a specially crafted link via email or IM. The injected payload can read the username, password, the current challenge for login and the respective response if a browser token is stored in the browser. However, we assumed that the attacker can not break cryptography, thus, he can not compute the browser token from the challenge and the response. The captured data is still insufficient because it lacks the related cookie which is inaccessible to JavaScript. Finally, we consider the task to develop invulnerable static code for the secure subdomain feasible such that there is no attack vector for the XSS attacker.

5.1.3 Further Security Advantages

The proposed authentication approach comes with additional security features.

First of all, though running purely in the browser, the scheme thwarts email-based phishing attempts. Phishers try to evade browser-based protection by asking the victims to reply to their phishing emails and give credentials in the email. The obtained username and password, however, do not give an attacker access to the user's web account. We consider phishing for the browser token to be infeasible as it requires major efforts and advanced knowledge of a user to read the token from the browser's web storage.

Phishing attacks rely on unprepared users that disclose credentials. This general observation holds true for any kind of credentials a user may know. So, second authentication factors that must be entered by the user in a web form are inherently susceptible to phishing attacks, too. Examples include one-time passcodes sent to the user's cellphone or generated by apps. Our approach utilizes complete URLs to overcome second factor phishing. A user clicking on a link not only proves access and knowledge but is also directed to the right web application.

5.1.4 Open Issues

Next, we emphasize on potential attacks on our authentication scheme.

Our approach relies on the security of the user's email account. The attacker can request and read the confirmation URL sent to the user if he has access to the email account and the user's credentials to the target web application. We are here in line with today's best practices for password reset as virtually all web applications offer email-based processes at least if no cell phone number is given or the attacker pretends the cell phone is stolen.

An email account can not be protected if the confirmation URL is sent to the same address. There are two options to make sure that the user can always access the confirmation URL: First, the user can provide an alternative email account where the confirmation URL is sent to. Second, the email provider can offer application-specific passwords[7]. These passwords are chosen by the provider and not remembered by the user. Instead, they are stored by client appli-

[6]http://www.chromium.org/sts

[7]https://support.google.com/accounts/answer/185833

cations to obtain access to the user's account. Given an application-specific password and an email client on a PC or mobile device, access to the confirmation URL is assured.

An attacker can try to acquire the confirmation URL sent to the user by making the user enter it into a prepared input field on the phishing site. The easiest way to avoid this is to make the user click on the link. Moreover, a highlighted warning in the email reduces the attacker's chances.

A window of vulnerability towards a pharming attacker remains before the first PKP header is received by the user's browser (see Section 4.2.1). As long as the browser did not pin the server's public key, a pharming attacker can present a spoofed certificate and submit malicious content. In the future, a similar pre-defined list of certificates might be implemented as it happened for HSTS.

Finally, though completely out of scope of our authentication scheme, fully fledged spyware can read and transmit the browser token. Less elaborate keyloggers, however, are ineffective because the browser token is never entered via the keyboard.

5.2 Validation of Design Goals

In this section, we evaluate the compliance of PhishSafe with the design goals given in Section 3.1.

Sidestep the arms race between phishers and the anti-phishing blacklist community: Our approach does not exploit features of phishing sites or emails for classification. So, there is no motivation for actions and reactions. In fact, we do not consider phishing activities at all but only hide some piece of information from phishers. We argued in the section above that phishers can hardly learn the browser token.

Reduce reliance on the user: The security of the approach barely relies on the user. She neither needs to enter her credentials only when a dedicated indicator is shown, nor does she need to remember additional credentials. In fact, the only change compared to her used workflow is the decision if a computer is trusted or not and the click on the confirmation link. The actual login procedure does not change at all on regularly used browsers.

Avoid dependence on the browser's interface: PhishSafe does not depend on the browser's interface, nor does it change the browser's appearance. This feature not only avoids confusing the user but is one important aspect of cross-platform applicability (see below).

Waive the need for additional devices and the installation of protective tools PhishSafe only needs an off-the-shelf browser and neither relies on extensions, nor toolbars, nor third-party plug-ins, like Flash or Silverlight. A second device is also not necessary.

Withstand the attackers defined in Section 2.2 We showed in Section 5.1 that PhishSafe resists attacks by the phishing attacker and the XSS attacker.

Further points: PhishSafe runs on mobile as well as desktop browsers because all modern browsers support the WebStorage and postMessage APIs. It does not rely on visual indicators which makes it applicable on mobile devices with limited screen size.

PhishSafe can easily complement other approaches. If the browser maintains a blacklist of phishing sites, it can prevent that the user reveals her credentials. Approaches leveraging a secure password entry field to some degree also work together with PhishSafe even though details need to be sorted out.

6. RELATED WORK

There is a long history of approaches to overcome phishing attacks. We classify the existing body of work into three categories: approaches that augment the visible user interface with trust indicators (Sec. 6.1), approaches leveraging sophisticated authentication protocols to prevent that the real password is sent to the attacker (Sec. 6.2), single sign-on protocols (Sec. 6.3), and approaches aiming to distinguish between reliable and phishing sites (Sec. 6.4).

6.1 Augmenting the User Interface

A number of approaches tries to protect the user from phishing attempts using individual authenticity features. The overall goal is to ensure that the user enters her password only if a pre-shared symbol indicates trustworthiness. Basic approaches just embed personalized images in the login page [40, 48].

Other approaches require the installation of client-side extensions to tune the browser's user interface. They display custom names, logos, and the certification authority (CA) of the visited website [17], open personalized windows including user-defined pictures [6], combine images with custom names of websites [56], or use colored frames to indicate the website's trust level [53, 55].

This class of approaches burdens the user with challenging tasks, including

- remembering a visual authenticity feature [40, 48, 17, 56],
- tolerating adverse impacts on usability and browsing experience [53],
- passing complex setup processes, for instance, choosing site labels, master passwords, appropriate protection service providers, and finally start the protection feature by hand [56],
- manually maintaining a list of supporting sites and compare two displayed pictures to authenticate the server before login [6], and
- install a dedicated browser [55].

Finally, these approaches are not portable and require support by the server and the client, thus being subject to a chicken-and-egg problem.

6.2 Sophisticated Authentication Protocols

The second class of phishing mitigation approaches applies changes to the common username and password based authentication. The main goal is to not submit the password in plaintext to an unauthenticated remote server but mutually authenticate client and server [16, 42, 47], utilize a zero-knowledge protocol to avoid transmitting confidential information [22], check for user-specific knowledge that changes over time [31], use trusted second devices to establish an authenticated session [34], generate site-specific passwords from a seed [39], or use bookmarks as a secure entry point [1].

The implementation of non-standard authentication protocols by design requires effort on both communication parties for support. The user either needs to store and maintain a particular bookmark for every protected website [1], remember to activate protection before entering her credentials [39], install a plugin [47] or a browser toolbar and regularly verify that it is not spoofed [42], use a dedicated browser [22] or a second device that must be trustworthy but also able to establish a direct connection with the browser [34], or remember every past action with respect to this account [31], while some approaches are not implemented or practically evaluated [16].

6.3 Single Sign-On

A set of so-called single sign-on protocols aims at releasing the user from maintaining one unique password for each web account respectively, among them OpenID [36], Mozilla Persona [30] (aka BrowserID), SAML [24], and Shibboleth [20]. They allow a user to login once with a single authority in order to access several accounts at different providers.

The distributed authorization protocol OAuth [33] is used in some cases to log into third party web applications, too. A previous login with the provider, usually a social network, is required.

These protocols decrease credential management overhead caused by the trend of an increasing number of web accounts. Nevertheless, the user must log in once in order to apply such a protocol. In this respect, PhishSafe complements those approaches to secure the one remaining login.

6.4 Detecting Phishing Sites

This class of approaches tries to identify phishing sites in order to warn the user and prevent information leakage. There are three main vectors for site classification: First, approaches use web crawlers to check websites for phishing features [51, 54, 13]. These approaches utilize machine learning algorithms to update their classification criteria. They generate blacklists of suspicious domains. Browsers can download those blacklists and warn the user whenever she accesses a listed site. The delay between the setup of a phishing site and the time it is listed in the browsers grants phishers a temporal advance.

Approaches of the second vector attempt to classify visited websites in real time [21, 37, 3, 38, 27]. These approaches do not suffer the time delay the blacklisting approaches have. However, they create an overhead for examination for every page access.

Finally, some approaches feed suspicious websites with bogus credentials and observe the reaction on those spoofed login requests [2, 57, 41]. The point is that phishing sites supposedly accept all combinations of username and password or always answer with an error message.

All described vectors for classification of phishing sites are part of an arms race with phishers. The approaches rely on features that can be easily changed by phishers to circumvent classification. In a next step, the classification criteria can be adjusted and so on. Moreover, classification is always prone to mistakes, i.e., genuine websites may be classified as phishing attempts while phishing sites are treated as genuine. Nevertheless, phishing detection approaches can serve as a first line of defense and complement PhishSafe to prevent leakage of username and password.

7. CONCLUSION

In the course of this paper, we analyzed the root causes for the continuing prevalence of phishing attempts and classified existing solutions into three main categories: incomplete countermeasures prone to false positives and false negatives, countermeasures cumbersome for the user compared to common logon processes, and countermeasures relying on browser extensions or toolbars, thus, expecting risk-awareness by the user and excluding users of not supported browsers and platforms.

Then, we identified the user as the weakest link when it comes to phishing protection. She can neither be expected to apply cumbersome countermeasures, nor install protective tools, nor take care of security indicators. We presented PhishSafe, a reliable approach to overcome phishing attacks, that runs in browsers out of the box and barely changes known logon processes. The user must only decide if she uses her private browser or not. This way, PhishSafe implements a two-factor authentication scheme where the second factor is only accessible to the genuine web application but not to the phisher nor to the user. Without knowing the second factor, the user cannot disclose the necessary information for account access to an attacker.

PhishSafe can be easily deployed by web application providers and is not susceptible to the chicken-and-egg problem. Moreover, it complements SSO protocols and anti-phishing blacklists.

Acknowledgments

This work was in parts supported by the EU Project WebSand (FP7-256964), `https://www.websand.eu`. The support is gratefully acknowledged.

8. REFERENCES

[1] B. Adida. BeamAuth: Two-Factor Web Authentication with a Bookmark. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, 2007.

[2] M. Chandrasekaran and R. Chinchani. PHONEY: Mimicking User Response to Detect Phishing Attacks. In *International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM 2006)*, 2006.

[3] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. C. Mitchell. Client-Side Defense against Web-based Identity Theft. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '04)*, 2004.

[4] D. Crocker, T. Hansen, and M. Kucherawy. DomainKeys Identified Mail (DKIM) Signatures. RFC 6376, `http://tools.ietf.org/html/rfc6376`, (09/03/13).

[5] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627, `http://tools.ietf.org/html/rfc4627`, (09/03/13).

[6] R. Dhamija and J. D. Tygar. The Battle Against Phishing: Dynamic Security Skins. In *Proceedings of the 2005 Symposium on Usable Privacy and Security (SOUPS '05)*, 2005.

[7] J. S. Downs, M. B. Holbrook, and L. F. Cranor. Decision Strategies and Susceptibility to Phishing. In *Symposium On Usable Privacy and Security (SOUPS)*, 2006.

[8] Dr. Web. New Trojan steals short messages. [online], `http://news.drweb.com/show/?i=3549`, (09/12/13).

[9] S. Egelman, L. F. Cranor, and J. Hong. 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 (CHI '08)*, 2008.

[10] EMC Corporation. RSASecurID. [online], `http://www.emc.com/security/rsa-securid.htm`, (09/12/13).

[11] C. Evans, C. Palmer, and R. Sleevi. Public Key Pinning Extension for HTTP. Internet-Draft, `http://tools.ietf.org/html/draft-ietf-websec-key-pinning-08`, (09/10/13).

[12] A. P. Felt and D. Wagner. Phishing on Mobile Devices. In *Web 2.0 Security and Privacy (W2SP)*, 2011.

[13] I. Fette, N. Sadeh, and A. Tomasic. Learning to Detect Phishing Emails. In *Proceedings of the 16th international conference on World Wide Web (WWW '07)*, 2007.

[14] E. Gabrilovich and A. Gontmakher. The Homograph Attack. *Communications of the ACM*, 45, 2002.

[15] Google. Authenticator. [online], `http://code.google.com/p/google-authenticator/`, (09/12/13).

[16] M. G. Gouda, A. X. Liu, L. M. Leung, and M. A. Alam. SPP: An anti-phishing single password protocol. *Computer Networks*, 51(13):3715 – 3726, 2007.

[17] A. Herzberg and A. Jbara. Security and identification indicators for browsers against spoofing and phishing attacks. *ACM Transactions on Internet Technology (TOIT)*, 2008.

[18] I. Hickson. Web Storage. [online], `http://www.w3.org/TR/webstorage/`, (09/10/13).

[19] J. Hodges, C. Jackson, and A. Barth. HTTP Strict Transport Security (HSTS). RFC 6797, `http://tools.ietf.org/html/rfc6797`, (09/10/13).

[20] Internet2. Shibboleth. [online], `http://shibboleth.net/`.

[21] E. Kirda and C. Kruegel. Protecting Users agains Phishing Attacks with AntiPhish. In *29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, 2005.

[22] P. Knickerbocker. Combating Phishing through Zero-Knowledge Authentication. Master's thesis, Graduate School of the University of Oregon, 2008.

[23] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, `https://tools.ietf.org/html/rfc2104`, (09/03/13).

[24] H. Lockhart and B. Campbell. SAML V2.0. `https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf`, March 2008.

[25] C. Ludl, S. McAllister, E. Kirda, and C. Kruegel. On the Effectiveness of Techniques to Detect Phishing Sites. In *Proceedings of the 4th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '07)*, 2007.

[26] M. Marlinspike. New Tricks For Defeating SSL In Practice. Talk at BlackHat '09, `http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf`, (09/10/13).

[27] E. Medvet, E. Kirda, and C. Kruegel. Visual-Similarity-Based Phishing Detection. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm '08)*, 2008.

[28] Microsoft. SenderID. [online], `http://www.microsoft.com/senderid`, (09/03/13).

[29] K. D. Mitnick and W. L. Simon. *The Art of Deception: Controlling the Human Element of Security*. John Wiley & Sons, 2002.

[30] Mozilla. Persona. [online], `https://developer.mozilla.org/en-US/docs/Mozilla/Persona`, (09/03/13).

[31] N. Nikiforakis, A. Makridakis, E. Athanasopoulos, and E. P. Markatos. Alice, What Did You Do Last Time? Fighting Phishing Using Past Activity Tests. In *Proceedings of the 3rd European Conference on Computer Network Defense*, 2009.

[32] Y. Niu, F. Hsu, and H. Chen. iPhish: Phishing Vulnerabilities on Consumer Electronics. In *Proceedings of the 1st Conference on Usability, Psychology, and Security (UPSEC '08)*, 2008.

[33] OAuth. [online], `http://oauth.net/`, (09/03/13).

[34] B. Parno, C. Kuo, and A. Perrig. Phoolproof Phishing Prevention. In *Proceedings of the 10th International Conference on Financial Cryptography and Data Security (FC'06)*, 2006.

[35] T. Raffetseder, E. Kirda, and C. Kruegel. Building Anti-Phishing Browser Plug-Ins: An Experience Report. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems (SESS '07)*, 2007.

[36] D. Recordon and D. Reed. OpenID 2.0: a platform for user-centric identity management. In *DIM*, 2006.

[37] V. P. Reddy, V. Radha, and M. Jindal. Client Side Protection from Phishing Attack. *International Journal of Advanced Engineering Sciences and Technologies (IJAEST)*, pages 39–45, 2011.

[38] A. P. E. Rosiello, E. Kirda, C. Kruegel, and F. Ferrandi. A Layout-Similarity-Based Approach for Detecting Phishing Pages. In *Proceedings of the third International Conference on Security and Privacy in Communication Networks (SecureComm 2007)*, 2007.

[39] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *Proceedings of the 14th Usenix Security Symposium (USENIX 2005)*, 2005.

[40] RSA Data Security. SiteKey. [Hosted at Bank of America], `https://www.bankofamerica.com/privacy/online-mobile-banking-privacy/sitekey.go`, (08/01/13).

[41] H. Shahriar and M. Zulkernine. PhishTester: Automatic Testing of Phishing Attacks. In *Fourth International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, 2010.

[42] M. Sharifi, A. Saberi, M. Vahidi, and M. Zorufi. A Zero Knowledge Password Proof Mutual Authentication Technique Against Real-Time Phishing Attacks. In *Third International Conference on Information Systems Security (ICISS 2007)*, 2007.

[43] S. Sheng, B. Wardman, G. Warner, L. F. Cranor, J. Hong, and C. Zhang. An Empirical Analysis of Phishing Blacklists. In *Sixth Conference on Email and AntiSpam (CEAS 2009)*, 2009.

[44] The Anti-Phishing Working Group (APWG). Global Phishing Survey: Domain Name Use and Trends in 2H2012. [online], `http://docs.apwg.org/reports/APWG_GlobalPhishingSurvey_2H2012.pdf`, (09/03/13).

[45] The Anti-Phishing Working Group (APWG). Phishing Activity Trends Report, 1st Quarter 2013. [online], `http://docs.apwg.org/reports/apwg_trends_report_q1_2013.pdf`, (09/03/13).

[46] The Anti-Phishing Working Group (APWG). Phishing Attack Trends Reports. [online], `http://www.apwg.org/resources/apwg-reports/`, (09/03/13).

[47] H. Tout and W. Hafner. Phishpin: An Identity-Based Anti-Phishing Approach. In *International Conference on Computational Science and Engineering (CSE '09)*, 2009.

[48] L. Varteressian. Yahoo! Sign-In Seal. [online], `http://security.yahoo.com/sign-seal-000000996.html`, (08/01/13).

[49] B. Violino. After Phishing? Pharming! [online], `http://www.csoonline.com/article/220629/after-phishing-pharming-`, (09/10/13).

[50] WHATWG. Cross-document messaging. [online], `http://www.whatwg.org/specs/web-apps/current-work/multipage/web-messaging.html`, (09/06/13).

[51] C. Whittaker, B. Ryner, and M. Nazif. Large-Scale Automatic Classification of Phishing Pages. In *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS '10)*, 2010.

[52] M. Wu, R. C. Miller, and S. L. Garfinkel. Do Security Toolbars Actually Prevent Phishing Attacks? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*, 2006.

[53] M. Wu, R. C. Miller, and G. Little. Web Wallet: Preventing Phishing Attacks by Revealing User Intentions. In *Proceedings of the Second Symposium on Usable Privacy and Security (SOUPS '06)*, 2006.

[54] G. Xiang, J. Hong, C. P. Rose, and L. Cranor. CANTINA+: A Feature-rich Machine Learning Framework for Detecting Phishing Web Sites. *ACM Transactions on Information and System Security (TISSEC)*, 2011.

[55] Z. E. Ye and S. Smith. Trusted Paths for Browsers. In *Proceedings of the 11th USENIX Security Symposium (USENIX 2002)*, 2002.

[56] K.-P. Yee and K. Sitaker. Passpet: Convenient Password Management and Phishing Protection. In *Proceedings of the Second Symposium on Usable Privacy and Security (SOUPS '06)*, 2006.

[57] C. Yue and H. Wang. Anti-Phishing in Offense and Defense. In *Annual Computer Security Applications Conference (ACSAC 2008)*, 2008.

[58] M. Zalewski. Browser Security Handbook, part 2. [online], `http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy`, (09/03/13).

[59] Y. Zhang, S. Egelman, L. Cranor, and J. Hong. Phinding Phish: Evaluating Anti-Phishing Tools. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS 2007)*, 2007.