Spyware Resistant Web Authentication Using Virtual Machines

ABSTRACT

Password collection by keyloggers and related malware is increasing at an alarming rate. We investigate client-only defenses and methods that require server-side assistance. Password hashing and password injection, in which passwords are isolated from spyware, provide protection against phishing, common-password attacks, and spyware on the client platform. To protect against network sniffing and dictionary attacks, we suggest an appropriate combination of password-authenticated key exchange (PAKE) and SSL. As further defense against pharming, cookie sniffing, and session hijacking, we propose a form of transaction confirmation over an authenticated channel. Our implemented and freely distributed client-side system providing all of these mechanisms consists of two components: a browser extension that runs in an untrusted environment, and an authentication agent that runs in an environment that is protected from spyware. Using a virtual machine monitor, the trusted and untrusted components can both run on the same physical machine.

1 Introduction

Web password theft, carried out via phishing attacks, pharming attacks, and keyloggers, is a serious and growing problem. The Anti-Phishing Working Group (APWG) [4] reports over 170 different types of keyloggers distributed on thousands of web sites. The APWG statistics all point to a dramatic increase in the use of phishing-like email to lure unsuspecting users to web sites containing keyloggers and other forms of malware. In contrast to traditional phishing attacks that steal a single user password, these keyloggers may steal all user info (including passwords) entered at many different sites. One reason that modern browsers are vulnerable is that they must interact with a great deal of third-party code, including plugins, extensions, applets, and JavaScript. Therefore, even if an underlying operating system provides advanced security, the browser itself is susceptible to compromise. The widespread penetration of end-user machines by spyware and bot networks [27] underscores the vulnerability of many of today's machines to keylogging attacks.

In this paper, we describe a range of attacks that may be carried out using spyware, and present effective defenses that may be implemented by client functionality only, or by combining client and server functions. The attacks we consider include deceptive phishing-style web sites, break-ins at weak-security sites that yield passwords usable elsewhere, logging keystrokes, network sniffing or site-in-the-middle, cookie theft, pharming, and malware-based session hijacking. Our client-side defenses (including the client-side portion of client-server defenses) are all implemented in two client-side components freely distributed as a single download: a browser extension that runs in an untrusted environment, and an authentication agent that runs in an environment that is protected from spyware. Using a virtual machine monitor, the trusted and untrusted components can both run on the same physical machine.

The main idea behind our two-part approach is that the authentication agent provides a trusted path to the user for obtaining authentication credentials, while the untrusted browser extension participates in various types of communication, without direct access to authentication credentials. For example, if deployed without server-side accommodations, the user may supply a password to the authentication agent using a trusted path, and the authentication agent may use *password injection* to insert this password into an outgoing transmission without exposing the password to the untrusted environment. Alternatively, if the server supports password authenticated key exchange (PAKE), then properties of PAKE protocols allow messages from the authentication agent to the server to safely pass through the untrusted application environment.

Briefly, phishing involves luring a user to a replica of a legitimate website, and pharming compromises the domain name system (DNS) to direct the user to a malicious site. The common-password problem arises because many users supply the same password to many sites. Malware can potentially not only log keystrokes to capture passwords and other sensitive information, but may initiate fraudulent transactions with a server once the user is authenticated, or export cookies or other information. Our system protects against phishing, common-password attacks, and keystroke logging by password hashing and password injection, without requiring any server assistance. If the server supports PAKE, then protection is also provided against network sniffing and dictionary attacks. With server support for transaction confirmation leveraging the trusted authentication agent interface, our system protects against pharming, cookie theft, and session hijacking. Although many prior antiphishing tools such as [9, 11, 29, 31] protect unsuspecting users against phishing web sites, none of these previous countermeasures protect against malware on the user platform.

The security of our approach relies on isolation between the untrusted environment that contains user applications and spyware, and the trusted environment supporting the authentication agent. In the configuration we have used most frequently, a Virtual Machine Monitor (VMware) provides isolation while allowing both client-side processes to run on the same physical machine. Alternatively, lighter-weight isolation methods such as FVM [38] or a windows-based jail [15] could be used. While these methods may not provide the same security of a VMM, they may have other benefits. For greater resilience, specialized hardware such as smart cards, secure co-processors, or Bluetooth-enabled cell phones could run the authentication agent. However, virtualization seems to provide many of the advantages of specialized hardware without the associated cost or inconvenience.

We summarize the threats addressed by SpyBlock in section 2 and describe the SpyBlock architecture in section 3.1. Section 3.2 explains the function of the authentication agent, including methods used to assure the user that any password dialog generated by the authentication agent actually comes from the authentication agent, while Section 4 describes the authentication protocols. Transaction confirmation is discussed in section 4.1.2. Implementation techniques and security challenges are described in section 5, with additional comments on related work in 6. Concluding remarks appear in section 7.

2 Web authentication threats

Conventional web authentication asks the user to enter a plaintext password, which is then transmitted to the server over http or https. When https is used, the password is encrypted in transmission, and some measure of server authentication is provided by the SSL/TLS certificate check. However, the certificate check is not always effective in practice, because users do not notice if http is used in place of https, and users may ignore https warning messages [36]. Even worse, because anyone can buy a certificate, users who click on a link in a phishing email can end up at a properly certified https site that happens to be a phishing site. Once authenticated, malware on the authenticated client platform may generate actions that are not intended by the user. SpyBlock provides

security measures that address all these threats. More accurately, SpyBlock defends against the following threats:

- **Common password.** Although this practice is discouraged, users often have the same password for several accounts. If the user's password at one account is compromised, either due to local malware, a network attack, or a break-in at the remote web server, other user accounts that use the same password are at risk. We consider a system secure against common password attacks if the attacker obtains only a hash of the password and must perform a dictionary attack in order to recover an unhashed password that can be used at other accounts. Dictionary attacks can be expensive for strong passwords, and various techniques to further strengthen password hashes are available [16, 31, 30].
- **Phishing.** The network is trustworthy, but the user has connected to a malicious spoof site that is visually similar to another site that she has an existing relationship with. This malicious spoof site may act as an online man-in-the-middle. The system is considered secure if the spoof site cannot authenticate as the user at the real site.
- **Spyware keylogger.** The client machine that is used to browse the web has a malicious program running on it that may be passively recording keystrokes and sending them to the attacker. However, it is not actively altering outbound requests. The system is considered secure if the attacker cannot authenticate as the user from another machine.
- **Network password sniffing.** An attacker can examine network packets but cannot alter them. The system is considered secure if the attacker cannot obtain a password that can later be used to authenticate as the user. HTTPS is sufficient for preventing these attacks.
- **Network cookie sniffing.** An attacker can examine network packets but cannot alter them. The system is considered secure if the attacker cannot remotely hijack the user's session by stealing the short-lived session cookie. HTTPS is sufficient for preventing these attacks, but many sites such as Hotmail [17] and Amazon [2] only use HTTPS for the login process, and use HTTP elsewhere for performance reasons.
- **Pharming.** The user enters the correct DNS hostname into the browser, but due to a compromise of the DNS server, this hostname resolves to the IP address of a malicious attacker that may be operating a phishing site or acting as a man-in-the-middle. The system is considered secure if the spoof site cannot authenticate as the user at the real site.
- **Malware session hijacking.** The client machine that is used to browse the web has a malicious program running on it that may actively alter requests or perform unwanted requests on behalf of the user. The system is considered secure as long as no unauthorized transaction is completed.

3 SpyBlock architecture and authentication agent

3.1 SpyBlock architecture

A typical configuration of SpyBlock uses a single virtual machine in which all user applications run, including the browser, word processor, mail reader, etc. We refer to this virtual machine as the *application environment*. The SpyBlock system consists of two distinct components, as shown in Figure 1.

- **Authentication agent** software runs on the host operating system. It provides a trusted path to the user for obtaining authentication credentials. Although generally stateless, it can store temporary session keys and may optionally act as a password manager. The trusted path may be implemented using one or more of the options described in Section 3.2. The authentication agent also provides an interface for transaction confirmation, described in Section 4.1.2.

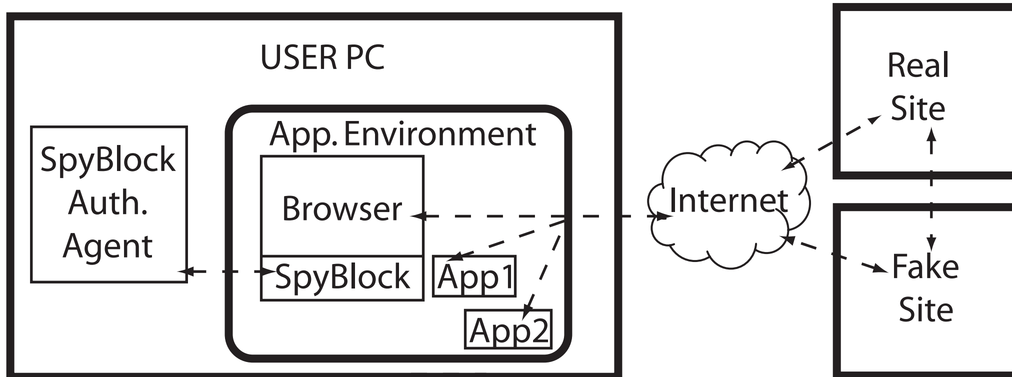

Figure 1: The components of a typical SpyBlock configuration. For most authentication modes, SpyBlock does not need to interfere with outgoing network requests.

- **Browser helper** software runs in the untrusted application environment, inside a virtual machine. It provides both the user and the remote website with options to initiate various types of authentication, but it does not actually gain access to the necessary authentication credentials to complete this authentication unless it is assisted by the authentication agent. The goal of the browser helper software is to provide the authentication agent with the information it needs to complete the authentication process, and to protect against threats that reside on the network. To accomplish this task, it must have access to the user's browser; thus, it is natural for this component to be distributed with the browser or as a browser extension.

  If the application environment itself is compromised by malware that interferes with SpyBlock or uninstalls it, the authentication agent may be unable to perform authentication, resulting in a denial of service attack. In this paper, we focus on more insidious attacks that attempt to steal the user's authentication credentials, or send unwanted requests on behalf of an authenticated user. Defenses for these attacks are described in Section 4.

In addition to these client side components, there is a possibility of server-side changes. Server support is not required to use SpyBlock, but it enables some of the optional authentication modes described in Section 4. The server can use JavaScript to test whether SpyBlock is available on the client, and which authentication modes it supports.

3.2 Authentication agent

In order to provide security against malicious software that may be running in the application environment, the authentication agent should be run in an isolated environment. In our implementation of SpyBlock, the authentication agent runs in the host operating system and the application environment runs in a guest virtual machine.

SpyBlock provides a mechanism for the authentication agent to interact directly with the user in a manner that is protected against malicious interference from the untrusted application environment. This feat is accomplished through the use of a dialog box that appears over the virtual machine window. The dialog box is "always on top" over the application environment virtual machine, so that malicious code inside application environment cannot cover the trusted password field with a popup window or steal away the user focus.

Because a platform running SpyBlock includes a mix of trusted and untrusted elements, it is essential to provide some defense against spoofed dialog boxes, so that the user knows they are not giving their secret information directly to the untrusted application environment. Thus, the authentication agent needs some way of convincing the user that it is trustworthy. There are several options that can be implemented:

- **Pre-Shared Secret** [29, 11]. One option is for the authentication agent to present the user with a secret that was made known to the user during the setup phase, but which the application environment does not have access to. This secret could take the form of an image or phrase. It does not need to be high entropy, because the user will be immediately alerted to suspicious activity if an incorrect secret is provided. Either the authentication agent or the user could choose the secret.
- **Secure Attention Sequence**. Another way that the authentication agent can prove itself to the user is to take advantage of the fact that it gets an earlier opportunity to intercept keystrokes than the application environment, which is trapped inside a virtual machine. By training the user to always press a certain key or combination of keys before entering sensitive information, the authentication service can ensure that the application environment never gains access to sensitive user information.
- **Screen Partition**. The authentication agent can also demonstrate its trustworthiness by writing to an area of the screen that the untrusted virtual machine does not have access to. This technique works when the virtual machine is confined to a clearly identified window on the screen and does not have the capability to take control over the full screen. The authentication agent dialog box could then appear off to the side of the virtual machine window, or it could fill the entire screen. [24]

The advantage of the secure attention sequence is that it is self reinforcing; the user must always use it to log in, so it becomes a habit to provide it. Checking to ensure that the pre-shared secret is displayed is not necessary to log in, so the user may be willing to log in anyway even if it is not displayed.

The advantage of the pre-shared secret and screen partition approaches is that authentication prompts can be safely triggered by the application environment. This additional control allows the designers of the untrusted operating system and possibly the remote web server to make the login process as robust and user-friendly as possible. As described in Section 5, we implemented the pre-shared secret approach with trusted images.

4 Authentication options

In SpyBlock, we implemented several authentication modes, providing different levels of security depending on the level of support available from the remote web server, as shown in Table 1.

- **Password hashing** is an anti-phishing defense that generates a unique password per site from the user's master password. By itself, it is vulnerable to keylogging.
- **Password injection** is an anti-keylogging defense that hides the user's passwords from spyware in the application environment. By itself, it is vulnerable to phishing attacks.
- Combined, **hashing and injection** provide an anti-phishing and anti-keylogging defense.
- **Strong password authentication** provides all the benefits of password hashing and password injection, and is resistant to dictionary attacks. However, it requires server-side changes.

Authentication \ Threat	Common Password	Phishing	Spyware keylogger	Network password sniffing	Network cookie sniffing	Pharming	Malware session hijacking
Password hashing	✓	✓					
Password injection			✓				
Hashing and injection	✓	✓	✓				
Strong Pwd Auth (PAKE)	✓	✓	✓	✓			
Transaction Confirmation	✓	✓	✓	✓	✓	✓	✓

Table 1: A checkmark indicates that the authentication mode achieves the security goals outlined in Section 2. Note that all of these authentication modes are secure against network password sniffing and network cookie sniffing if HTTPS is used for passwords and cookies, respectively.

- **Transaction confirmation** provides all the benefits of strong password authentication, and further protects the user from transactions generated by malware. It also requires server-side changes.

We begin our discussion with the authentication options that require server-side changes, and continue with the authentication options that can be deployed using existing web authentication.

4.1 Server-assisted authentication

If the remote web site is willing to provide server support for stronger authentication, using the SpyBlock JavaScript API (see Appendix), then the best SpyBlock defenses are available: strong password authentication and transaction confirmation.

4.1.1 Strong password authentication

Web password authentication today is most often a one way process: the client sends a password (or hashed password or one-time password) to the remote web site. Although this connection may be encrypted by HTTPS, if the remote site happens to differ from the user's intent (as in phishing), the attacker can now authenticate as the user. Far stronger password authentication is possible if one allows for a little interaction between the web site and the user's machine. These authentication protocols are generally called *Password Authenticated Key Exchange* or PAKE for short. Some examples of PAKE are EKE [6, 7, 5], SPEKE [18, 19], SRP [37], and others [8, 23, 21, 13]. Of these, SRP in particular has been widely implemented in a variety of environments including an implementation in JavaScript. PAKE protocols ensure that active attackers (such as phishing sites) learn no information about the user's password. Moreover, PAKE can be used for password-based mutual authentication and prevents offline password dictionary attacks. Using PAKE for web authentication requires both server and client side changes — both sides must participate in the PAKE protocol.

PAKE protocols by themselves are of little use in protecting users from phishing or key-logging. A key-logger can easily record user passwords as they are keyed in. Similarly, a phisher can fool the user into entering the password in a non-password field [31]. Both phishers and key-loggers thus obtain cleartext user passwords, bypassing all the protection provided by the PAKE protocol.

SpyBlock, on the other hand, provides the perfect environment for securely using PAKE protocols to ensure that users cannot inadvertently divulge their passwords. In this section, we explain how PAKE is used in SpyBlock to ensure that neither spyware nor a phishing site can

obtain any information about the user's password.

When using PAKE authentication, the SpyBlock authentication agent sits "behind" the application environment and never communicates with the network directly (as shown in Figure 1). At a high level, PAKE authentication in SpyBlock proceeds as follows:

1. **Login page.** Using the browser in the application environment, the user navigates to a login page on a remote web site. This page could be completely under the control of spyware, or it could be controlled by an online phishing site playing man-in-the-middle. The page contains (untrusted) JavaScript that detects that SpyBlock is installed and supports PAKE.

2. **Connection setup.** When the user is ready to log in, the login page uses JavaScript to communicate the URL of the website's authentication server to the SpyBlock browser helper.

3. **PAKE protocol.** When the authentication agent is first contacted by the application environment, the authentication agent allows the user to provide a password by presenting a dialog box. Using one of the techniques discussed in Section 3.2, the user can be confident that this dialog box is from the authentication agent. The dialog displays the hostname taken from the URL of the remote authentication server, provided by the application environment. The authentication agent and web site then engage in the PAKE protocol, sending messages back and forth through the application environment. This protocol reveals nothing about the password to the application environment or to any man-in-the-middle phishing site.

4. **Server HTTPS confirmation.** The PAKE protocol, which in our case runs over an HTTPS session, produces a shared secret K. Now, we bind the HTTPS session to the negotiated PAKE secret key. The reason for this step is explained below. We define the HTTPS context, denoted by S, to include the authentication server's certificate for that session, and optionally the HTTPS master secret. The server computes $T = \mathrm{HMAC}_K(S \,||\, 0)$ and sends T to the client, which normally is the browser helper. The browser helper builds its own version of the HTTPS context S' (which normally would be the same as S) and sends S' and T to the authentication agent. The authentication agent checks that T is a valid MAC for $S' \,||\, 0$. If the MAC check fails, the authentication agent warns the user.

5. **Client HTTPS confirmation.** The authentication agent computes $T' = \mathrm{HMAC}_K(S' \,||\, 1)$ and sends T' to the remote authentication server through the browser helper. The remote authentication server similarly verifies that T' is a valid MAC for $S \,||\, 1$. If the MAC check fails, the server rejects the login.

6. **Session Cookie.** If the client authentication step succeeded, the server provides the SpyBlock browser helper with a short-lived authentication token that can be used for future requests. This token can be stored into a secure HTTPS-only cookie or a regular cookie. The server should use a HTTPS cookie if protection against network cookie sniffing is desired.

To reiterate, PAKE ensures that neither the application environment nor a phishing site learn any useful information about the password. SpyBlock ensures that no information about the user's password leaves the authentication agent, and thus the user's password is inaccessible to spyware.

As described in Section 3.1, the isolated authentication agent protects the password from the application environment, but it must cooperate with the browser helper in the application environment to protect the network session. The client and server HTTPS confirmation steps (steps 4 and 5) are intended to prevent an online phishing site from hijacking the HTTPS session once PAKE authentication completes. Suppose the application environment is not corrupt by malware, but the

user fell victim to a phishing attack. The phishing site plays an online man-in-the-middle (it has an open HTTPS session with the application environment and another open session with the web site). Under this threat model (online phishing, but no spyware) the phishing site can hijack the session as soon as the PAKE protocol completes. The HTTPS confirmation steps defeat this attack since the certificate used by the application environment will be the phishing site's certificate, which should be different from the legitimate web site's certificate. Both the user and the legitimate remote web server will thus be warned if an online phishing attack occurs.

Of course, if the application environment is corrupt, HTTPS confirmation does not protect against an active attacker since the corrupt browser helper can provide a false HTTPS context. We discuss how to prevent session hijacking by spyware in the next section.

4.1.2 Transaction confirmation

In previous sections we explained how SpyBlock prevents malware running in the application environment from stealing user passwords. We also discussed how SpyBlock uses strong password authentication (PAKE) to protect users from an offline dictionary attack by a phishing site. Unfortunately, strong authentication is insufficient for fully protecting users from malware. Recall that once the authentication completes, web sites typically issue a session cookie used to authenticate subsequent messages from the browser. These session cookies reside in the application environment and are fully accessible to malware. Malware can thus wait for the user to securely login to a banking site and then issue money transfer requests using the session cookie. Similar attacks apply even if the authentication is repeated for every request. Some such malicious transaction generators have already been found in the wild, especially for clicking on Google ads. *Malware session hijacking* targeting financial and e-commerce sites is imminent, especially as stronger user authentication methods are deployed.

In this section we discuss the component of SpyBlock that ensures that every web transaction from the user's machine is issued by the user. Needless to say, session hijacking malware is extremely difficult to block. The generated transactions appear to be coming from the user — neither the authentication agent nor the remote web site can tell the difference.

Our approach to blocking malware session hijacking is to ask the user to confirm transactions issued by the user's machine. SpyBlock provides an API that allows websites to customize the transaction details and initiate confirmations, so this process does not have to be burdensome on the user. In fact, with the exception of Amazon's patented One-Click purchasing method [2], the vast majority of e-commerce and financial websites already provide users with some form of transaction confirmation. SpyBlock moves this process into the trusted authentication agent, using the following steps:

1. **Determine if confirmation is necessary**. When the user attempts to perform an action using the remote website, the server checks whether the transaction is such that user confirmation is required (the decision could be made, for example, based on transaction value or shipping address).
2. **Initiation**. If confirmation is required, the website runs a JavaScript that passes the transaction details D (including layout information) to the browser helper in the application environment. The untrusted browser helper notes the context of the site making the request and passes this information along with D to the authentication agent.
3. **Verification by user**. The authentication agent displays transaction details D to the user in a trusted window (similar to the password entry window) and waits for the user to click the confirmation button. The user can abort the transaction, or proceed.

4. **Computing MAC**. If no cached PAKE key was previously negotiated with the submission site, then the authentication agent asks for the user's password for the site and runs a PAKE protocol with the submission site to generate K. As in the previous section, protocol messages are relayed through the application environment.

   Let K be the PAKE key negotiated with the submission site. The authentication agent computes a MAC, $T \leftarrow \text{HMAC}_K(\ D\ )$.

5. **Response to server**. The authentication agent returns T to the JavaScript running in the application environment. The JavaScript includes T in the transaction submission message.

6. **Verification by server**. The submission site validates T using the PAKE key K and rejects the transaction if T is invalid.

The script on the transaction confirmation page is running in an environment that is fully under the control of malware. Clearly the malware can cause denial-of-service. But it cannot issue transactions on its own — it has no access to the PAKE key K and thus cannot build MACs itself. We assume that D contains a unique transaction ID so that malware in the application environment cannot use the same T to cause user transactions (such as stock purchases) to be issued multiple times.

For backwards compatibility, web sites must support browsers that do not support SpyBlock. This seems to suggest that malware in the application environment can simply uninstall the SpyBlock extension and thus disable the transaction confirmation mechanism. Fortunately, traditional fraud detection methods will defeat this: if SpyBlock was available for a previous transaction and suddenly becomes unavailable, the site can default to other fraud detection methods (such as calling the user to verify credit card use).

The JavaScript approach to implementing confirmation policy and sending transaction data to the authentication agent is described in the Appendix. It enables web sites to implement dynamic policies (perhaps dependent on other fraud detection signals) and to easily specify what transaction data should be displayed to the user.

Alternate designs. User confirmation was previously used before to defend against transaction generators. Some proposals use smartcards to display transaction details to the user [33] while others redirect the user to a confirmation web site [34]. SpyBlock, in contrast, requires no additional hardware or additional network infrastructure.

If the web site insists on using plain password authentication (as opposed to PAKE), then the authentication agent could use the user's password to directly to derive K and compute T. However, T exposes the user's password to a dictionary attack.

We also note that the MAC T can be replaced with a digital signature using a pre-registered public/private key pair. Digital signatures complicate the registration process and in many cases only provides marginal benefit to the site. Nevertheless, SpyBlock can be easily extended to support non-repudiation for transaction confirmation when needed.

SpyBlock is primarily concerned with protecting the secrecy of the password and the integrity of transactions. However, the same mechanism (using the PAKE key) could be used to send private information in the other direction — from the web site to the authentication agent, where it is displayed to the user. The information is routed through the application environment, but malware cannot view it. This ability to send private messages from the web site to the user could be used, for example, for displaying a purchase order number or a credit card number to the user.

4.2 Unassisted authentication

Although better authentication protocols can dramatically improve web security, significant server changes are required. To provide better web security *now*, SpyBlock also supports unassisted authentication. Password hashing provides protection against phishing, while password injection provides protection against keyloggers and hosts file hijackers. Used together, these techniques counter the most common web threats in a manner that is invisible to the server. As shown in Table 1, password hashing and injection do not protect against all possible threats, but they represent a dramatic improvement over existing web authentication.

4.2.1 Password hashing

Password hashing is an old idea [1, 12] and was recently studied in the context of a web browser [31]. For completeness, we sketch the main ideas here.

To initiate password hashing, the application environment provides the domain name of the site where the password field appears, and the authentication agent obtains a password for that domain from the user. The authentication service computes the hash of that password using a Pseudo Random Function (PRF) [14]:

$$hash(pwd, dom) = PRF_{pwd}(dom).$$

The hash is sent back to the application environment, which is supposed to insert it into a password field on a page belonging to the provided domain.

Password hashing requires the user to set up a hashed password at the server during the initial signup process or using the website's password reset page [31]. However, the server can treat the hashed password as an ordinary password. Additional techniques, such as appending user-specific salt to the password and using a slow hash function [30, 16], can provide additional protection against offline dictionary attacks.

If the application environment is well-behaved, then password hashing provides protection against phishing. As long as the phishing page is using a different domain from the site it is trying to spoof, it does not receive a hash that can be used to log in at the target site. Password hashing also provides protection against the common password problem: if the user's account at a low-security site is compromised, the attacker does not automatically gain access to any other sites where the user has the same password.

If the application environment is malicious (for example, if a hosts file hijacking has occurred), then the user's authentication credential at the domain they are logging in to can be compromised. However, the user's authentication credential at other sites are not compromised (until the user attempts to log in to them), even if the user has the same master password at those sites. Thus, password hashing provides a limited defense against the common password problem with respect to spyware. If the user frequently resets the application environment back to a "known good" state, then password hashing may be used to limit the damage of a breach — the attacker only gains access to those sites that are accessed during the limited time it has control over the compromised virtual machine.

4.2.2 Password injection

Password injection is designed to protect passwords from passive spyware. By itself, it provides no protection against phishing, as indicated in Table 1. In the basic password injection scheme, the SpyBlock application environment does not send the user's password to the remote server di-

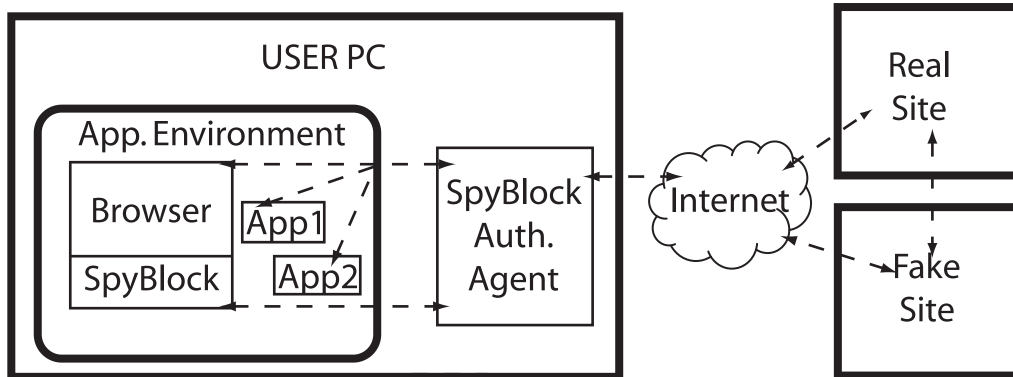

Figure 2: For password injection, SpyBlock must modify the outbound network requests.

rectly. Instead, as shown in Figure 2, it contacts the authentication agent, which obtains the password from the user, inserts it into the POST request and forwards it on to the remote server. When the response comes back, the authentication agent forwards it back to the application environment so that it may be authenticated for the current session.

The SpyBlock browser helper detects when the user is filling out a form and helps the authentication agent intercept the request and inject the password into the correct place. For this scheme to be considered successful, authentication should succeed if all parties are honest, and the password should not be readable by the application environment if the application environment is malicious and the remote server is honest. (The user decides whether to trust the server based on its hostname.) Used in conjunction with password hashing, we have the additional goal that the server cannot gain access to the user's master password, even if it cooperates maliciously with a compromised application environment.

Passwords are often encrypted in transit, but because SpyBlock has components running in both the application environment and the authentication agent, it could assist the authentication agent in modifying the encrypted request via either of these two methods:

- **Man-in-the-middle HTTPS Proxy**. If the authentication agent is configured as a proxy server for the application environment, then it can modify the raw HTTP outgoing request, replacing a placeholder value with the user's password. For standard HTTP requests, the authentication agent can simply act as an HTTP proxy server. If HTTPS is used, it is necessary for the application environment to voluntarily submit to a man-in-the-middle attack on HTTPS by the authentication agent, so that the password can be injected. The authentication agent acts as a certificate authority, presenting a self-signed certificate to the application environment authenticating itself with the same HTTPS certificate details that the remote server is using. The application environment, which has been pre-programmed to accept this CA certificate, will interact normally with the server. Under normal circumstances, this configuration will not present the user with any warnings.
- **Cleartext HTTPS Proxy**. Because we have control over the application environment inside the virtual machine, rather than treating it as a "black box" we may wish to reconfigure the way the browser handles HTTPS and permit the data to be sent to the authentication agent

(which generally resides on the same machine) in cleartext. The authentication agent then assumes responsibility for setting up a separate HTTPS connection for the authentication and displaying warnings to the user if there is a problem. We chose this technique for SpyBlock.

A naive implementation of password injection might assume that the password sent to the server would never be sent back to the untrusted machine. However, in practice servers often echo back information that is sent to them. Data in HTML forms is sent to the server as a series of `key=value` pairs, with no meta information about the type of data being transmitted. Without knowing any information about the structure of the site, it is not possible for the authentication agent to determine just from the HTTP request whether the server will treat a particular field as sensitive data or unwittingly assist the attacker in reflecting the password back in plaintext.

In order to prevent these types of attacks, we need some way of determining whether the server will treat the password as private data. A simple assumption based on well-known best practices is that the site hosting the form containing a password field will ensure that the contents of that password field will not be reflected back at the user. Using this assumption, the authentication agent can prevent reflection attacks by interposing itself on two HTTP requests: the one that received the password form field, and the POST request that actually contains the user's password data.

The password injection method used in SpyBlock uses the following multi-stage approach:

- **Caching Server Response**. With the help of the browser helper in the application environment, the authentication agent intercepts outgoing browser HTTP requests and parses incoming HTTP responses that contain password fields, caching the `name` attribute of the password field and the `action` attribute of the form that contains each password field. If password hashing is used, the URL of the outgoing request will be used for the domain of the hash.
- **Verification**. When the browser helper in the application environment makes a request to the authentication agent for password injection, the authentication agent checks to ensure that the field where the password is being inserted matches one of the intercepted password fields in the cache.
- **Injection**. The authentication agent then modifies the outbound `POST` data, setting the form variable corresponding to the password field to the desired password.

4.2.3 Password hashing and injection

Password hashing and injection can be combined to give phishing and spyware protection without server-side changes. In Section 4.2.2 we assumed that the remote site was explicitly trusted by the user, so reflection attacks would have to originate from spyware. However, if the remote site is not trusted, we must now consider the possibility that a phishing page might attempt to initiate a reflection attack.

An example of such an attack is a phishing page that contains a form with a password field. When the form is submitted, it posts the hashed password as a comment on a public forum, where it can be read by an attacker. Password hashing and injection defends against this attack, as long as the site name used to hash the password belongs to the attacker and not the forum site. The following pitfalls should be avoided:

- **Using the form `action` domain for hashing**. It is tempting to hash using the domain of the site where the password is being sent. However, if the site where the form data is sent differs

from the site where the form appears, this approach is unsafe. The location where the form appears could be controlled by an attacker, and it could launch a reflection attack, posting into a form variable that the remote server does not consider to be a password field. Thus, the domain of the site where the password field appears is the domain that should be used for hashing.

- **Using the HTTPS certificate**. Since the password is usually sent over HTTPS, it is tempting to hash using the common name of the certificate or some other certificate details, rather than using the domain of the site to perform the hashing. However, many major websites, such as American Express [3] and Hotmail [17], do not protect the page where the password field appears with HTTPS. Only the page where the form data is sent is protected. Because the page where the password field appears is the one that should be used for hashing, the HTTPS certificate is not generally usable for hashing.

- **User submitted HTML content**. Some sites, such as blogs and wikis, allow users to contribute arbitrary data such as password fields and forms that may have an `action` attribute pointing to a different domain. These sites are not protected by the reflection defense described in this section. For security reasons, high security sites such as banks generally do not allow normal users to upload such content. Sites that require users to be able to upload arbitrary HTML content as part of their core functionality can use server-assisted authentication to eliminate these issues.

- **Unusual web authentication techniques**. Until recently, there was one major website (Yahoo) that computed a client-side hash of the password using JavaScript, which was sent unencrypted over HTTP. This technique was presumably designed to reduce the server workload of processing HTTPS connections. Password injection would not work in this case, because the injected password will not be hashed. However, Yahoo now uses standard password authentication over HTTPS, and we have not encountered any other major sites that use this technique.

5 Implementation

We implemented SpyBlock as a browser extension for Mozilla Firefox. For simplicity, it is distributed as a single installer that is run on each participating machine. During the initial setup step, the extension asks the user which authentication modes are allow, and which role (authentication agent, application environment) it should perform. These roles communicate with each other using local network sockets.

Distribution as a browser extension is convenient because it allows a single cross-platform installer that can work with any virtual machine. Because SpyBlock is running as a browser extension rather than a separate user-level application, it can easily communicate with the remote server through a simple JavaScript API (see Appendix). By checking if the global `window.spyblock` is defined, the server can allow graceful degradation in case SpyBlock is not available. By querying the `spyblock` object for the authentication options it supports, the remote server can negotiate an authentication protocol that is acceptable to both the client and the server.

We implemented the pre-shared secret approach described in Section 3.2, allowing the user to pick a trusted image during the setup of the authentication agent. The primary reasons for our choice of approach were ease of use and ease of implementation. An example of an authentication agent trusted dialog box is shown in Figure 3. The dialog box shows the type of authentication that will be used, the server where it will be sent, and the trusted image that was chosen by the user. The transaction confirmation dialog box additionally shows the text that the site wants to MAC. To

Figure 3: The SpyBlock password dialog shows the trusted image and offers optional password management.

prevent a malicious site from confusing the user, the text is confined to a clearly delimited area and only a small subset of HTML tags are allowed.

In case a user is concerned about accidentally choosing an inappropriate authentication mode (for example, password injection with no hashing in a phishing situation), SpyBlock explains the relative merits of each authentication mode and allows modes to be selectively disabled.

6 Related Work

Several previous projects have developed client-side anti-phishing methods that are vulnerable to spyware attacks. For example, SpoofGuard [9] is a browser extension that attempts to detect spoof sites and prevent users from posting passwords to them. PwdHash [31] is another browser extension that produces site-specific passwords using a cryptographic hash, implemented in combination with countermeasures against malicious JavaScript. Password Multiplier [16] uses a slow hashing function and caches an intermediate result, in order to discourage dictionary attacks on the hashed password. Other password hashing implementations include PasswordMaker [20], Password Composer [22], Passwdlet [35], Genpass [39]. Client-side measures that process passwords in an untrusted environment are inherently vulnerable to spyware running on the client platform.

Passmark's [29] custom images and dynamic security skins [11] combat phishing by providing visual web site authentication to users. These methods can combine information on the client with server enhancements that generate or contribute to user-specific images. While server activity is protected from client-side malware, client-side keyloggers can capture web passwords as well as information related to the authentication images.

Previous efforts to provide stronger web authentication, in a form that is resistant to keyloggers and other client-side malware, use additional hardware. These include the "phoolproof" [28] cell-phone-based authentication protocol and the PwdCell [26] Bluetooth-based cell phone pass-

word hashing method. Using a combination of temporary keypairs, proxy certificates, and wireless protocols, PorKI [32] supports the use of PKI credentials on Bluetooth-enabled workstations. In addition to the hardware requirements, these approaches require the user to interact with an additional device, interrupting the screen-keyboard-mouse orientation of most computer users. Since these methods rely on server protocol changes, none can be deployed today to protect against spyware when accessing conventional websites.

The SpyBlock use of virtualization for isolating a trusted component from the spyware-vulnerable environment is related to some other virtualization and visualization-like efforts. In particular, NSA's NetTop [25] architecture uses VMware to isolate processes at different security levels running on the same physical hardware. Some alternatives are the GreenBorder [15] isolation methods, and Featherweight Virtual Machines [38].

Another system that uses an isolated authentication agent to manage secrets on behalf of the user is Plan 9's "factotum" agent [10]. The project focused on the minimalistic, reliable, protocol-agnostic nature of the agent. Our emphasis is different — we provide specific protocols to protect against known web threats, and tackle the hard problem of user interface spoofing attacks.

7 Conclusion

The SpyBlock architecture and implementation protect web users by isolating spyware in the untrusted application environment from a trusted authentication agent in a trusted environment. Using a virtual machine monitor, the trusted and untrusted components can both run on the same physical machine. Alternatively, a trusted operating system could provide sufficient isolation through interprocess separation, or the trusted and untrusted environments could be installed on communicating hardware.

A crucial element of SpyBlock is its use of a trusted user interface that assures the user through a pre-shared secret such as a chosen image, through requiring a secure attention sequence, or by using an unspoofable screen partition. In combination with the browser extension, the authentication agent implements several password authentication options and provides protection against a range of threats. In particular, password hashing and our novel password injection mechanism together provide protection against password phishing, the common password problem, and keyloggers, without requiring server-side support. Thus, SpyBlock is a marked improvement over previous antiphishing and related technology that works with current, existing web password interfaces and protocols.

SpyBlock can provide significantly stronger protection, also resisting network-based password sniffing, pharming, session hijacking, and theft of session cookies, if web servers are configured to support stronger authentication protocols. We explore this option by implementing a strong password-authenticated key exchange protocol, a novel mechanism combining the exchanged key with SSL/TLS authentication material, and transaction confirmation. Perhaps one of the more interesting insights of this study is the need for mechanisms that provide some form of session integrity. Specifically, strong authentication alone does not prevent cookie theft and does not prevent spyware from generating malicious transactions after authentication. Our authenticated transaction confirmation in the authentication agent prevents fraudulent spyware-generated transactions from being completed by the server. While these SpyBlock features rely on significant changes in web server operation, it is interesting to see how secure web browsing can be achieved. In addition, these measures are entirely realistic for high-security sites or applications since no changes in other parts of the network infrastructure are required.

SpyBlock is available for free download (URL omitted for anonymity).

References

[1] M. Abadi, L. Bharat, and A. Marais. System and method for generating unique passwords. US Patent 6,141,760, 1997.

[2] Amazon. `http://www.amazon.com/`.

[3] America Express. `http://www.americanexpress.com/`.

[4] Anti-phishing working group. `http://www.antiphishing.org`.

[5] M. Bellare, D. Pointcheva, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Proceedings of Eurocrypt 2000*, 2000.

[6] S. Bellovin and M Merritt. Encrypted key exchange: password based protocols secure against dictionary attacks. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, 1992.

[7] S Bellovin and M. Merritt. Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. In *1st ACM Conf. on Computer and Communications Security*, pages 244–250, 1993.

[8] V. Boyko, P. MacKenzie, and S. Patel. Provably-secure password-authenticated key exchange using diffie-hellman. In *Proc. of Eurocrypt 2000*, volume 1807 of *LNCS*, pages 156–171. Springer-Verlag, 2000.

[9] N. Chou, R. Ledesma, Y. Teraguchi, and J. Mitchell. Client-side defense against web-based identity theft. In *Proceedings of Network and Distributed Systems Security (NDSS)*, 2004.

[10] Russ Cox, Eric Grosse, Rob Pike, Dave Presotto, and Sean Quinlan. Security in plan 9. In *Proceedings of the 11th USENIX Security Symposium*, San Francisco, 2002.

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

[12] E. Gaber, P. Gobbons, Y. Mattias, and A. Mayer. How to make personalized web browsing simple, secure, and anonymous. In *Proceedings of Financial Crypto '97*, volume 1318 of *LNCS*. Springer-Verlag, 1997.

[13] R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In *Proc. of Eurocrypt 2003*, volume 2656 of *LNCS*, pages 524–543. Springer-Verlag, 2003.

[14] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[15] GreenBorder. `http://www.greenborder.com/`.

[16] J. A. Halderman, B. Waters, and E. Felten. A convenient method for securely managing passwords. Proceedings of the 14th International World Wide Web Conference (WWW 2005), 2005.

[17] Hotmail. `http://www.hotmail.com/`.

[18] D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communications Review*, 26(5):5–20, 1996.

[19] D. Jablon. Extended password key exchange protocols immune to dictionary attack. In *Proc. of WET-ICE '97*, pages 248–255. IEEE, 1997.

[20] E. Jung. Passwordmaker. `http://passwordmaker.mozdev.org`.

[21] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Proc. of Eurocrypt 2001*, volume 2045 of *LNCS*, pages 475–494. Springer-Verlag, 2001.

[22] J. la Poutré. Password composer. `http://www.xs4all.nl/~jlpoutre/BoT/Javascript/PasswordComposer/`.

[23] P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on rsa. In *Proc. of Asiacrypt 2000*, volume 1976 of *LNCS*, pages 599–613. Springer-Verlag, 2000.

[24] Davor Matic. Xnest. http://www.xfree86.org/4.2.0/Xnest.1.html.

[25] R. Meushaw and D. Simard. Nettop: Commercial technology in high assurance applications. *Tech Trend Notes, National Security Agency*, 9, 2000.

[26] Nicholas Miyake. Bluetooth password hashing. `http://www.stanford.edu/~nfm02/pwdcell/`.

[27] A. Moshchuk, T. Bragin, S. Gribble, and H. Levy. A crawler-based study of spyware on the web. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS 2006)*, February 2006.

[28] Bryan Parno, Cynthia Kuo, and Adrian Perrig. Authentication and fraud detection: Phoolproof phishing prevention. In *Proceedings of Financial Cryptography and Data Security (FC '06)*, 2006.

[29] Passmark. `http://www.passmarksecurity.com`.

[30] N. Provos and D. Mazières. A future-adaptable password scheme (the electronic version). In *USENIX '99, Freenix Track*, June 1999. From `http://www.usenix.org/events/usenix99/provos.html`.

[31] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. Mitchell. Stronger password authentication using browser extensions. In *Proceedings of the 14th Usenix Security Symposium*, 2005.

[32] S. Sinclair and S.W. Smith. PorKI: Making user PKI safe on machines of heterogeneous trustworthiness. In *Proceedings of the 21st Annual Computer Security Applications Conference*. IEEE Computer Society, 2005.

[33] David Steeves. Securing online transactions with a trusted digital identity. `http://crypto.stanford.edu/TIPPI/slides/steves.ppt`, 2005.

[34] Verified by Visa. `https://usa.visa.com/personal/security/vbv/`.

[35] N. Wolff. Password generator bookmarklet. `http://angel.net/~nic/passwdlet.html`.

[36] Min Wu. Users are not dependable - how to make security indicators to better protect them. In *Trustworthy Interfaces for Passwords and Personal Information (TIPPI) Workshop*, 2005. Invited talk. Abstract at `http://crypto.stanford.edu/TIPPI/wu.html`.

[37] T. Wu. The secure remote password protocol. In *Proc. Internet Society Symp. on Network and Distributed System Security (NDSS)*, pages 97–111, 1998.

[38] Yang Yu, Susanta Nanda, Lap-chung Lam, Fanglu Guo, and Tzi-cker Chiueh. Feather-weight virtual machine and its applications. Technical Report. http://www.ecsl.cs.sunysb.edu/fvm/, March 2005.

[39] C. Zarate. Genpass. http://labs.zarate.org/passwd/.

Appendix: SpyBlock JavaScript API

A web server can initiate communication with the authentication agent by embedding the following JavaScript snippets in a standard HTML page using the `<script>` tag. The JavaScript request is passed through the application environment to the authentication agent.

Password hashing The script begins by checking whether the global `window.spyblock` is defined, which should only be true if SpyBlock is installed. If SpyBlock reports that it is configured to allow password hashing, the server can initiate hashing by using the `usePwdField` method. The server must indicate a password field to insert the hashed password into. Password hashing can also be initiated by the user using the password key or a context menu selection.

```
if(window.spyblock && spyblock.supports('hashing'))
  spyblock.usePwdField('hashing', document.form1.pwdfield);
```

Password injection For password injection, the script provides a password field. When the form that the password field belongs to is submitted, the injection occurs. Password injection can also be initiated by the user.

```
if(window.spyblock && spyblock.supports('injection'))
  spyblock.usePwdField('injection', document.form1.pwdfield);
```

To perform both hashing and injection, the script passes in `'hashing,injection'` as the first argument of `usePwdField`.

Strong password authentication Strong password authentication requires asynchronous communication with the server. A JavaScript observer object is used to receive notification when the authentication is completed. The object must implement one method, `observe`. We omit error handling code here, so no arguments are used.

To trigger the authentication protocol, the script calls `useDocument` on an HTML document node (possibly hidden inside an inline frame) that will be used to communicate with the server through a series of HTTP GET requests. When the protocol is finished, the observer's `observe` method is called, and the session authenticator can be read. In this case, we saved it into the document cookie so that it can be used as an authenticator for subsequent requests.

```
var save = function(token) { document.cookie = token; };
var observer = { observe: function() { save(doc.body.innerHTML); } };
if(window.spyblock && spyblock.supports('srp'))
  spyblock.useDocument('srp', doc, observer);
```

Transaction confirmation Transaction confirmation also requires asynchronous communication with the server. Transaction confirmation can be used to MAC the contents of a (possibly hidden) inline frame. As before, a JavaScript observer object is used to receive notification when the protocol is complete. The observer obtains the MAC of the transaction data, and in this example the script puts it into a hidden input field of a standard web form.

```
var save = function(mac) { document.form1.hiddenfield.value = mac; };
var observer = { observe: function() { save(doc.body.innerHTML); } };
if(window.spyblock && spyblock.supports('srp,conf'))
  spyblock.useDocument('srp,conf', iframe.contentDocument, observer);
```