Browsers Defenses Against Phishing, Spoofing and Malware

08/25/06

Amir Herzberg
Dept. of Computer Science
Bar Ilan University

Abstract

Web users are increasingly victims of phishing, spoofing and malware attacks. In this article, we discuss existing and proposed defense mechanisms. We highlight the vulnerabilities of current defenses, and the challenges of validating and adopting new defenses.

1 SSL-based Logon

Most web browsers and servers support the Secure Socket Layer (SSL) protocol (or its standard version, the Transaction Layer Security (TLS) standard); see [R00]. SSL (and TLS) are advanced, public-key cryptographic protocols. Their main goal is it to protect the confidentiality of sensitive traffic against an eavesdropper, who can listen to the traffic between the client and the server. For example, merchant sites and login pages use SSL to protect, respectively, credit card numbers and passwords, sent by users to the servers.

1.1 Simplified description of SSL as used in most sites.

SSL operation is divided into two phases: a handshake phase and a data transfer phase. We illustrate this in Figure 2, for connection between a client and an imaginary bank site (*http://www.bank.com*).

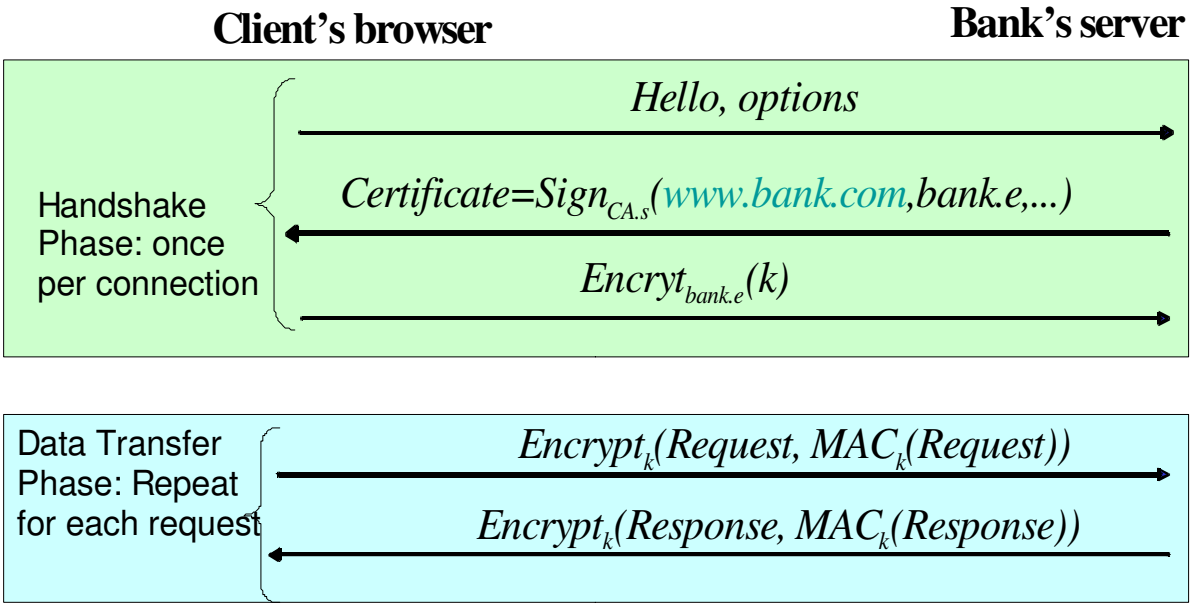

Illustration 1: Simplified flows of the SSL protocol

During the handshake phase, the browser confirms that the server has a *domain name public key certificate* issued by a *trusted Certificate Authority (CA)*. A domain name public key certificate

contains a domain name, e.g. *www.bank.com,* a public encryption key for the domain (bank), e.g. *bank.e,* and a *digital signature* over them: $Sign_{CA.s}$(*www.bank.com, bank.e).* The CA computes the signature using a *digital signature algorithm*, e.g. RSA or DSA, with the secret signing key *CA.s,* chosen by and known only to the CA. The bank receives the certificate, in advance, from the CA.

The browser validates that the domain in the certificate is the desired domain, i.e. *www.bank.com.* It then validates the signature over the certificate, by applying the *signature validation function* to the received certificate, using the *public validation key CA.v* of the CA, installed in the browser in advance. If the certificate has a valid signature, then the browser selects a random key *k*, and send it, encrypted, to the server. The browser encrypts the key *k* using a *public-key cryptosystem* such as RSA, with the public encryption key of the bank *bank.e* from the certificate, i.e. it sends $Encrypt_{bank.e}(k)$ to the bank. This completes the (simplified description of) SSL handshake phase, executed at the beginning of every SSL connection.

The data transfer phase of SSL uses the key *k,* sent encrypted from browser to server during handshake phase, to authenticate and encrypt requests (sent from browser to server) and responses (from server to browser). Again simplifying, the browser computes $Encrypt_k(Request, MAC_k(Request))$ for each *Request*, and the server computes $Encrypt_k(Response, MAC_k(Response))$ for each *Response,* where *Encrypt* is a shared-key cryptosystem, such as *AES,* and *MAC* is a shared-key *Message Authentication Code (MAC),* such as *HMAC*. This protects the confidentiality and integrity of requests and responses.

The handshake phase, as described, identifies the server (by its certificate and domain name), but not the client. SSL includes an optional mechanism for client authentication, using a certificate of the client, but this is rarely used. Instead, users usually identify themselves to the site, by presenting their name and a password. This sensitive data is transferred using SSL, during the data transfer phase, encrypted with the key *k* shared at the handshake phase.

1.2 SSL Certificate Validation

Currently, each certificate authority (CA) can decide on its own process for validating the identities of subjects, upon issuing SSL certificates. Gradually, due to competitive pressures, certificate authorities optimized and automated the validation process, sometimes compromising on the level of identity validation.

In the extreme, some certificate authorities offer low cost certificates by simple, automated validation of domain name ownership, e.g. by e-mail challenge-response. This validates only the domain, and does not provide any validation for the actual identity of the organization (beyond the domain name). As a result, such certificates are usually referred to as *domain-validated.*

Unfortunately, current browsers do not indicate to the user when a site uses a domain-validated certificate. Furthermore, even for `full` certificates, where the identity is validated (not just the domain), each authority may decide on its own process, again motivating weak (inexpensive and quick) validation process. To conclude, the current SSL certificate validation is vulnerable, and may allow rogue sites to obtain certificates for misleading identifiers (domain names and possibly even organization names).

Extended validation certificates. There is an ongoing effort by several certificate authorities and browser vendors, to define a new class of certificates, with strong, uniform identity validation process, to be defined. Hopefully, the validation process and the certifying authorities issuing such *extended validation certificates,* will be trustworthy, and this will prevent attackers from obtaining such certificate for a domain with misleading identifiers. Browsers should indicate to users the use of extended validation certificate.

There are some concerns about the viability of extended validation certificates. When will the

validation process be defined (it is already delayed)? How `painful' would it be? Would corporations pay for the extended validation certificates? Would vendors indicate such certificates to the users (and how)? Would user notice?

Certificate registry services. We now briefly present an alternative method to improve the validation of certificates, which may avoid the costs of the extended validation certificates. The idea is to establish one or more *certificate registries*. Such registries will be fully automated and very low cost; essentially, all they do is to openly list *all* SSL certificates.

To prevent impersonation attempts, corporations will monitor new additions to the certificate registries. Whenever detecting a new certificate which uses identifiers similar to their, the corporation will *protest,* i.e. contact the issuing certificate authority and demand revocation of the unauthorized certificate.

Upon receiving a new SSL certificate, the browser will query the certificate registry (chosen by the user). The browser will differentiate between recently-listed certificates (which are suspect) vs. long-listed certificates (which are unlikely to be unauthorized). Sites will register new certificates for sufficient period, before beginning to use them, or sign the new certificate (using a registered public key).

This idea requires the establishment of one (or few) certificate registries, comparable to existing trademark registries (usually provided as a government service). Establishment of a new service can be a serious obstacle; luckily, in this case, the operational expenses are very low.

Both the `extended validation` certificates, and the `certificate registry` service, attempts to improve the validity of the identifiers in certificates. However, improving certificate validity is only meaningful if users are aware of the use of certificates and of the identifiers in the certificate. In the next subsection we argue that with current browsers, users are often unaware of whether SSL is used, and of the relevant identifiers.

1.3 Usability Vulnerabilities of SSL Login in Current Browsers

The security of the SSL-based login process depends on several **correct-usage assumptions**. A failure of **any** of these assumptions can lead to exposure of the password. Here are three assumptions which are related to user behavior; unfortunately, *all three assumptions are unrealistic*:

SSL Usage Assumption 1: Users send their password only via an SSL/TLS protected connection.
Reality: Several experiments confirmed the folklore belief, that in reality, most users usually do *not* validate that SSL/TLS is active, before entering their password on a login form; see [DTH06,HG04]. This is expectable, as the SSL indicators are not very visible; most browsers indicate SSL activation only by a padlock icon in status area, and expect users to notice SSL inactive by the *lack* of the padlock icon. Furthermore, many logon forms are **not** sent to the browser over a protected (SSL/TLS) connection; most of these forms invoke SSL/TLS to protect the password, but

Illustration 2: The login portion of the homepage of Chase bank (SSL not yet invoked).

an attacker could send a look-alike page that will send the password to the attacker instead, and users will not be able to detect this. Adding to users confusion, many of these pages contain an *image* of a padlock as part of the page itself; see e.g. the unprotected logon page of Chase in Illustration 2. Such failures happen on several sensitive, widely used logon forms, e.g. of PayPal, Chase, Microsoft passport, Bank of America, and (currently) many more (see the I-NFL Hall of Shame).

SSL Usage Assumption 2: Users specify or confirm the identity of the site before entering passwords.
Reality: As described above, browsers validate that the address (URL) of the site, belongs to the domain in the certificate. However, users often do not directly specify the address of websites; instead the browser often uses address from a link in another (potentially malicious) page, or from a (potentially malicious) email message. In many browsers, sites can remove the address/location bar and possibly include a fake one to prevent users from detecting the use of wrong address. Furthermore, many users are not aware of the structure of domain names, e.g. may believe that the domain name BankOfAmerica.reo.com belongs to Bank of America; indeed, this address was in fact used legitimately for a BankOfAmerica service, without concern for the apparent domain mismatch, showing users and corporations lack of attention to domain names; see Illustration 3. Experiments confirm, that users indeed often fail to notice the use of incorrect addresses for sensitive sites [DTH06,HG04].

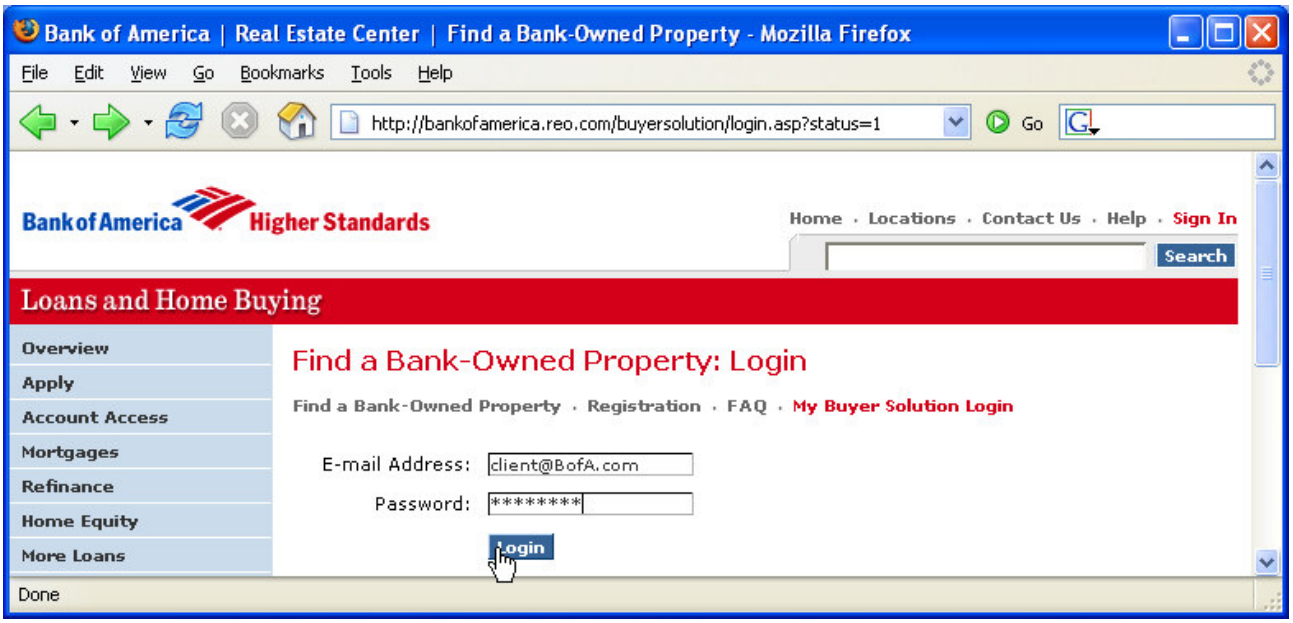

Illustration 3: Example of a corporate site (belonging to Bank of America), residing in a domain unrelated to the corporate. This demonstrates that users are not sensitive to the usage of correct domains by corporate sites.

SSL Usage Assumption 3: Users use independently, randomly chosen passwords for each service, so that each password cannot be guessed, even given passwords used by same user for other services.
Reality: Users often use weak passwords, listed in `dictionaries` of common passwords. Users often reuse the same passwords for multiple sites, and provide passwords in insecure environments such as public terminals.

In addition to the common failure of the user-behavior assumptions, there are other vulnerabilities of the current SSL-based login process. In particular, attackers have been able to obtain SSL certificates with misleading identifiers. We next discuss the certificate validation process.

2 Site Identification and Security Toolbars

As explained in section 1.2 above, with current browsers, users often fail to detect an incorrect address and/or the lack of protection (SSL). One natural remedy is to add a *toolbar,* providing site identification (e.g. name/logo) and/or other security indicators, such as an improved indicator for the usage of SSL/TLS, or an indicator for trusted vs. suspect sites.

Toolbars try to make it easier for users to identify sites and/or be aware of security status, by presenting information in a dedicated area – in addition to the (limited) information already presented by (existing) browsers. Two basic questions arise: would users notice (any) indicators provided in Toolbars? And, if so – what is the right information to present? Such questions should be tested experimentally.

Wu et al. [WMG06] compared the effectiveness of three possible contents for toolbars, as in Illustration 4, which they termed: *neutral-information toolbar* (domain, established date, country), *SSL verification toolbar* (organization and CA, or `not protected' indication), and *system-decision toolbar* (color-coding: red for suspect site, green for `good site`, yellow otherwise). The SSL-verification toolbar simulation follows TrustBar [HG04] (which we developed). However, the term `SSL verification` does not capture TrustBar's main goal of site identification. We therefore consider that the correct third category of toolbars is not SSL-validation but rather *protected-site identifying toolbars*; see discussion below, in subsection 2.3.

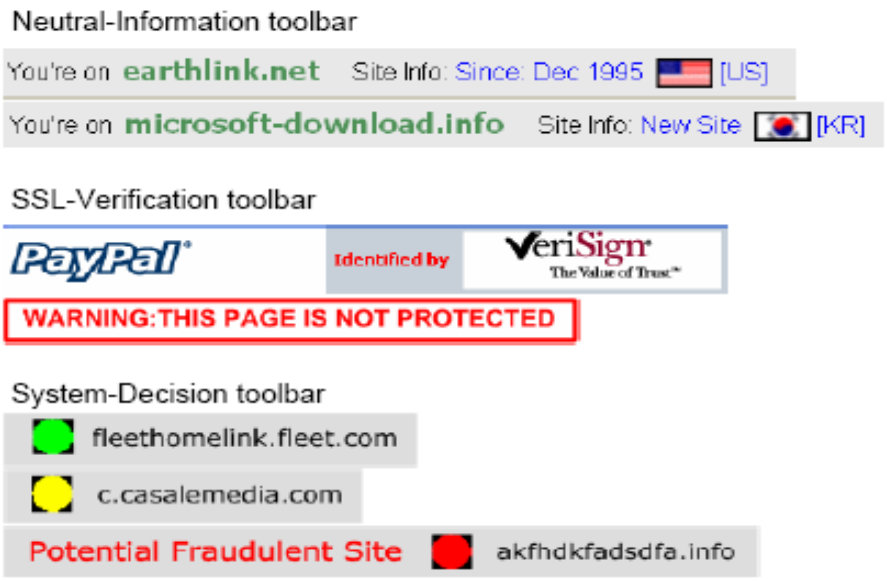

Illustration 4: The three simulated toolbars from [WMG06]

With all three simulated toolbars, [WMG06] found high *spoof rates*, which is the fraction of simulated attacks to which the participants disclosed the password.. The rates depended on the contents of the bar and the method of attack, but mostly – on *how users were introduced* to the toolbars. In a pilot study of [WMG06], one group received a printed tutorial on the indicators, and had low (7%) spoof rate. The other group did not receive a tutorial, and had 94% spoof rate. We found similar rates in a pilot study preceding [HG04].

Wu et al. concluded, that the tutorial caused users to be more alert than in normal use. To cancel this effect, in their main study, Wu et al. only sent a tutorial by email, roughly in the middle of the test. They found high spoofing rates: 52% before receiving the tutorial email, and 26% after receiving it; notice that users were still more aware of security than usual, since they were aware of the goal of the experiment. Wu et al. concluded that users ignore indicators on toolbars.

However, in our experiments, reported in [HG04], spoof rates dropped from 37% (FireFox 1.5, no toolbar) to 8% (FireFox with TrustBar, displaying <site> identified by <CA>), and even 5% (TrustBar displaying a *petname,* i.e. user-specified identifier). We believe the difference is mostly due to the fact that in [HG04], we gave a brief, one-on-one introduction to the security indicators. We believe that an email message is simply insufficient to allow users to use (any) security indicators. Therefore, we used a different strategy to avoid user's focus on security: we defined their goal as maximizing the number of selections (speed). This explains the high spoof rate of 37% using only the default FireFox security indicators (after the tutorial, which also covered the existing SSL indicators).

It therefore seems, contrary to the conclusion of [WMG06], that properly designed *security and identification toolbars can reduce spoofing*. However, additional, more realistic and long-term experiments are necessary, to measure how much of the reduction is long-lived, and not due to increased security awareness due to the experiment.

In the reminder of this section, we discuss what indicators should appear in the toolbar. In Subsection 2.1, we consider the case of web pages which are not protected (by SSL/TLS); our conclusion is that secure authentication, using SSL/TLS or other cryptographic mechanisms, is desirable. In Subsection 2.2 we discuss how to efficiently and securely authenticate web pages. Finally, in Subsection 2.3, we discuss site identification indicators (for authenticates pages), and *spoofed-indicator attacks*.

2.1 Toolbar Indicators for Unprotected Pages

Currently, most web pages, including many login pages, are *not* protected (by SSL/TLS). Users are often unaware that a login page is unprotected; one reason is that most browsers only indicate *protected* pages (by padlock, use of *https* not *http*), with only the absence of such indicator to identify unprotected pages – and users may not notice the lack of the protected-page indicators. Furthermore, [DTH06] showed that users often base their decisions of whether a site is trustworthy of spoofed, on the (easily spoofable) content of the page, including `security indicators` such as padlocks and `seals`, often placed within web pages.

Unprotected web pages are easy to spoof. Indeed, most users do not notice when these pages are spoofed, even when the pages are hosted in a different domain – a very easy attack. This motivates toolbars such as SpoofStick, simply showing the domain name more clearly, and more detailed `neutral-information` toolbars such as Netcraft's (see Illustration 4). However, the information presented by such toolbars seem to require analysis by the user for each use.

Indeed, [WMG06] found that `neutral-information` toolbar have higher spoof rate than `system decision` toolbars. System decision toolbars display an evaluation of the site: red, green or yellow `lights`, for suspected, trusted and other sites, respectively (see Illustration 4). After receiving the tutorial, the spoof rate using system decision toolbars fell to 15%.

This improvement in spoof rate is partially due to the fact that in [WMG06], the `system indicators` were emulated as completely accurate, displaying `red' warning for *all* spoofed sites (and for no other site). This may be too optimistic.

System decision toolbars detect spoofed sites either using blacklists (of spoofed sites), or using heuristic detection by the contents of the page. Currently, blacklists and heuristic detection can be fairly effective. However, their effectiveness may reduce over time, as attackers adapt. Blacklisting of IP addresses fails against attackers which use dynamic IP addresses (e.g. of Zombies, i.e. maliciously-controlled computers of innocent users). Blacklisting of domain names is even easier to circumvent, since the cost of changing domain names is small. URL blacklisting is even less effective, since an attacker can have unlimited number of URL addresses within the same domain.

A reliable `green light` indicator of trustworthy sites could be valuable. Currently, `green light`

indicators are usually based on `white-list` of trusted sites. However, a `green light` indicator must be secure against attacks, or it could lead to false sense of security, and to loss of trust when its failures are published. Unfortunately, existing system-indication toolbars are not secure enough, and are usually displayed, for unprotected sites, based on domain name and/or IP address. Such indicators are therefore vulnerable to Man in the Middle (MITM) attacks as discussed above. The (more common) domain-name white-lists also depend on the trustworthiness of information in the Domain Name System (DNS), which is often controlled by attackers (e.g. using `DNS-poisoning` attacks). This problem also exists for `neutral-information` toolbars. A `green light` indicator should only be used for sites securely identified as trustworthy, e.g. using SSL/TLS.

We conclude that toolbars should avoid giving users a false impression of security, when using unprotected sites. In TrustBar, we indicate unprotected sites by special icon (crossed-out padlock). In early versions of TrustBar we displayed a warning for unprotected sites (as in Illustration 4), but this caused users anxiety; a common response was, `I know my bank site is secure, so I can't trust the toolbar'.

A problem with this conclusion, is that most sites, including of financial organizations, are *not* protected (using SSL/TLS). This is mostly due to efficiency concerns. We next discuss efficient techniques to securely authenticate web pages.

2.2 Efficient Web Page Authentication

Only few sites use SSL/TLS, and sites that do, often use it only for special pages such as login. The main reason for that is the overhead of SSL/TLS. We now present two techniques that may allow efficient web page authentication, with acceptable overhead.

First, the overhead of using SSL/TLS for authentication of pages can be significantly reduced. Use of `null` encryption and efficient message authentication will significantly reduce the overhead of the data transfer phase (Illustration 1). The overhead of the SSL/TLS handshake phase is mainly due to the public key decryption at the server. SSL/TLS include a `session resumption' mechanism to avoid this overhead, by saving the shared keying material in the server, but this is not widely used, mainly due to the cost of such storage, and replicating the information among multiple web-servers (for a large site).

However, RFC 4705 [SZ*06] provides an efficient mechanism for session resumption without server-side state. RFC 4705 uses the `cookie` predicate: the client saves the state for the server. To prevent client from changing the state, the state includes an authenticating tag, using a symmetric message authentication key, known only to the server(s) . Once RFC 4705 is adopted by browsers, the overhead of SSL/TLS session resumption would not be a critical problem.

However, such adoption may take substantial time. Furthermore, this solution requires each server to know the secret information (keys). This can also be problematic, e.g. when using site hosting or content distribution networks. The second technique, therefore, avoid the use of SSL/TLS entirely.

Instead, to authenticate their web pages, the sites will *digitally sign* the contents, e.g. using the XML DSIG standard [ERS02]. Digital signing is computationally-intensive, however it can be applied once to satisfy all requests for the same (static) web page; some convention may allow limited variability in the content, e.g. for customized values such as user name.

We next discuss how a toolbar should indicate, to the user, the identify of a (securely authenticated) web page.

2.3 Site Identification Indicators and Spoofed-Indicators Attacks

Site identification indicators may help users identify trustworthy sites, known by name (logo, trademark, etc.), and/or known to the user from previous visits. Site identifiers can be *certificate-*

derived or *user-customized*, and appear as textual strings (e.g. organization name or `petname`) or graphical elements (e.g. logo or picture).

For **certificate-derived identifiers,** TrustBar, and later also version 7 of Internet Explorer, use the organization name in the SSL/TLS certificate, and the text `Identified by:` followed by the name of the certification authority (CA) responsible for this identification. The identification of the CA is important as long as users do not indicate which certificate authorities they trust, or delegate this authority to an appropriate service; browser vendors are not offering evaluation of Certificate Authorities. In particular, by identifying the CA, users can differentiate between a CA that issues low-cost `domain-validated` certificates, and an CA which issues only `full` certificates, which are more expensive but better validated; see Subsection 1.2.

User-customized identifiers are individual to each user, selected by the user and/or randomly by the system. User-customized identifiers can be implemented by the server, the browser, or a browser extension (toolbar). User-customized toolbars include TrustBar [HG04] (user-selected graphics or text), Petname [C06] (user-selected text), Dynamic Security Skins [DT05] and PassPet [YS06].

The simplest deployment of user-customized identifiers is by a personalized (login) webpage, typically containing a user-selected picture, e.g. [P04, Y06]. In these schemes, the server normally selects the picture based on a cookie kept and provided by the browser. In [Y06], the picture identifies the computer (more correctly, the browser), while [P04] allows users to move computers or recover from lost cookie, but their solution may allow a Man In The Middle (MITM) attack. No usability experiments with these methods were published yet.

Illustration 5 An SSL protected page with TrustBar, with user-defined (or default) logo, and identification of the Certificate Authority.

In usability experiments of TrustBar, the user of user-customized identifiers resulted in better detection rates and throughput [HG04], compared to certificate-derived identifiers, and much better compared to classical browser indicators. Further experiments are required to confirm this advantage, and to compare between different types of customized identifiers (chosen randomly or by user, textual vs. graphical, in toolbar vs. provided by the site in the page).

Furthermore, user-defined identifiers may offer better protection against *spoofed indicators* attacks. Spoofed indicator attacks present fake security indicators (e.g. toolbar), in order to trick the user, e.g. to ignore the real security indicators. Several papers [DTH06, FB*97, LY03, YSA06, YYS02] investigated different spoofed indicators attacks, and presented possible defenses. We only discuss the *fake internal window* variant of the spoofed indicator attack.

The fake internal window attack uses a single browser window, controlled by the attacker. In the content area of this window, the attacker *emulates* another browser window, which appears to contain the (spoofed) web page of the victim organization, e.g. bank login page. This attack is most effective if the attacker is able to hide all (or most) indicators of the existence of the `external` window, such as borders, menus and other areas not defined by the page. For better security against

such defenses, web pages should not be allowed to hide such indicators.

However, even if the attacker cannot avoid some visibility of the `external` page, there is a risk that users may not pay attention to the external page, and not realize that the `internal page` is a fake browser window. The attacker controls the entire contents of the `internal', fake page, and in particular can display there any (known) security indicator, including certificate-derived identifiers such as organization name and logo. User-customized identifiers may help users detect the fake page, if the adversary is unlikely to be able to correctly clone the identifiers.

To conclude, improved site identification can reduce the success probability of spoofing and phishing attacks. User-customized identifiers, as in TrustBar and Petname, seem even more effective, and may also provide some defense against spoofed indicators attacks. However, significant probability of spoofing may remain, mainly due to the fact that users often are more influenced by the content of the page, than by toolbars. Ideally, security mechanisms should *prevent* the display of misleading content, instead of displaying it (in most of the window) along with contradicting identifying or warning indicators (in smaller toolbars). In the next section, we discuss some ideas toward this ambitious goal.

3 Defending against malicious content and malware

The basic function of browsers is to download content – web pages and other objects – from servers. Many of the threats against web users, are due to malicious content, ranging from pictures (e.g. unauthorized use of bank's logo for spoofing) to malicious code (*malware*), including `regular` programs as well as malicious scripts and applets. In the first subsection, we discuss existing defenses against malicious content and specifically malware. In the second subsection, we discuss a new direction for defense – *web content registries*.

3.1 Current defenses against malicious content and malware

Current browsers offer some defenses against malware, but almost no defenses against malicious non-executable content. Essentially, the only available defense against malicious content consist of blacklists of suspect sites/objects. However, blacklists are ineffective against determined attacker, that changes domain names (at negligible cost) and/or IP addresses (using IP addresses of `zombie`, broken-into computers).

Additional defenses exist specifically against malware:

- **Sandboxing:** prevent remote code from harming the user, by limiting its abilities. The most well-known application of sandboxing is for Java applets. Sandboxing is also used for other automatically-downloaded code, in particular for JavaScript and other remotely-downloaded scripts. There were numerous bugs and exploits due to failures to properly enforce sandboxing on applets, scripts or other kinds of remote code. Furthermore, sandboxing allows the remote code to control the user interaction (in a specified; limited area of the screen); therefore sandboxing cannot prevent malware from attacks based on presentation of misleading information inside this area (e.g., unauthorized presentment of corporate logo).
- **Detection of known malware (by `signature`):** malware may be detected and blocked by identification of it, using known patterns (`signature`), often by anti-virus utilities. This can fail when the attacker adaptively modifies the malware to avoid detection, and especially for easily modifiable scripts.
- **Certified code:** code (or more generally, content) which is *digitally signed* and certified by an appropriate authority to be trustworthy. This indicates that the code is not malware. Certification can be effective, if executing only certified code, with appropriate, careful certification process and heavy penalties to submitters of malware. However, existing efforts of certifying code [R95,M] have limited success; and code executed in sandbox (mainly

applets and scripts) is rarely certified.

We conclude that existing browsers defenses against malicious content, and in particular malware, are insufficient. Secure implementation of the sandbox model is challenging, and there have been numerous vulnerabilities allowing attacks via scripts, applets, documents and even images.

In particular, some of the most common web attacks involve different forms of *cross site scripting (XSS)* [C03], where a web site is tricked into sending a malicious script to the browser, usually embedded in a web page. Such scripts often expose confidential information of the user of the site, e.g. cookies allowing access to the user account, or otherwise attack the user, the site, or other Internet services.

Furthermore, attackers can send *deceptive* content, attacking the user directly. This is how most phishing/spoofing sites work: they copy the content, e.g. images and logos, of the original, cloned site, thereby misleading users to believe they are at the trusted site (e.g. bank or download site). In particular, when users think they reached a reliable download site, they may download and install programs, overriding any security warnings.

How can browsers improve protection against malicious content, and in particular malware? One possible answer may be to try to fix the certified code/content approach, which is technically sound and secure, and failed in deployment. In the next subsection, we present one idea of how to possibly `fix` the certified code/content approach, namely by using automated, inexpensive `registration` process as a complement to the manual, expensive `certification` process.

3.2 Content Registry: Defense against Malicious Content

From the previous section, we conclude that current browsers often fail to protect against malware, and malicious content in general. Some improvement may be possible by adopting improved identification and security indicators, as discussed in Section 2. In particular, when using customized identifiers, we found that spoof rate fell below 5% [HG04]. Unfortunately, [HG04] only compared different indicators, over short periods; we expect much higher spoof rate over realistic usage and long periods.

Users often trust the contents of the web page, and may ignore `external` security indicators such as padlock [DTH06]. This trust is unjustified, since the contents are controlled by the website, or by a Man in the Middle attacker (for unprotected sites). Most of the proposed solutions try to improve awareness of the security indicators, and to block suspected `bad` sites. However, as we discussed, this provides only limited protection.

In this subsection, we consider a different, somewhat revolutionary idea: *trusted content registry service*, similar to the certificate registry services presented in Section 1.2. Registration is a low-cost alternative to certification: all content on the web is registered automatically and free of charge, similarly to search engines. Browsers and content-filters (e.g. firewall) can use the registration data, to *limit* or block suspect content. *Content limiting* refers to using it with some precautions and restrictions, e.g., reducing the capabilities of code (applets, scripts), or fading images. Using the content registry service, we attempt to *fulfill* the users expectations that the browser displays and uses (runs) only trustworthy content.

Trusted content registries can list code (scripts, applets, etc.) as well as non-executable content (pictures, etc.). In both cases, the client `trusts` content which has been registered long enough, say for T days; content which is not registered or only recently registered is limited or blocked.

Registry entries may also include the IP address, domain name and/or URL in which the content was published. For images, this prevents unauthorized use of a logo in an unauthorized site (see below on extending the solution to defend against Man in the Middle attackers). For code, this prevents an attacker from registering malware at an isolated site, and later sending this malware from other sites, e.g. in MITM or XSS attacks.

Hopefully, registered content, coming from the registered IP address (and domain name/URL), will rarely be malicious, for two reasons:

1. Malicious content, including unauthorized use of content, would hopefully be detected (and blacklisted and/or removed from registry) before the *T* days period. In particular, in the case of images designed to emulate the appearance of a trusted site (e.g. bank or download site), it may be possible to identify imposters by scanning new registry entries.
2. Currently, attackers often use very-short lived sites for spoofing and other attacks; one reason is that such attacks are often based in victim computers, which often do not have a fixed IP address, and without authorization or knowledge of the administrators. It can be harder for the attacker to maintain the malicious content or code to remain available in the site for *T* days, and the risk of detection is higher. Similarly, when attackers provide fake content from a site using MITM or XSS attacks, this content is short lived, and furthermore site owners can detect attempts to register new content to their site. Long-lived malicious sites can be effectively blacklisted.

Obviously, not all recently published content is malicious. In particular, many sites display new content frequently (e.g. pictures in a news site). Also, currently there are many (legitimate) sites which use scripts which contain frequently changing data such as user identifier. Therefore, we need careful design of the defense mechanism, to minimize disturbance to legitimate, recently-published or recently-modified content.

We may use different heuristics to allow new content or frequently changing content. For example, we can allow a news site, identified by its IP address, to present new images. More generally, we can allow a `white-list` of `trusted` sites, for which the registry validation is waived.

Furthermore, browsers can allow new objects (unregistered or recently registered), if certified by a *rating authority*; this allows rapid publication of new objects (scripts or images). Rating authorities may also indicate specific ratings for objects, e.g. `PG-13`, possibly using PICS or similar markup [MRS96]. The user may indicate the trusted rating authorities, or delegate this choice, e.g. to antivirus service provider. By using well defined, easily disputable ratings, security service providers will be able to use reputation and accreditation services to ensure the quality of the content filtering process. Hence, the registration and certification mechanisms are complementary.

```
On query(h,IP,URL):
  If IP ∈ BlackList :
     Return (NO);
  If h ∈ Ok(IP) :
     Return(Ok);
  If h ∉ Objects :
     Obj = GET(URL);
     If h=hash(Obj) :
       add h to Objects;
       Obj(h)=Obj;
       add <h,IP,date()> to Pend;
  Return(NO);
For every <h,IP,date> ∈ Pend
       s.t. date<date()- T
       and h ∈ Objects:
     add h to Ok(IP);
```

Example 1: Content registry algorithm (simplified)

We can also reduce disturbance by limiting content rather than blocking it, e.g. displaying images with fading or other visual modification (against unauthorized display of logo), or allowing a script to run with reduced priviledges. Browsers can also allow users to manually waive content blocking and limiting for a particular page.

Blocking/limiting MITM attacks. The registry as described was based on IP addresses (and domain name/URL), and hence could be circumvented by a MITM attacker, providing incorrect

content for a page requested by the browser. It also does not defend against Cross Site Scripting (XSS) attacks, exploiting a weakness in the server to send adversary-controlled script, or other code/object, to the browser. Such attacker can exploit the whitelist mechanisms to avoid registry validation entirely, or send the fake content using the address of the legitimate server, exploiting the IP (or domain/URL) based lookup (the `Ok(IP)` table, in the simplified content registry server code in Example 1).

A site can defend itself against XSS attacks, by always signing all of its scripts (and other objects), and signaling to the browser to only allow signed scripts (or content in general).

Defending against MITM attacks seems to require elimination of the (address based) white-list, and therefore, it also requires support by all (or most) sites, to avoid `false positives` (blocking/limiting of legitimate content). Such support may consist of content signing and/or certification.

Subtle issues. Blocking or limiting unregistered content is a drastic change, and requires careful consideration of many subtle issues beyond this article. In particular:

1. Currently, scripts often contain variables such as dates or customer identification. As a result, such as script may appear to be rapidly changing. One solution is to separate script code (which does not change frequently) from data, however it is preferable to avoid such changes. Instead, it seems that it may be possible to automatically detect changing areas within the script, and remove them from the registry.
2. Attackers may try to create misleading pictures by superimposing several legitimate pictures, e.g. tiny pictures of few pixels. This may be addresses by restricting the number of pictures and other heuristics.
3. The discussion required hashing of `objects`; this is easy for images and other types of content which are handled as objects in the HTTP protocol, but may be harder to apply to scripts and other elements sent within a web page. This may require careful parsing by the browser; it may be hard to implement such functionality in an extension.
4. The computation of hash for each object is efficient and not expected to be a bottleneck. However, the lookup operation at the registry should be well designed for efficiency, possibly using Domain Name System (DNS) records and queries, and/or other efficient caching mechanisms.
5. The registry may identify clients and secure queries, to protect against excessive use and Denial Of Service attacks, e.g. providing incorrect URL.
6. Our discussion focused on code (especially scripts and applets) and images. Other forms of content can also be malicious, including even attacks using some HTML mechanisms.

Conclusions

Web browsers implement the SSL/TLS protocols, defending the communication between client and server from man-in-the-middle and eavesdropping adversaries. However, browsers provide insufficient protection against simpler and more common attacks, such as phishing, spoofing and malware. We explained the vulnerabilities of current browsers, and discussed some possible improved defenses. Some of the defenses were tested experimentally, with encouraging results; we hope that the use of such defenses may significantly improve protection of web users from phishing, spoofing and malware.

References

[C03] Steven Cook, A Web Developer’s Guide to Cross-Site Scripting, SANS Institute, GIAC practical repository, January 11, 2003.

[C06] Tyler Close, Petname Tool: Enabling web site recognition using the existing SSL

infrastructure, presented in W3C Workshop on Transparency and Usability of Web Authentication, March 2006, New York City. Available from http://www.w3.org/2005/Security/usability-ws/papers/02-hp-petname/.

[DT05] Rachna Dhamija and J. Doug Tygar. The battle against phishing: Dynamic security skins. In Proc. ACM Symposium on Usable Security and Privacy (SOUPS 2005), pages 77–88, 2005.

[DTH06] Rachna Dhamija, J. Doug Tygar and Marti Hearst, Why phishing works. Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 581-590, Montréal, Québec, Canada, 2006.

[ERS02] Eastlake, J., Reagle, J. and D. Solo, XML-Signature Syntax and Processing, RFC 3275, March 2002. Also a W3C Recommendation available at http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/.

[F05] Rob Franco, Better Website Identification and Extended Validation Certificates in IE7 and Other Browsers, published in Microsoft Developer Network's IEBlog, November 21, 2005 (http://blogs.msdn.com/ie/archive/2005/11/21/495507.aspx).

[FB*97] Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach. Web Spoofing: An Internet Con Game. Proceedings of the Twentieth National Information Systems Security Conference, Baltimore, October 1997. Also Technical Report 540–96, Department of Computer Science, Princeton University.

[FS*01] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster, Do's and Don'ts of Client Authentication on the Web, in the Proceedings of the 10th USENIX Security Symposium, Washington, D.C., August 2001.

[HG04] Amir Herzberg and Ahmad Gbara, Protecting (even) Naïve Web Users, or: Preventing Spoofing and Establishing Credentials of Web Sites DIMACS Technical Report 2004-23, May 2004. Latest version available from http://eprint.iacr.org/2004/155.

[LY03] Tieyan Li, Wu Yongdong. "Trust on Web Browser: Attack vs. Defense". International Conference on Applied Cryptography and Network Security (ACNS'03). Kunming China. Oct. 16-19, 2003. Springer LNCS.

[M] Introduction to Code Signing, Microsoft. Available online at http://msdn.microsoft.com/workshop/security/authcode/intro_authenticode.asp

[MRS96] Jim Miller (Ed.), Paul Resnick and David Singer, Rating Services and Rating Systems and Their Machine Readable Descriptions Version 1.1, W3C Recommendation, http://www.w3.org/TR/REC-PICS-services, October 1996.

[P04] Passmark Security: Protecting Your Customers from Phishing Attacks: an Introduction to Passmarks. (http://www.passmarksecurity.com/), 2004.

[R95] Aviel D. Rubin, Trusted Distribution of Software Over the Internet, Proc. ISOC Symposium on Network and Distributed System Security, pp. 47-53, February, 1995.

[R00] Eric Rescorla. SSL and TLS: Designing and Building Secure Systems. Addison-Wesley, 2000.

[SZ*06] J. Salowey, H. Zhou, P. Eronen and H. Tschofenig, Transport Layer Security Session Resumption without Server-Side State, RFC 4507, Networking working group, Internet Engineering Task Force (IETF), May 2006.

[WMG06] Min Wu, Robert C. Miller and Simson L. Garfinkel, Do security toolbars actually prevent phishing attacks?, Proceedings of the SIGCHI conference on Human Factors in computing systems, pp. 601-610, Montréal, Québec, Canada, 2006.

[Y06] Yahoo, Give password scams the boot with personalized sign-in seals,

https://protect.login.yahoo.com/, 2006.

[YSA06] Zishuang (Eileen) Ye, Sean Smith and Denise Anthony. Trusted Paths for Browsers. ACM Transactions on Information and System Security (TISSEC), vol. 8 , no. 2, pp. 153-186, May 2005.

[YS06] Ka-Ping Yee and Kragen Sitaker, Passpet: convenient password management and phishing protection, Proceedings of the second symposium on usable privacy and security, pp. 32–43, 2006.

[YYS02] Eileen Zishuang Ye ,Yougu Yuan ,Sean Smith . Web Spoofing Revisited: SSL and Beyond . *Technical Report TR2002-417* February 1, 2002.