SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2012; **5**:535–544
Published online 13 July 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.349

RESEARCH ARTICLE

A survey of client-side Web threats and counter-threat measures

Daniel Hein[1], Serhiy Morozov[2] and Hossein Saiedian[2,3]*

[1] Garmin International, Olathe, KS 66062, U.S.A.
[2] EECS, University of Kansas, Lawrence, KS 66045, U.S.A.
[3] ITTC, University of Kansas, Lawrence, KS 66045, U.S.A.

ABSTRACT

The increasing frequency and malevolence of online security threats require that we consider new approaches to this problem. The existing literature focuses on the Web security problem from the server-side perspective. In contrast, we explore it from the client-side, considering the major types of threats. After a short threat summary, we discuss related research and existing countermeasures. We then examine intuitive human-oriented trust models and posit a flexible, multilayer framework to facilitate automated client-side decision making. The proposed suggestions are not intrusive and do not require advanced technical knowledge from end users. Copyright © 2011 John Wiley & Sons, Ltd.

KEYWORDS

information security; Web security; browser attacks; cross-site scripting; client-side security; trust and trustworthiness; policy enforcement

***Correspondence**

Hossein Saiedian, ITTC, University of Kansas, Lawrence, KS 66045, U.S.A.
E-mail: saiedian@eecs.ku.edu

1. INTRODUCTION

Modern Web browsers are an indispensable tool for information retrieval, entertainment, shopping, and banking online. At the same time, the popularity and ubiquity of Web browsers has made them attractive targets for attack. In fact, a recent Internet threat report from Symantec calls out several disturbing trends in this domain [1].

(1) Home users are the most highly targeted sector and account for 95% of all attacks.
(2) Client-side attacks often originate from questionable sources such as malicious Web sites or spam. Although the best practices advise end users to avoid such type of content, it appears that *attackers are using legitimate and trusted sites as a basis for attacks*.
(3) The emergence of underground economy servers as the *de facto* trading place for illicit information, for example, credit card or social security numbers, is indicative of the increased professionalization and commercialization of malicious activities.

Not only that *individuals are the primary attack targets*, but also that *the attacks are more malevolent in nature*. Instead of prank-like attacks, where the goal is to earn bragging rights, *attacks are increasingly being linked to organized crime* [2], where the goal is identity theft and other exploitation for financial gain [1]. Attacking a large corporation may offer a larger payout, but involves a greater risk. Preying on individual users involves less risks and more stable financial rewards.

Compounding and complicating this situation are the requirements thrust upon a user desiring to safely and securely surf the Internet. A typical user-oriented list of security recommendations [3] contains a variety of guidelines, suggestions, and best practices for safe browsing. Such recommendations include staying away from "suspect" Web sites and ensuring that applications and operating system software include the most recent patches and fixes [3]. Such expectations are unrealistic because detecting a "suspect" Web site is not necessarily straightforward—even for highly technical users.

Today attackers can easily produce friendly looking Web sites with community appeal [4]. Additionally, even "trusted" Web sites can be compromised to serve malicious content to users [1]. Usually, an otherwise friendly Web site can find itself serving malicious content simply as a result of syndicated ad content [5]. Furthermore, many applications, browser plug-ins, and content handlers require manual updates to stay current with the latest

security-related patches. However, even experienced administrators are slow to apply security-related updates [6]. In general, people wish to surf freely, without having to make a judgment call before following an interesting link. Therefore, moving the burden of security enforcement to inexperienced users is naive.

Given the current state of affairs, it is time to rethink the design principals of modern browser software and networking protocols. Current research indicates that the application layer will continue to be heavily targeted [7, 8] because the root cause of many security vulnerabilities stem directly from software design and implementation [9, 10]. To improve security, we need to fully understand existing threats and their enabling factors. The first objective of this paper is to give a concise overview of current threats and to summarize existing research concerning Web browser security. Once we have established the context for better browser, protocol, and infrastructure security, we call out notions of trust on the Web, posit the quantification of trust, and outline a framework for decision making based on resulting trust values.

This paper is organized in five sections that accomplish these objectives. Section 2 reviews various browser attacks. Section 3 then reviews ongoing research efforts to improve browser security. Towards the end of Section 3, we allude to *avoidance* as a strategy to prevent interaction with malicious sites. We consider trust relationships on the Web and how users enter into such trust relationships in Section 4. Finally, in Section 5, we consider how trust values may be used to automatically avoid unwanted sites.

2. BROWSER ATTACKS

This section reviews various attacks against Web browsers, while pointing out the fundamental flaws in remedies used by modern applications. Many security threats are possible because of our trust and dependency on underlying technology. For example, past research has shown that most users do not understand the complexities of certificate messages [11]. In fact, even computer science students tend to accept forged certificates as they "click-through" typical warning dialogs [12]. Therefore, modern Web threats are rooted in poor software design as much as human carelessness.

Many Web-based attacks are very similar in nature and involve a few basic underlying principles. One of such principles, known as the same origin policy (SOP), is used by the browser to sandbox mobile code, typically written in Java or JavaScript [13]. The SOP, and flaws in its assumptions, is a popular research topic [13–15]. In fact, the SOP has a critical "loophole" in various extensible hypertext markup language elements, like iFrames [5] and images [13]. This loophole allows scripts hosted on alternate domains, to be included into the current page. Once loaded, the malicious script gains extended capabilities and can access other page elements [13]. SOP fails to produce security because it allows cross-domain access, which renders it inadequate for providing the needed domain isolation. The following three attacks are possible due to this limitation.

2.1. Basic reconnaissance

Through simple reconnaissance, information about a client's interaction with other sites can be inferred. This type of attack is often used as a stepping stone for more advanced attacks, such as cross-site request forgery (CSRF). For example, Johns describes how a malicious Web site can use standard JavaScript method calls to deduce simultaneous connections of interest, for example, bank and e-commerce sites [13]. This vulnerability may violate user privacy as well as provide additional information for more advanced attacks described as follows.

2.2. Cross-site request forgery

A CSRF is often the ultimate goal of a longer attack chain. A CSRF attack is typically characterized by replaying an authenticated request in order to gain unauthorized access or to carry out some unauthorized action, for example, transferring funds to an off-shore account. The items replayed often include a particular URL or a session cookie. Often, executable script code comes to be bootstrapped into the current page, via aggregation mechanisms allowed by JavaScript and the SOP [5]. Once loaded, this script can replay hypertext transfer protocol (HTTP) requests within the context of a currently authenticated session. There are various entry vectors for CSRF attacks, the most common being cross-site scripting (XSS).

2.3. Cross-site scripting

In XSS attacks, a script hosted on a different server is executed with the privileges of the current page. Such attacks are commonly carried out by enticing the user to click on a malicious link that exploits a weakness in the server's input validation and error reporting mechanisms. A common example is the case where a user is shown an HTTP 404 message naming the requested file [16]. In such cases, the attacker crafts a link in such a way that the requested file name encodes script content. Failure to appropriately recognize and filter the injected content means that the browser will execute the malicious code as part of a normal page rendering process.

2.4. DNS infrastructure

An important element of many browser threats is trust and dependence on the underlying protocols such as the domain name system (DNS). Attacks on Web applications based on DNS result manipulation are called DNS infrastructure attacks. They target the domain name

DOI: 10.1002/sec

service but have the ultimate goal of affecting the dependent application, for example, Web browser [17]. DNS attacks either corrupt cached resource records (DNS cache poisoning) or spoof authoritative name servers (DNS hijacking) [17]. The security of the Web client rests on the security of the DNS [18]. In fact, by targeting a layer beneath the Web layer, *the attacker can completely subvert the SOP or any other controls based on the assumption that the domain name properly maps to an internet protocol (IP) owned by a given content provider*. However, as these attacks immediately affect the DNS, the use of secure DNS (DNSSEC) will effectively prevent them.

A compromised DNS server could redirect a user to a malicious Web site without the user's knowledge. For instance, if the browser requested an IP address for the Web site http://hotmail.com, a server could point it to a malicious site that looks just like the original. This type of attack is called phishing [19,20]. Usually the attacker attempts to steal the authentication credentials, by making a user think that he/she is logging into the original site. This kind of attack is especially dangerous because an average user cannot distinguish between a real Web site and a well-made fake [21]. In fact, it is relatively easy to make nearly perfect copies of a legitimate Web site, with no visible signs of forgery.

2.5. DNS rebinding

A DNS rebinding attack is not a direct attack on DNS infrastructure. The basic idea of this attack is to change the IP address "out from underneath" the browser, thereby circumventing any remaining controls imposed by the SOP. Typically, a malicious Web site sets up a short-term DNS record, causing the client to re-request its IP address. However, the attacker changes the DNS record such that a new IP is returned.

Such attacks are considered dangerous, considering that the session can be rebound to a private address behind a firewall. Because firewalls typically allow HTTP traffic through (port 80), such rebinding can be used to circumvent the firewall and further profile other machines on a protected intranet [14]. Although this attack is accomplished via the modification of DNS resource records, the modification is legitimate. Therefore, using DNSSEC does not counter a DNS rebinding attack.

2.6. Summary

The list below summarizes the browser attacks discussed in this section in order from the least to the most dangerous.

(1) Basic reconnaissance.
(2) DNS rebinding.
(3) CSRF.
(4) XSS.
(5) DNS infrastructure.

Basic reconnaissance is the least harmful of the browser attacks. Usually, the goal of this attack is extracting information that is private but not crucial, for example, sniffing user agent information or browser history. This information is not explicitly protected, which is the reason reconnaissance attacks work. DNS rebinding attack can be harmful, but it assumes that a user wants to visit a malicious site. This can be arranged by persuading a user to click a link, but the users will unlikely complete any serious transactions, unless the site has a secure certificate available. CSRF is more dangerous because the malicious code is acting on the behalf of the user and the destination Web site cannot distinguish between the two. However, it is possible for a Web site to ignore duplicate requests. Because the protection for this vulnerability resides on the server side, this type of attack is very dangerous. Finally, the DNS infrastructure attack is the most dangerous. The entire Web is built in layers, where each layer trusts that the functionality below it is trustworthy. Therefore, if a DNS server is compromised, any security precautions that rely on it will fail.

3. ONGOING COUNTERMEASURE AND CONTAINMENT EFFORTS

This section reviews efforts to counter the attacks discussed earlier. The most straightforward solution to Web security woes would be to disable Java and JavaScript. However, even though almost every known attack stems from the use of mobile code, the user experience of modern Web applications would be unacceptable without it. There are many approaches to countering mobile code threats including "Band-Aid" solutions applied after the fact, improved design of script execution control, and policy solutions that seek to avoid the problem altogether. We consider these approaches in the following sections.

3.1. Band-Aids

Band-Aid solutions cover up the fundamental problems in design and implementation. These solutions include "penetrate-n-patch" software updates as well as browser plug-ins to counter Web threats. The term "penetrate-n-patch" refers to the paradigm of patching up security holes after the fact [9]. Unfortunately, this is by far the most widespread approach to software security in practice. Furthermore, this approach often introduces new software bugs and other unexpected security vulnerabilities.

One type of Band-Aid solution is the "add-on" approach, where additional software components are used to repel threats at their point of entry. Firefox plugin NoScript is this type of add-on [22]. It allows the user to control which sites can execute mobile code, for example, PDF, Flash, and Silverlight contents. This plugin solves the immediate problem of accidentally surfing onto a malicious Web site, but it provides no means of evaluating

the trustworthiness of a site. Additionally, once a site is trusted, all scripts originating from it are allowed, leaving users open to XSS attacks.

A possible improvement to NoScript would be to keep track of trusted and suspicious sites and automatically decide whether or not to execute their code. Tiwari, Bansal R., and Bansal D. developed such a system [23]. The white and black lists are cross-validated against a security advisory site, so an incorrectly classified Web site may be appropriately labeled. The main advantage of their approach is the fact that this plugin barely interferes with the speed of the browser, thus providing the user with transparent security.

Alternatively, NoScript could be extended to contain a list of "recognized scripts" by implementing browser-enforced embedded policy (BEEP) [24]. Specifically, the "whitelist" approach could be implemented in NoScript such that only those scripts designated by authors of trusted Web sites are allowed to execute on the client. Usually this is done by providing cryptographic hashes for each script. Results published by Jim, Swamy, and Hicks found this improvement to be effective in preventing known attacks [24]. BEEP is an effective tool against XSS attacks, but it is not a widely adopted method. In fact, preventing XSS attacks is largely a standardization, conformance, and implementation effort.

3.2. JavaScript instrumentation

One approach to countering the mobile code problem is through rewriting key portions of the JavaScript code to ensure compliance with the client's policy. Some approaches also consider rewriting of higher-order scripts or code that has been generated dynamically [25]. The instrumentation approach identifies JavaScript calls that are traditionally problematic and adds additional code to ensure that users have visibility into the inner workings of the script. This approach is fine grain and provides significant protection.

Other approaches are not as fine grained. Even signed code cannot achieve this level of control [25]. The primary advantage is insight into the actual "actions" carried out by the script. Simply because some site signs a script does not necessarily mean that the script is benign. For instance, a malicious site could easily sign a script, which would be executed according to a standard BEEP protocol. However, such a fine-grained approach requires a higher degree of user involvement, which is not desirable for a majority of Web users.

The key limitation of this approach is the level of user involvement. Most users will not understand code or system-related information presented by the rewriting engine. In fact, the users will become frustrated from user interface notifications generated by the rewriting engine as it encounters new and varied scripts. Therefore, the fine-grained nature of this approach will interfere with the user's browsing experience and will likely be deactivated. Because of these issues, we favor a solution that offers less user involvement and redirects the decision point to the question of who to trust. However, the rewriting mechanism could be useful in the context of a larger framework, which dynamically enables it depending on the trust rating (TR) for a particular site.

3.3. Virtualization

Virtualization solutions seek to provide a system-level sandbox for mobile code. Rather than attempting to prevent or restrict execution of the script, this approach allows bad things to happen, but in a "virtual" environment. In fact, any damage is completely reversible by deleting the virtual machine instance [26]. Virtualization is often used by honey clients to identify malicious Web sites [4] but does nothing to prevent attacks that exploit the "real" session between the virtualized client and a host site. Although execution attributes on the local machine are virtual, the machine is still processing real HTTP(S) traffic.

A notable work in this area is Tahoma [26]. It utilizes a virtualization to run conventional browsers without actually defining its own browser architecture. Tahoma models a browser as an operating system, with the Web sites as "programs". The operating system uses a separate browser instance for each Web site in order to ensure isolation. The user experience offered by Tahoma closely parallels that offered by NoScript. Both solutions prevent script execution by default and require a user to explicitly approve code from a particular Web site. Tahoma is also similar to BEEP [24] in that it uses manifests on the server side to dictate the policy.

3.4. Architecture

Sound software architecture is essential for proper security. In fact, one of the key contributions of good architecture is the ability to reduce complexity of critical components, allowing these components and the data flowing through them to be more rigorously analyzed. Simple design facilitates model checking, which would be impossible in more complex systems.

For example, the Opus Palladianum (OP) browser centralizes analysis through a browser kernel [15], which coordinates the message passing various subsystems: user interface, Web page, storage, and network. A key advantage to this design is that the browser kernel can be kept relatively small, facilitating its analysis. Another key contribution of the OP browser is its notion of *provider policy* [15]. It binds referenced objects to their domains of origin, rather than allowing such objects to become part of the page. Therefore, a movie from Apple's Web site and a movie from YouTube can be included on a page hosted by uiuc.edu, but they cannot access any of the resources on the page [15]. As a result, this approach offers more security than SOP without sacrificing functionality.

Another approach by Ioannidis and Bellovin [27] enforces the separation of privilege concept [28]. Many

DOI: 10.1002/sec

browsers execute mobile code with the privileges of the user. The security could be improved by tying the code execution to individual processes [27]. By doing so, this approach provides isolation and prevents malicious mobile code from executing under unauthorized privileges.

3.5. Secure DNS

Given that much of Web security is based on avoiding suspect sites, the information that a browser uses to reference a particular site must be correct. Secure DNS, or DNSSEC, secures the DNS infrastructure on top of which Web security is based. It provides for cryptographic authentication of resolution data [29]. However, DNSSEC is not without its own problems.

First, DNSSEC differs from conventional public key infrastructure (PKI) in its use of cryptographic keys, key escrow, and signature structure. This could hamper adoption because it is unfamiliar to designers and cryptographic functionality may affect performance of the system [17]. Although DNSSEC is mandated for the ".gov" top-level domain [17], it is not yet widely deployed in the Internet. Finally, it does not provide end-to-end security, so an evil man-in-the middle, between the client and the caching name server, can alter response data.

3.6. Avoidance

Avoidance is one of the simplest and most popular strategies. In addition to avoiding malicious sites, it can prevent exposure to objectionable content, usually in the context of parental control. Because malware is increasingly installed by simply visiting a malicious site [5], avoidance is a reasonable approach to increasing security.

One solution that employs avoidance is Web of Trust. It is a Web-based service that works in conjunction with a browser plugin to color links: red indicating "malicious" and green indicating "safe" [30]. Web of Trust is backed by a large user community where individuals rate Web sites on overall trustworthiness, child safety, vendor reliability, and privacy. This "power in numbers" approach is very effective. By distributing the work-load among its users, complex tasks such as policing the Web can be realized. In fact, much of Section 5 is dedicated to exploring the various aspects of a community rating model.

Another avoidance service is OpenDNS. It combines both DNS and a user policy [31]. As a result, it can filter records from any of the 30 categories including adult, drugs, social networking, P2P file sharing, and so forth. Depending on the policy, a user might filter only malicious sites and allow all others. However, when attempting to visit a Web site explicitly prohibited in the user policy, its domain name will not be resolved.

3.7. Summary

The list below summarizes the countermeasures presented in this section in order from the least to the most effective.

(1) JavaScript control and instrumentation.
(2) Virtualization.
(3) Secure architecture.
(4) Secure DNS.
(5) Avoidance.

Browser plug-ins that disable or modify JavaScript execution are not solid solutions for browser security. There are many ways to circumvent these countermeasures either by purchasing a new domain name or obscuring your source code. Virtualization is more effective because the damages of the attack are contained in a virtual instance. However, it does not prevent one from visiting a malicious site. Secure architecture is one of the more effective approaches. The goal is to design browsers with security in mind. However, it goes completely against user friendliness and requires major updates to most existing software. Secure DNS is an effective measure against DNS infrastructure attacks, which are the most dangerous. Finally, avoidance is the most reliable way to ensure Web security. Instead of trying to modify the site or alleviate the symptoms of vulnerability, this approach simply prevents it from happening. However, there is uncertainty about which sites people should avoid and how to establish the trustworthiness of a site. The following section discusses this topic in more detail.

4. TRUST AND TRUSTWORTHINESS

Most browser attacks exploit the trust relationship that the client places on a Web site. Given that Web threats are both increasingly prevalent and successful, this trust is often misplaced. In this section, we review intuitive models of trust among humans and show how to quantify it.

4.1. Human trust models

Humans establish trust via history, reputation, credentials, and/or contract. History and reputation are closely related and are based on human qualities like honesty, fairness, loyalty, and professional qualifications. History is a qualitative, perceptual measure of these qualities over time. On the other hand, reputation is established by a larger group. It relies on *consensus opinion* concerning an individual's trustworthiness. Credentials say little about a person's qualities but do provide a third-party certification of an individual's qualification for a given task. Finally, a contract provides a means of recourse, should a trusted individual fail to deliver services as trusted. Contract in particular is interesting in that the parties typically do *not* trust each other—hence the contract. Rather, the parties to a contract trust the higher authority of the government and the legal system backing the contract.

4.2. Web trust models

Many experts agree that misplaced trust results from the roots of the Internet itself, which was created to foster

unrestricted information sharing. Back then, security was not a primary design consideration [8]. As the Web has matured, security has been an ever-present concern, and various mechanisms have been retrofitted into the original protocols.

Of all the human trust models, history and reputation implicitly apply to modern Web. In fact, these trust relationships reveal themselves daily as users surf the Web. It is natural for an individual to form a positive opinion about a site that they frequently visit. Just as with interpersonal relationships, users build their trust based on historical interactions. However, this trust may be completely unjustified because an average user is unaware of the policy, practices, and implementation of the Web site, for example, how information may be collected or the quality of the server software. Furthermore, when surfing the Web, a user does not utilize a single service long enough to establish any history.

However, there is less of a basis for credentials. Although some organizations allow compliant Web sites to display a seal of approval [32], there is no *infrastructure* supporting the concept of credentials to inform automated decision making. For instance, TRUSTe explicitly certifies a site along trust dimensions of interest. However, rather than a seal, we seek a trust quality rating, cryptographically signed by a certifying organization.

Although one might think of the PKI as an issuer of credentials, this is not correct. It only serves as a third-party confirmation of identity. To give out credentials, one must determine if a particular Web site is correctly applying industrial best practices and has exercised due care in appropriately applying server-side countermeasures to current threats (e.g., via an audit). The PKI and participating certificate authorities are simply a framework for determining the binding between an entity's public key and its identity. The entities within the PKI state nothing about the quality or trustworthiness of the entity in question.

4.3. Trust and trust algebra

Trust relationships outlined in the previous sections are rather intuitive, but in order to make automated policy decisions, we need a way to quantify trust values. Furthermore, the decision must rely on information of varying certainty from sources of varying repute. For example, people are more likely to trust a well-known friend or a widely acknowledged authority than a friend of a friend. Therefore, as we approach the fringes of our trust chains, our uncertainty about neighboring *recommendations* increases, thereby reducing the trustworthiness of the proposition.

Jøsang discusses an algebra for quantifying trust [33]. This approach models an *opinion* as a vector of three dimensions: d, the disbelief in a certain binary proposition; b, the belief in a certain binary proposition; and u, the uncertainty concerning d and b. The algebra defined for the combination of values is grounded in a "framework for artificial reasoning known as *subjective logic*" [33]. We use trust algebra and concepts from subjective logic to combine individual opinions about a site's trustworthiness such that consensus views about the site's trustworthiness may be derived.

The subjective logic provides a framework for the combination and aggregation of the aforementioned opinions. Operations within this framework allow opinion combination based on *recommendation*, *consensus*, and *conjunction*. These operations are explained as follows:

Recommendation	Statement of belief concerning some proposition made from one agent to another, for example, a particular client's trust for a particular Web site based on the recommendation of a close friend.
Consensus	Statement of the aggregate collective belief across a community of agents concerning some proposition, for example, the collective community opinion concerning the trustworthiness of a given Web site.
Conjunction	A particular agent's overall opinion, given two different propositions, for example, the overall belief that a Web site is both trustworthy *and* meets content standards. Under certain conditions, this operation is equivalent to a logical AND.

The uncertainty component of an opinion model enables realistic *subjectivity*. It quantifies the confidence in the basis for a belief in a given statement, for example, "This Web site is not malicious." In fact, Jøsang stated that "Uncertainty is caused by the lack of evidence to support either belief or disbelief," [33]. That is, users may trust a Web site even though there is no reason to do so. In some cases, a user trusts a Web site just because he or she feels comfortable with the entity hosting it. Other than this arbitrary feeling, there may be no actual grounds or basis for the user's opinion.

Users are generally unaware of the Web site's policies and practices, which constitutes lack of evidence or uncertainty. We assume that such uncertainty can be reduced by review and audit conducted by a qualified third party. There is an issue of trusting the third-party auditor, that is, "watching the watchers," but we leave this as an open problem satisfying ourselves that some consortium could bless approved auditors.

Jøsang's model allows the expression of subjective aspects of trust to be quantified. More research could provide a better rationale for choosing uncertainty, the u parameter in Jøsang's model, but we suggest starting with the following intuitive values:

[0.30..1]	User believes that a Web site is trustworthy. The user can be uncertain, $u = 1$, but probably not more than 70% certain.

DOI: 10.1002/sec

[0] User has been attacked; Web site is *not* trustworthy.
[0..1] Value established by third-party auditor based on their assessment.

We have omitted cases when auditors have a pessimistic disbelief; we assume that the uncertainty value for audit applies toward a positive belief. We omit negative belief for audit because we think it rare that malicious sites would charter an audit. That is, trustworthy sites elect to be audited to be more marketable.

Given the correct mechanisms for enforcement, there could also be a place for self-rating. This concept has been around since the mid 1990s. In fact, the Platform for Internet Content Selection (PICS) [34] was the first metadata interchange standardized for such descriptions. However, PICS has been replaced by resource description format (RDF) and web ontology language (OWL) [35]. RDF can be used to describe various attributes of individual data items on a Web page, whereas OWL can be used to define a vocabulary and specify allowed relations between said entities.

4.4. Integrating the trust model

Virtualization mechanisms working inside browsers could "crowd-source" TRs for sites on the Web. As discussed in Section 3.3, a virtualized client can recover from an attack. Therefore, client-side attacks are allowed to execute in the virtual environment for the purpose of detecting malicious behavior. Multiple browser instances reporting a given site as malicious could help to provide the "consensus" needed to update a site's global TR. This crowd-sourced TR can then be published in the site's policy vector for consumption by yet other clients.

Recall that as discussed in Section 3.4, the OP browser implemented a provider policy. Coupling the notions of provider policy with TRs could reduce the chances that syndicated ad content could exploit weaknesses in the SOP that is commonplace in today's systems. The fact that the provider policy better isolates resources from one another prevents cross-domain access by virtue of the architectural structure itself. Moreover, if a given domain or IP address is known to be malicious, perhaps by a crowd-sourced rating, the browser may elect not to connect to that site in order to download content. The net result when viewing a modern Web page might be that portions of the page simply do not draw or draw a warning icon. The warning icon could be clicked by the user to provide an override so that maliciously classified page areas could be rendered on demand. Such an override would be welcome in cases where this technology is being debugged and many false positives result. Although such an override increases the user's exposure to danger, the use of an architecture like the OP browser would better confine attacks from malicious scripts—rather than having access to the whole page, malicious scripts would only be able to access resources from the same provider.

5. SUGGESTIONS FOR BETTER POLICY ENFORCEMENT

This section outlines a policy evaluation suggestions tailored for Web-based client applications, which facilitates *avoidance* of malicious Web sites. We describe a conceptual framework, which allows policies to be applied at various layers, each utilizing technologies of varying maturity. The final solution might utilize a highly distributed registry service like DNSSEC as a repository for site policy vectors (SPV), specifying a domain's community TR. With such an infrastructure in place, clients would not attempt connections to sites that fail to meet policy-specified rating requirements. Although some user override would be allowed, by default, the client would automatically avoid unwanted sites.

Similar services such as Web of Trust [30] and OpenDNS [31] are already online but have overhead or vulnerabilities that require a more thoroughly integrated solution. For example, Web of Trust contains an additional communication overhead introduced by their browser plugin as it checks back with the Web-of-Trust server to analyze link targets. OpenDNS comes much closer to our avoidance goals, allowing users to blacklist domains. On the other hand, OpenDNS is offered by a single entity and presents a single point of failure. It is also vulnerable to existing attacks on DNS because OpenDNS does not use DNSSEC.

The key difference in our proposed solution is the addition of an SPV. In the following paragraphs, we consider this vector added to DNSSEC. However, note that the addition of such a vector to DNSSEC is just one suggestion. Internet architects will likely balk at our abuse of DNSSEC in such a Web-centered view of name resolution. That is, the whole notion of adding the policy vector is for avoiding malicious Web sites. In this way, the avoidance strategy is centered on Web applications because of the mobile code that might cause the client to execute. Other services such as file transfer and media streaming might also make use of DNS(SEC) for name resolution.

The final solution could be integrated into DNSSEC by adding SPV information into the metadata records. This would allow flexible policy decisions as various solutions come into being. The possibilities for information expressed in the SPV site attributes are endless, but we seek to add a single binary measure; for example, the site is malicious, or the site is trustworthy. In fact, we could build on other techniques for content description and categorization, for example, OWL and RDF [35]. However, if our goal is to create a record format to be integrated into the DNS infrastructure, extensible markup language representations would introduce too much overhead.

Regardless of the encoding method, the essential information in the SPV consists of the triple {*type*, *TR*, *CT*}, representing type, trust rating, and third-party certification. The *type* could be one of several application

types such as entertainment, banking, and education. It allows the client to change policy requirements for *TR* depending on the *type* of site being visited, for example, requiring higher levels of *TR* and a *CT* for "banking" Web sites. The *CT* parameter indicates that the site has achieved a certification for correctly adhering to standards for self-description and best Web development practices. Note that the terms *TR* and *CT* may themselves be multidimensional. For example, *TR* could be expressed as $TR = T, R, CS$, where the values are trustworthiness, vendor reliability, and child safety.

Figure 1 shows an overview of security policy and its application at various layers in a pseudo Open Systems Interconnection (OSI) model We use the original OSI model because it offers a more detailed conceptual model for modern Web applications. For example, although from a pure networking standpoint, DNS is an application layer protocol, from a Web-centric view of the Internet, DNS is part of the networking infrastructure. Therefore, in our security model, the session layer contains Internet infrastructure technologies such as DNSSEC. By more clearly delineating Web technologies into these conceptual layers, we can more easily reason about the application of particular policies.

For example, the session layer, tasked with state and connection management, provides a point for content and connection-related policy enforcement. Driven by a DNS policy, the session layer could automatically prevent connections to sites that do not meet policy requirements for trustworthiness or allowed content. The presentation layer also provides a point for content-related policy enforcement. For example, the work of BEEP could be applied at this level to limit unauthorized mobile code execution.

We view DNS address resolution as a prerequisite operation upon which the rest of the Web-centric application software is layered. We assume that the higher-level dependence on session layer services is both temporal and logical. That is, in order to connect to a Web host specified in a URL, the client software must *first* resolve the DNS name to an IP address (a temporal relationship) and then is logically dependent on the resulting data read from the resolved IP address.

Given that the original intent behind the session layer was to manage connections, it makes sense to apply security-related decisions concerning the supposed quality of those connections. The session layer provides a point of policy enforcement by automatically preventing connections to sites that do not meet our requirements. In some cases, DNS will resolve to the local host address, preventing the user from accidentally visiting an unwanted site. Note that preventing name resolution is the technique currently used by OpenDNS [31]. However, our session layer scheme does not depend on any particular services but is instead integrated into the client communication stack.

More expressive constructs could be conceived specifically for Web applications running HTTP(S), where an HTTP HEAD request is used as a standard mechanism to retrieve self-describing page content. In our model, such mechanisms would work within the presentation layer. Ideally, it would apply higher-level policy constructs, especially for fine-grain control over content filtering. However, it could control connection policy as a stop-gap solution until the appropriate DNSSEC infrastructure is deployed.

Although allowing a great deal of flexibility, any self-describing mechanism is only applicable to those sites that wish to cooperate and provide metadata describing themselves and their content. To be truly meaningful for a secure browsing experience, any self-describing content would have to be certified. Without a third-party signature certifying the SPV, such a mechanism is applicable only to content ratings. The inclusion of a third-party certification is based on the assumptions that the client's HTML rendering engine does not execute mobile code when processing a response to a HEAD request and that the SPV provided by the site has been signed by a trusted third party.

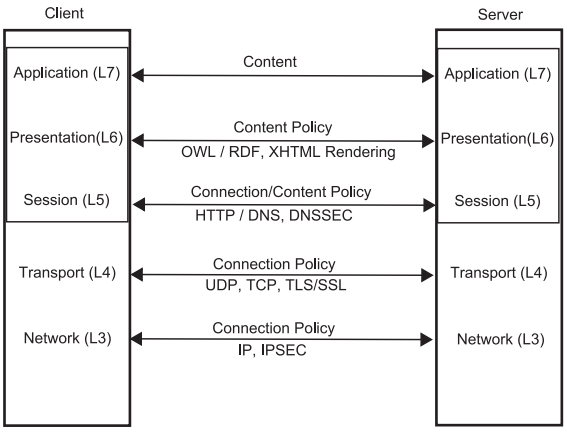

Figure 1. Layered policy. OWL, web ontology language; RDF, resource description format; XHTML, extensible hypertext markup language; HTTP, hypertext transfer protocol; DNS, domain name system; DNSSEC, secure DNS; UDP, user datagram protocol; TCP, transmission control protocol; TLS, transfer layer security; SSL, secure sockets layer; IP, internet protocol; IPSEC, secure IP.

6. CONCLUSIONS

This paper presented several types of Web threats and discussed various ongoing countermeasures in Web security from the client's perspective. Along the way, we discussed practical applications, interesting cognitive models, and forward-looking enforcement mechanisms. One of our more notable suggestions was that of applying BEEP to NoScript, as a solution to address XSS. As the pairing of NoScript and BEEP suggests, there are some fairly straightforward countermeasures, despite the fact that the Web security problem is widespread and growing.

With any countermeasure, it is important to remember that users are inherently bad at making security decisions. Considering human limitations is especially relevant in the context of Web clients. In fact, security is often viewed as a hindrance to users' immediate objectives as they surf the Web. Additionally, many existing controls are not well understood by the user community. If possible, we should strive to design systems that support the application of policy in a more automated fashion.

By embedding decisions into the policy framework, we believe that we can make security mechanisms less intrusive while complying with policy. We specifically considered avoidance as a strategy for dealing with security on the Web. However, such as strategy is not easily carried out because it depends on our notions of trust and the infrastructure required to support it. Community-based models provide assurance that such infrastructure goals are not out of reach.

Moving forward on the idea of trust, we described how trust algebra could be applied to establish TRs for various sites. Finally, we present a set of suggestions for a multilayer policy enforcement that could be used to prevent resolution of questionable site. We draw out Web security issues and provide some intuitive thoughts regarding a multilayer policy enforcement structure. Ultimately, we would like to extend DNSSEC to include SPV-based resource records and support avoidance strategies. We would also appreciate more discussion on our suggestions within the research community.

REFERENCES

1. Symantec. Symantec Internet Security Threat Report: Trends for January–June 07, September 2007. URL http://eval.symantec.com/mktginfo/enterprise/white_papers/ent-whitepaper_threat_report_xii_09_2007.en-us.pdf [accessed on 9 May 2008]
2. Bellovin SM, Benzel TV, Blakley B, *et al*. Information Assurance Technology Forecast 2008. *IEEE Security and Privacy* 2008; **6**(1): 16–23. doi: 10.1109/MSP.2008.13
3. Dormann W, Rafail J. Securing Your Web Browser, 2008. URL: http://www.us-cert.gov/reading_room/securing_browser [accessed on 1 June 2011]
4. Provos N, Mcnamee D, Mavrommatis P, Wang K, Modadugu N. The ghost in the browser: analysis of Web-based malware. *USENIX HotBots 07*, Cambridge, MA, 2007. URL http://www.usenix.org/events/hotbots07/tech/full_papers/provos/provos.pdf [accessed on 1 June 2011]
5. Provos N, Mavrommatis P, Rajab MA, Monrose F. All your iFRAMEs point to us, *Technical Report*. Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA, 2008. URL http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/archive/provos-2008a.pdf [accessed on 28 April 2008]
6. Rescorla E. Security holes… Who cares? In *SSYM'03: Proceedings of the 12th Conference on USENIX Security Symposium*. USENIX Association: Berkeley, CA, 2003; 75–90.
7. Ahmad D. The Contemporary Software Security Landscape. *IEEE Security and Privacy* 2007; **5**(3): 75–77. doi: 10.1109/MSP.2007.73
8. Ortiz S. Internet Researchers Look to Wipe the Slate Clean. *Computer* 2008; **41**(1): 12–16.
9. Hoglund G, McGraw G. Point/counterpoint. *IEEE Software* 2002; **19**(6): 56–59. doi: 10.1109/MS.2002.1049389
10. Hoglund G, McGraw G. *Exploiting Software: How to Break Code*. Addison-Wesley: Boston, MA, 2004; 1–23, 37–70.
11. Rubin AD, Geer DE. A survey of web security. *Computer* 1998; **31**(9): 34–41. doi: 10.1109/2.708448
12. Xia H, Brustoloni JC. Hardening web browsers against man-in-the-middle and eavesdropping attacks. In *WWW'05: Proceedings of the 14th International Conference on World Wide Web*. ACM Press: New York, NY, 2005; 489–498. doi: 10.1145/1060745.1060817
13. Johns M. On JavaScript malware and related threats. *Journal in Computer Virology* 2007; **4**(3): 161–178. doi: 10.1007/s11416-007-0076-7
14. Johns M, Winter J. Protecting the intranet against "JavaScript malware" and related attacks. *Lecture Notes in Computer Science*, vol. **4579**. Springer Berlin: Heidelberg, 2007; 40–59. doi: 10.1007/978-3-540-73614-13
15. Grier C, Tang S, King ST. Secure Web browsing with the OP Web browser. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, Oakland, CA. IEEE Computer Society: Washington, DC, 2008; 402–416. URL http://www.imchris.org/research/grier_sp08.pdf [accessed on 1 June 2011]
16. Rafail J. Cross-site Scripting Vulnerabilities, 2007. URL http://www.cert.org/archive/pdf/cross_site_scripting.pdf [accessed on 1 June 2011]

17. Chandramouli R, Rose S. Challenges in securing the domain name system. *IEEE Security and Privacy* 2006; **4**(1): 84–87. doi: 10.1109/MSP.2006.8
18. Avizienis A, Laprie JC, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 2004; **1**(1): 11–33. doi: 10.1109/TDSC.2004.2
19. Dhamija R, Tygar JD, Hearst M. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems, CHI '06*. ACM: New York, NY, 2006; 581–590. doi: 10.1145/1124772.1124861
20. Felten EW, Balfanz D, Dean D, Wallach DS. Web spoofing: an internet con game. *Technical Report Technical Report 540-96*, Department of Computer Science, Princeton University, West Windsor, NJ, 1997.
21. Ye Z, Smith S, Anthony D. Trusted paths for browsers. *ACM Transactions Information System Security* May 2005; **8**(2):153–186.
22. NoScript. 2008. http://noscript.net/ [accessed on 1 June 2011]
23. Tiwari S, Bansal R, Bansal D. Optimized client side solution for cross site scripting. In *ICON 2008. 16th IEEE International Conference on Networks*, New Delhi, December 2008; 1–4.
24. Jim T, Swamy N, Hicks M. Defeating script injection attacks with browser-enforced embedded policies. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*. ACM: New York, NY, 2007; 601–610. doi: 10.1145/1242572.1242654.
25. Yu D, Chander A, Islam N, Serikov I. JavaScript Instrumentation for Browser Security. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM: New York, NY, 2007; 237–249. doi: 10.1145/1190216.1190252.
26. Cox RS, Hansen JG, Gribble SD, Levy HM. A safety-oriented platform for Web applications. *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, Oakland CA, May 2006; 350–364. doi: 10.1109/SP.2006.4.
27. Ioannidis S, Bellovin SM. Building a secure Web browser. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, Boston, MA, June 2001; 127–134. URL http://www.cs.columbia.edu/~smb/papers/sub-browser.pdf [accessed on 1 June 2011]
28. Saltzer JH, Schroeder MD. The protection of information in computer systems. *Proceedings of the IEEE* 1975; **9**(63): 1278–1308. URL http://web.mit.edu/Saltzer/www/publications/protection/ [accessed on June 1, 2011]
29. Arends R, Austein R, Larson M, Massey. DNS Security Introduction. Internet RFC 4033. IETF March 2005. URL http://www.ietf.org/rfc/rfc4033.txt [accessed on 1 June 2011]
30. Web of Trust 2008. http://www.mywot.com/ [accessed on 1 June 2011]
31. OpenDNS 2008. http://www.opendns.com/ [accessed on 1 June 2011]
32. TRUSTe 2008. http://www.truste.org/about/index.php [accessed on 1 June 2011]
33. Jøsang A. An algebra for assessing trust in certification chains. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, San Diego, CA, 1999. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.1233
34. World Wide Web Consortium (W3C). Platform for Internet Content Selection (PICS), 1996. URL http://www.w3.org/PICS/ [accessed on 1 June 2011]
35. Daly J, Forgue MC, Hirakaw Y. World Wide Web consortium issues RDf and OWL recommendations—semantic web emerges as commercial-grade infrastructure for sharing data on the Web, 2004. URL http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ [accessed on 1 June 2011]