Web 2.0 Vulnerabilities

"A Thesis Presented in Partial Fulfillment of the Requirements for the Degree of Masters of Science in Forensic Computing, John Jay College of Criminal Justice of the City University of New York"

Mark Aaron Frankel

August 2008

<u>**Abstract**</u>

The World Wide Web has changed. Internet users no longer read static pages published by webmasters alone. Today, the nature of the Web is that of interaction and collaboration and has been dubbed the Web 2.0 phenomenon by some. Web 2.0 represents a social upgrade to the way that users have come to know the Web. Websites now allow users to add and change content, thereby making a site their own space. Unfortunately, the technology that makes this all possible is vulnerable to attack. The very freedom that defines the Web 2.0 movement flaws it as well. The problem should be fixed in a way that does not hinder the innovation that Web 2.0 allows. This work looks at the problem by examining the vulnerabilities, quantifying the severity of the threat and ultimately offering solutions – both existing and new.

Introduction

There is no short answer to the question "What is Web 2.0?" In its simplest form, it could be described as the interactive and collaborative World Wide Web. Web 2.0 is characterized by a user's ability to post content on the web. Examples of Web 2.0 services include Wikipedia, YouTube and the various social networking sites, which provide facilities to allow users to upload information - everything from text, graphics, video, etc. This phenomenon is built on top of various frameworks and executed on both servers and clients alike to give users a more enlightening experience. It represents the evolution from read-only, static pages to rich, dynamic content where the "Everyone" group has read, write and execute permissions.

This presents several fundamental problems from a security standpoint. Web 2.0 technologies open up avenues that circumvent many of the security measures that have been put in place, such as desktop firewalls. Web-based attacks occur over standard HTTP protocols by exploiting both the user's trust in a website, and the website's trust in its users. The weak link in the security chain also is less apparent. It's no longer only the uneducated end-user or the careless developer working quickly to meet a deadline.

This work examines Web 2.0 technologies and the vulnerabilities they create. We examine the empirical attack vectors such as Cross Site Scripting to see how the basic exploits occur. The combination of basic attacks with old ideas like social engineering allow for sophisticated and targeted attacks. We show how it's possible to expand Web 2.0 exploits to create large malware infrastructures similar to botnets. After reviewing possible exploitations, we describe past and present Web 2.0 attacks and contrast the differences between conventional attack vectors and the new attack vectors. Then we compare the resulting infection footprint between past and present infections. Finally, we discuss what can be done, what has been done and what should be done to protect Web 2.0 technology from being utilized for nefarious means.

Web 2.0 Defined

The term Web 2.0 caught on after the O'Reilly Media Web 2.0 conference in 2004, where CEO Tim O'Reilly used it to describe the Web as a platform (O'Reilly, 2005). Although Web 2.0 sounds like a technical upgrade of some sort, it actually expresses the social upgrade that the Web and its users have experienced. Web 2.0 illustrates the trend of business embracing the web as a global platform on which to portray its vision and expand its audience. The hallmark of Web 2.0 and its supporting technologies are active web content and collaboration. Users not only get to view the content a website serves, they are able to explain their own position and debate opposing opinions by basically uploading content to the site. Supporting Web 2.0 requires a range of technologies that allow users to post both static and active content to websites.

From the perspective of users who are familiar with the early iteration of the Web, it is hard to deny that it has gone through some changes. Wikis, blogs, mashups, syndications, social networking, web services, file-sharing, podcasts, rich Internet applications and folksonomies are all terms that had no meaning 10 years ago. Yet in the current state of our web-enabled society, a website cannot be successful if it does not employ some if not all of these technologies. Due to the popularity and success of these now common web features, advanced application programming interfaces (APIs) are making it easier for rich Internet applications to be developed, many of them based on interrelated development technologies known as Ajax. Ajax, or Asynchronous JavaScript and XML, simply refers to the ability of client-side applications to asynchronously transfer small amounts of data with the server in the background, allowing for a seamless user experience on the front end (Kyrnin). Such tools include animation engines, forum builders, and page customizers - all additions to an existing website that can add a new dimension of interactivity to enrich the user's experience. Many of these tools are low-cost or even open-source and distributed under the GNU General Public License (http://www.opensource.org/licenses/gpl-3.0.html). This collaboration between the Web 2.0-space and the open-source-space has enabled just about every wannabe webmaster to be the king or queen of his or her own next-generation website.

The advantages of this movement are apparent. Every new technology, however, comes with its own set of vulnerabilities. The main idea of the Web 2.0 movement is to promote interactivity between website and browser. The line between client and server becomes skewed in this model. The web used to be filled with static pages that users would view to obtain information. Now, the majority of sites allow users to upload pictures, submit reviews, post their opinions, message their friends and interact with the website. Unfortunately, user supplied data can become very hazardous to the website and the users it serves. In addition, every plug-in or extension utilized by the web server represents another application that needs to be updated or patched to protect against exploitation.

Web 2.0 Empirical Attacks – A Modern Hacker's Building Blocks

In the past, allowing users to submit content to the web server, either in the form of text or multimedia, has proved hazardous (SQL Injection, 2006; Teenage Hacker, 2006). Improper input filtering allowed SQL injection attacks to deface high traffic websites such as Microsoft's U.K. website on July 5th, 2005 (Leyden, 2005) and the United Nations website on August 12, 2007 (Keizer, 2007). Web 2.0 attacks, however, can produce far more devastating results. For example, in 2005, incomplete input filters employed by MySpace.com allowed a savvy user to unleash a worm that affected more than 1 million computers in 24 hours, an infection rate higher than any conventional worm before it (Lal, 2005). The so-called Samy worm exploited this vulnerability by executing a fundamental Web 2.0 exploit called a cross-site scripting attack. This worm was made possible because MySpace, a popular social-networking website, allows all users to customize a homepage and add content viewable by many others. The worm's perpetrator was able to find ways for his supplied input to circumvent the input filters.

Web 2.0 services provide novel, effective mechanisms for malware propagation that are contributing to a rapid rise in cyber criminal activity. Every year, security firms report an increase in online criminal activity with an emphasis on fraud. Traditional attacks such as phishing and Trojan horses have led the propagation of fraudulent attacks. Botnets, or large groups of infected machines under the control of an attacker, have enabled cyber criminals to propagate these attacks to large numbers of users at once. These attacks, however, have come in the form of spam sent from anonymous addresses or drive-by downloads used to install Botnet binaries, originating from questionable websites. Users have been educated not to trust content sent to them from unknown sources. Web 2.0 technology thrives on the idea of collaboration and trust. It allows cyber criminals to accomplish the same attacks, but under the guise of the trust model inherent in this collaborative Web. Phishing attempts sent from legitimate but vulnerable websites, or attachments from fellow members of social networking sites are harder to protect against because they lack the tell-tale signs vigilant users have been trained to look out for.

In the last 10 years, the size of the Internet user-base has increased 1000% since the popularity of the interactive web has caught on (http://www.zakon.org/robert/internet/timeline/). With social networking websites ranking in the Top 10 most popular sites globally (http://www.alexa.com/site/ds/top_sites? ts_mode=global&lang=none), it is safe to say that many of these new Internet users have bought into the changing face of the collaborative Web, thus allowing cyber criminals to exploit their trusting nature and reinvent malware propagation techniques. Because of the trust model employed by online communities, good old-fashioned social engineering may be all that is needed to propagate cyber fraud. Social engineering helps the cyber criminal unleash successful attacks on the large user-base of social networking sites like MySpace and Facebook. Once some virtual relationships are created in cyberspace between these criminals and their unsuspecting victims, social engineering is combined with cross-site scripting attacks to execute what is known as cross-site reference forgery. Cross-site scripting is an attack that enables a browser to overcome the same origin policy, a standard that prevents a client-side script from referencing content from a site other than the one it's browsing. It exploits the user's trust in a legitimate website. Cross-site reference forgery exploits a cross site scripting hole to ride over a user's existing authenticated session with a third-party site, normally to execute transactions against the site that benefit the attacker. This attack exploits a website's trust in a legitimate user. Both of these attack vectors are explained in more detail in the following sections.

As stated above, the benefits of a highly interactive web are clear. But along with it comes a different threat model. It is no longer enough to tell users not to open unknown email attachments nor reveal passwords to anyone. User education alone is no longer sufficient, as many Web 2.0 attacks can be executed without any user intervention. By delving into the many attacks that are now possible on the web, it will become apparent that everyone is to blame. Systems designers, developers and end-users need to understand the risks posed by Web 2.0 technologies and what can be done to mitigate those risks.

Cross-Site Scripting

In order to understand Web 2.0 threats, it is imperative to understand cross-site scripting (XSS[1]) vulnerabilities. Most threats against Web 2.0 technology are variations on the cross-site scripting model (Nadir, 2007). XSS vulnerabilities are very widespread and affect many web applications because they are both operating system and platform independent. Generally speaking, XSS is an injection attack. This threat becomes possible when a website accepts user-supplied input that is not properly filtered, allowing the attacker to store malicious code on the server. A browser visiting this site will execute this code with the permissions of the hosting site. Since active code such as HTML and XML is allowable on many of these sites, code within script tags may be run by the client browser's scripting engine, causing anything from a pop-up to a drive-by download such as a root-kit or Botnet binary.

[1] Cross-site scripting has a few different abbreviations in use, such as CSS or XSS. CSS is also the abbreviation for Cascading Style Sheets. This paper will use XSS exclusively to avoid any confusion.

There are two types of XSS attacks: reflected and stored. Reflected vulnerabilities, also known as Non-Persistent, are the most common XSS vulnerabilities. This attack utilizes a request that is made from a client's browser to a remote website and a response that immediately displays the results back to the client's browser. Stored or so-called Persistent XSS attacks enable more powerful exploits. This attack involves true HTML injection and forces the web server to store the malicious script on the site. Once stored on a web sever, the site will present the malicious code to all visitors to the affected page or pages. At first, it may be a little difficult to understand the potential for vulnerability. The following examples of Non-Persistent and Persistent XSS attacks will make the threat more apparent.

Type 0 – Non-Persistent XSS

The Non-Persistent, or Reflective, XSS attack is the more common attack vector. This is a one-time attack that is made from the client browser to a site that echoes user-supplied data back to the client. Search forms, for example, are a commonly exploited in this attack. When a user enters a search term in the textbox on Google's page, the results page displays the search hits along with a textbox already propagated with the previously entered search term. This means that the search term on Google's homepage was stored in some variable and then displayed back on a different page after the engine completed the search. Normally, this search term is plain text encoded in an HTML or XML statement that is rendered by the client's browser. But what if the submitted search term was entered as a script statement enclosed in HTML script tags? Most likely, the desired input would not need to be executable script code. Secure websites normally employ best practices for data validation whenever input is accepted from a user. When a site is vulnerable and allows unfiltered code to be passed back to the user, however, the client browser's scripting engine will execute the statement.

Google's site is not currently vulnerable to this sort of attack. Such a high-profile search portal has many highly trained professionals reviewing the code, ensuring that secure development practices are always in use. However, this feature of echoing back user-supplied data is common on websites, as many of them feature bulletin-boards, forums, guestbooks and search engines. In the Web 2.0 environment, most sites allow HTML to be submitted rather than plaintext to allow rich content, animated icons and text effects to be displayed.

The following is a simple method that malicious users employ to test whether or not a website is vulnerable. This method is a good example of a reflective XSS attack. Let say www.weaksite.com is a website that hosts a forum for a discussion group for heavy metal concert reviews. At the top of the page there is a search box that allows users to enter a band's name to search for that band's specific reviews. If the term "Metallica" is entered, search hits are returned. At the top of the results page, another search box already has the previous search term "Metallica" in it. Again, this means that the search term was stored in a variable and the value of this variable was displayed back to the client browser on the next page. If instead of plaintext, "<SCRIPT>alert('xss');</SCRIPT>" was entered, this statement would then be presented back to the client browser. The browser would hand this statement over to its scripting engine, which would in turn pop up a message box with 'xss' displayed in it. Obviously this is a harmless result, but it effectively shows that www.weaksite.com has an XSS hole.

Reflective attacks are usually seen in the wild in the form of links that users click on, with the assistance of social engineering. The attacker would find a legitimate site that has a XSS vulnerability, construct a link that exploits this XSS hole and send it out in spam email, instant messages or forum posts. Users will be more inclined to click on such links because the URL will contain the name of a legitimate website. Perhaps an attacker wants to steal a user's browser cookies. If the attacker knew that Chase Bank's website had a hole exploitable by a Non-Persistent XSS attack on the site's search page, the attacker could send out a spam email with the Chase Bank logo and a link. The link would display a URL that is to a legitimate search page on the actual Chase Bank website. The URL, however, would be constructed so that instead of a search term, a malicious script is submitted to the vulnerable page, and echoed back to the user on the results page. Any scriptable action will then be executed by the victim's browser, including such actions as emailing the cookies to the attacker.

Type 1- Persistent XSS

Persistent XSS attacks are much more dangerous than Non-Persistent attacks. With Persistent attacks, the malicious code is actually stored on the web server. As previously mentioned, Reflective XSS attacks seek immediate results, but only one time. Stored XSS attacks may lie dormant on a web server unnoticed for long periods of time and can infect every client that visits the page. If the code is propagated with each infection, the potential for worm-like behavior exists. This infection could continue to spread for as long as the injected code remains undiscovered. Perimeter security measures like firewalls, which are successful against conventional worms propagating themselves over arbitrary TCP ports, fail to help in this situation. Web-based attacks such as this use standard HTTP protocol messages over port 80 for their transport. Most firewalls permit the initial HTTP message requests to the website and also allow the response messages to pass through unhindered.

Stored XSS attacks can manifest themselves in a number of ways. Script code could be stored inside a database using some SQL injection exploit. This attack would infect any user that executes a query that returns the tainted record. More commonly however, malicious code would be posted to a message board, or perhaps in a user's profile on a networking site. Anyone simply browsing the forum would execute the active code in their browser.

The desired result of a Persistent XSS attack is most likely different than that of a Non-Persistent attack. As Reflective attacks are more instantaneous in nature, the result tends to be session hijacking or website defacement. With Persistent XSS attacks, the results are much more interesting and potentially dangerous. If an attacker successfully exploits a XSS hole and stores code on a popular website, it could potentially infect many more users than conventional malware. Since XSS attacks occur in the browser, and usually in the background, infected users may not even know of their infection. Depending on the nature of the delivered payload, there doesn't even need to be any lasting effects on the client. If no malware needs to be installed on the client and the actual attack happens in the background, there may be few or no obvious effects. Thus an exploited XSS hole may go unnoticed for much longer than a conventional breach.

Unfortunately, more XSS vulnerabilities are discovered everyday. A good resource for Web 2.0 vulnerabilities is the SecurityFocus BugTraq mailing list (http://www.securityfocus.com/archive/1). In June 2008 alone, the list disclosed 13 XSS vulnerabilities. These disclosures usually include information such as the vulnerable website or application, the page the hole is in on, and the variable that the script code uses. The list details the nature of the XSS and type of harm it causes, for example, Reflective or Stored XSS page defacement or redirection exploit.

An example of a recent attack occurred on June 5th, 2008. A disclosure was posted to a mailing list warning about an application called WEBAlbum (2008). This application enables a webmaster to allow the website's users to store, display and share photo albums with other users. Version 2.0 of this application has an XSS vulnerability on the "Add Comment" page at http://[website]/[webalbum_path]/photo_add-c.php. This page allows users to submit a comment about a posted photo. The "comment" variable is not properly scrubbed for unwanted characters and allows an attacker to submit script code. According to the disclosure, Stored XSS and defacement attacks are possible on any website that employs this version of WEBAlbum.

Additionally, on June 17th, 2008 a disclosure about an application called OpenDocMan version 1.2.5 was posted to the list (OpenDocMan, 2008). OpenDocMan is an open-source document management system (DMS) that allows web-based access. Any website that employs such a tool must keep the contents of the DMS organized and secure. Any tampering with the stored documents could prove devastating to the application's owner. This disclosure describes the vulnerable webpage and the method used to discover the hole. The "last_message" variable at the URL http://[website]/opendocman_path]/out.php?last_message=%3Cscript%3Ealert(document.cookie)%3C/script%3E is not properly filtered for unwanted characters. Here, the input filters should not allow URL encodings as input for a variable that only expects text. The URL encodings for < and > are %3C and %3E, respectively. When the browser encodes the value, the last_message variable contains the value “<script>alert(document.cookie)</script>”, which causes the script-rendering engine in the browser to display a pop-up message box that contains the cookie for the current session. Showing in this manner that a XSS hole exists proves that any action allowable in this application can be run on the authenticated user’s behalf, including mangling or deletion of documents.

Web-Based Attacks

Now that we have introduced the attack building blocks that Web 2.0 vulnerabilities provide, we can examine more complex attack vectors. These attacks involve using XSS along with other methods of exploitation. Cross-site request forgery is a method that expands on the ordinary XSS hole. It is most effective when used against websites that are known to have an XSS vulnerability and employ poor authentication methods. The section on Puppetnets takes a look at how Web 2.0 vulnerabilities can be used to build complex attack infrastructures capable of controlling large numbers of victim browsers. Using such techniques, one can build attack infrastructures that rival the size of modern Botnets.

Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF) vulnerabilities are among the most dangerous attack vectors that threaten current web applications (Jackson-Higgins, 2007). These vulnerabilities allow an attacker to execute commands as a trusted user. A CSRF attack is similar to an XSS attack but with some fundamental differences. XSS attacks involve code that is sent from the website to the user and executes in the client's browser. Since most users tend to trust the website they are visiting, anything that executes in the client browser is also trusted. Thus XSS attacks exploit the user's trust in a website.

Since CSRF attacks can cause unauthorized transactions to be executed against a web application on a trusted user's behalf, CSRF attacks exploit the site's trust in its users. CSRF attacks are more difficult to carry out than XSS type attacks because a successful execution depends on several factors. First, the victim must be logged into the target website at the time of attack. Second, the site has to employ persistent cookies in its authentication method. Nonetheless, CSRF attacks are becoming increasingly common and costly given the widespread use of social networking sites and other Internet services that facilitate interaction among users (Huntington, 2007; Jackson-Higgins, 2006). Recently researchers have begun to identify CSRF vulnerabilities in application software widely used to build enterprise level web portals (CERT, 2008).

There are several ways to tell if a website is vulnerable. If actions against the website can be executed multiple times using static URLs, it is most likely vulnerable to a CSRF attack. As an example, if a banking site was to accept the URL 'http://bank.com/acount_action.php?action=transfer&source=victim&dest=attacker&amount=10000/' and execute the transaction continuously when the URL is entered repeatedly, a CSRF vulnerability most likely exists. Websites that allow transactions to occur without reauthentication are dangerous as well. Every site that executes transactions should require some form of user interaction every time a new transaction is requested. At the very least, the user should be required to approve each individual transaction. Ideally, valid credentials should be re-entered every time.

Like XSS, CSRF is not a new attack vector. In 1988, a paper was published by Norm Hardy describing an application trust level vulnerability that he called the Confused Deputy (Handy, 1988). Twelve years later, a popular website was discovered to have a confused deputy vulnerability (Auger, 2008). This discovery was published to the SecurityFocus Bugtraq mailing list (http://www.securityfocus.com/archive/1). At that time, more researchers and developers started looking for this specific security hole. In 2001, Peter Watkins coined the term Cross-Site Request Forgery[2] in response to a debate about the dangers of posting images to forums (Watkins, 2001).

[2] Cross-Site Request Forgery is also called Cross-Site Reference Forgery, XSRF and CSRF (pronounced C-Surf)

CSRF attacks can be carried out in a number of ways. Most commonly, HTML image references will point to specially constructed links for actions against an authenticated website. When the malicious page is viewed, the browser will attempt to load the images. Regardless of whether the image reference points to an actual image or not, the link is followed. If the victim is authenticated to the targeted site, the action will be carried out under the context of the trusted user.

What makes CSRF attacks so dangerous is the ease with which it can be carried out. Although the execution may be trivial, the difficulty lies in finding the right victim. Therefore, this attack will most likely be a targeted attack like spear-phishing, rather than a blind attack like reflective XSS. Using social engineering, the attacker will not only need to find out that the victim has an account on a vulnerable site, but that the victim is authenticated against the site at the time of attack.

The following example will show how this attack could be executed in the wild. Suppose Alice trades stocks on hypothetical website www.ezstocks.com. This site is known to use persistent cookies to maintain a trusted relationship between Alice and the site while she is logged in. Alice also enjoys keeping in contact with her friends and colleagues on www.keepintouch.com, a social networking online community. Bob is an online con-man who developed an evolved type of pump-and-dump scheme. Contrary to normal pump-and-dump schemes where spam that encourages users to purchase an up-and-coming penny stock is blindly sent out to a long list of known email addresses and instant messenger screen-names, Bob looks for people on social networking sites that belong to stock trading websites with known XSS vulnerabilities. Bob, also belonging to www.keepintouch.com, befriends Alice after noticing that her profile mentions that she likes to trade stocks online. In a short time, Bob learns that Alice trades using www.ezstocks.com, a site that Bob knows is vulnerable to CSRF attacks.

While Bob is talking to Alice, she mentions that she is currently trading stocks. Since Bob knows that at this time Alice is most likely authenticated to the stock trading website, perhaps logged into the site in another browser window, he just needs to get Alice to load some malicious code. Bob can send Alice an email that supposedly contains pictures of his kids. The HTML code for the pictures may look something like <img src="http://www.ezstocks.com/trade.php?action=buy&stock=FRGT&shares=10000">
If Alice is authenticated against the ezstocks website, which allows multiple transactions without reauthentication, and this static URL is a valid command on the trading page of that site, Alice would have just bought 10,000 shares of the stock FRGT by simply viewing Bob's malicious email. If Bob and four of his colleagues each were able to dupe 10 people into clicking on similar links, these unsuspecting traders would buy 500,000 shares of FRGT. These purchases could easily drive up the share price of a thinly traded stock, and the fraudsters would then dump their shares for a sizable profit.

Puppetnets

Attack vectors such as CSRF attacks are of special interest because they combine different empirical hacks, such as XSS and social engineering, to execute a more powerful and targeted attack. Perhaps the most intriguing attack infrastructure currently deployed is the botnet, an army of zombie computers that can be made to collectively execute a number of malicious activities while under the command and control of a so-called botmaster (Caravan, 2005, pgs. 104-114). Compromised machines called bots are provided with instructions to "report for duty". The most common way used today is through a private IRC channel that the botmaster sets up, though there is evidence to suggest that this model is fading, with more botnets using peer-to-peer techniques or even RSS feeds to communicate (Jackson-Higgins, 2007; Grizzard, Sharma, Nunnery, Kang and Dagon, 2007). Once the bot calls home, it is given one of a number of possible tasks that it and various other bots would carry out. Such tasks include attacks, administrative control or reconnaissance in the form of looking for more potential bots.

Researchers have identified botnets with more than 1.5 million controlled hosts (Keizer, 2005). With infrastructures this large, massive DDoS attacks are possible, such as the attack against a Tokyo company in December 2007. The attack rendered the company's site inaccessible for more than a week while the attackers sought a ransom. During this DDoS attack the botnet sent traffic to the company's web site at a rate of 6 Gbps (Tanaka, 2008).

Botnets currently are the leading cause for concern in the security community. I believe, however, that as operating systems become easier to patch for end users, fewer vulnerable hosts will exist for botmasters to control. The ability to infect large numbers of machines is a fundamental requirement for a botnet's existence. I anticipate attack infrastructures similar to botnets will regain momentum once attackers begin to use the techniques described in this paper and take advantage of the fact that Web 2.0 vulnerabilities are exploitable with little or no user intervention.

Web 2.0 vulnerabilities have made it possible for an uninfected machine to be part of a distributed attack infrastructure. This phenomenon was researched in an interesting paper (Lam, Antonatos, Akritidis and Anagnostakis, 2005). The term puppetnet, coined by the paper's authors, is used to describe an infrastructure similar to a botnet. The puppetnet idea is similar in that a large number of zombie browsers are instructed to execute a specified attack, but no lasting control is held over the “puppets.” Instead, an XSS hole is exploited on a website and all browsers that visit the site and view the infected page execute malicious code that instructs the puppet what to do. By exploiting this behavior, a website can effectively take control of a collection of visiting web browsers and turn them into a distributed attack infrastructure.

As previously mentioned, puppetnets depend on websites to take control of visiting browsers. These websites can either be legitimate sites that had harmful code injected into them, or hacker owned sites that receive a lot of traffic due to the offer of enticing services, such as free online storage, file hosting, file sharing or free downloads. There are also a number of sites that invite surfers to participate in vigilante operations. An alarming characteristic about these attacks is that while a user is visiting an otherwise innocuous site, an attack can be launched from the visitor’s browser without any indication, software to be installed or interaction from the user.

Thus far puppetnets seem to be very similar to the botnets. Although they are both in control of distributed armies of unwilling victim computers that can participate in malicious activities such as DDoS and worm propagation, there are three fundamental differences between the two phenomena. First, botnets rely on some software implementation flaw to be exploited and allow the unauthorized installation of some malicious payload. Puppetnets look to exploit functional vulnerabilities of web browsers, their plug-ins and rendering engines. They take advantage of mechanisms that enable dynamic web content and cooperation between different software implementations. Second, botnets keep complete control of their bots when they are online and can command them to do anything implemented in the bot code. Puppets are somewhat restricted in that they can only perform activities that are allowed by the browser. Therefore, puppetnets are not in complete control of the puppets. Lastly, participants in an ongoing distributed attack are a dynamic group. Since there is no software installed on the victim, there is no guarantee that this victim will ever fall prey again. The participants are users browsing an infected site at the time of the attack. This moving target makes puppetnets extremely difficult to defend against since there is nothing to track and no puppetmaster directly involved at the time of the attack. Puppets tend to be short-lived; essentially only active as long as they are still viewing the infected site. This dynamic nature also makes it easy for an attacker to always have an ample amount of participants, directly proportionate to the relative popularity of the infected website.

Unlike botnets, administrators of web clients and hosts involved in a puppetnet feel no sense of urgency to mitigate the problem. On the client's side, the attack is almost completely transparent. Even if the user notices a problem, such as a loss in performance due to a decrease in bandwidth, the problem goes away as soon as the browser is closed. From the server's perspective, the injected malicious code usually has no adverse effect on the site itself, unlike DoS attacks or defacement. Moreover, the puppetnet attack takes place against some third party, rather than the server or client involved. Thus there is little alarm regarding the threat and little urgency to try to fix the problem, even after the infection is discovered.

The study (Lam et al., 2006) looked into the threat assessment of such an infrastructure. They analyzed the possible effectiveness of a puppetnet to execute different attack vectors, such as distributed denial of service and distributed computations such as hash cracking. The following is a summarization of the findings.

As mentioned earlier, when an image is embedded in a website, the HTML code contains the location of the stored image. This location, specified as the value of the ‘img src’ variable, can be on another website altogether. Thus, a successful DDoS attack can be carried out if the compromised website forces a large number of visiting browsers to simultaneously make requests for an image on the victim’s site.

There are small nuances in the way browsers handle object requests that hackers must circumvent in order to maximize the traffic generated by the puppets. For example, Internet Explorer and Mozilla Firefox follow the HTTP 1.1 specification, which only allows two simultaneous connections to the same server (Breen, 2006; Fielding, Gettys, Mogul, Frystyk and Berners-Lee, 1997, sec. 8.1.4). This limitation can be overcome by using variations of the target URL. Replacing the DNS host name of the target web server with its IP address registers as a totally different connection. Adding a trailing period at the end of the URL or stripping out the ‘www’ also counts as a separate connection. Web servers that employ virtual hosting are especially vulnerable. Virtual hosting is the method used by many web servers to host multiple websites on the same server; essentially creating multiple DNS records for the same IP address. Attackers can also increase the stealthy nature of this exploit by using browser frames of size 0 to launch the attack in the background.

Potential Puppetnet Size

Researchers (Lam et al 2006) consider puppetnets to be a dangerous threat to the safety of the Internet due to their potential size. After all, one of the most menacing qualities of the botnet phenomenon is the large number of infected machines they can control. To determine the possible size of a puppetnet, researchers analyzed results from web statistics tools and reports. Many sites use tools such as Webalyzer (http://www.mrunix.net/webalizer/) and WebTrends (http://www.webtrends.com/) to establish site usage statistics, most of which can be located using any search engine. For example, audit companies like ABC Electronic (www.abce.org.uk/) offer their datasets to the public. Alexa (www.alexa.com/) supplies a client side toolbar that collects usage patterns and supplies statistics about, among other things, the most popular websites.

From the compiled statistics, it is possible to estimate puppetnet size from daily hits and typical viewing times of websites. The results are staggering. The most important observation is that puppetnets can reach sizes comparable to botnets. Top-100 sites can create puppetnets controlling 100,000 browsers at a time. The size of the largest potential puppetnets, from a top-5 infected site, could reach up to 2 million controlled puppet browsers simultaneously (Lam et al., 2006). Admittedly, the more popular the site, the more secure it will be. Occasionally, however, even very popular sites can have major vulnerabilities, such as we will see with MySpace.

A top-100 website is not necessarily needed to cause significant damage. A less popular site can generate enough traffic to control at least 1,000 puppet browsers simultaneously. Consider the following attack. Recently, both ICANN.org and IANA.org were defaced by Turkish hackers (Almeida and Fernandez, 2008). The Internet Corporation for Assigned Names and Numbers (ICANN) is responsible for the uniqueness of the Internet's domain names. The Internet Assigned Numbers Authority (IANA) maintains global coordination of IP address and DNS records.

Although when asked the group declined to disclose their methods of infiltration, web-based attacks like XSS are most likely the vectors used.

According to Alexa, ICANN was visited by 0.0078% of all global Internet users that day (http://www.alexa.com/data/details/traffic_details/icann.org), while IANA was visited by 0.006% of all global Internet users (http://www.alexa.com/data/details/traffic_details/iana.org). The Internet World Stats site (http://www.internetworldstats.com/stats.htm) measures the average global usage of the Internet per day to be 1,407,724,920 users. This means that combined, the two sites had about 194,267 visitors that day. If the attackers had more nefarious intentions, these compromised sites could have been the launching ground for attacks that could cause serious damage. Although the number of visitors per day to these specific sites would not likely be capable of bringing down a major website with a DDoS attack, other attacks, such as cookie stealing or drive-by downloads, would be possible.

<u>Vulnerabilities in Web 2.0 Technologies</u>

The trend toward active interaction between users and websites has spawned many online communities that center on this idea of interactivity, i.e., giving users the power to add and change content on websites. It is no secret that Web 2.0 has been embraced by the user community. Web surfers demand the ability to make visited websites their own by customizing homepages, uploading pictures and sharing a wide range of active content.   The large number of websites that employ a myriad of Web 2.0 technologies comes as no surprise. Social networking sites like MySpace.com and Facebook.com offer forums and content posting by their users. Flickr.com and ShutterFly.com focus on users sharing their pictures with others to see. Wikipedia.com allows anyone to edit encyclopedia entries. Even smaller sites like company intranets promote the use of wikis, or collections of documents that allow viewers to update and change the contents during project discussions. Many providers offer web services to be used in conjunction with other offerings to enrich the user's experience. The majority of sites syndicate their content to be used with feed readers so that users can keep current on with the large number of topics that interest them.

Anyone that uses the web today has heard of and used many of the aforementioned Web 2.0 technologies. Not many, however, realize the potential vulnerabilities or what is at risk when deploying or using these rich features. Both web clients and servers have a lot to gain by employing these features and just as much to lose if the right safeguards aren't taken. We will look at some of these technologies to see what's at risk, as well as actual exploits that have occurred. It's important to keep in mind that although most Web 2.0 exploits stem from XSS holes that lead to larger attacks, older more conventional attacks, such as social networking and drive-by downloads, exploit these technologies as well.

XSS Worms

Social networking sites like MySpace.com, Facebook.com, Friendster.com and hundreds of others offer many of the features that are highly in demand on the web today. MySpace alone has over 110 million subscribers worldwide (Swartz, 2008, para. 10). In fact, as of June 2008, 9 out of the 30 most popular websites are social networking sites (http://www.alexa.com/site/ds/top_sites?cc=US&ts_mode=country&lang=none). This large user base is able to customize a homepage by changing backgrounds, posting pictures as well as by embedding music, videos, and links, many of which load instantly. Although homepages can be viewed by both members and non-members, most content is available only to registered users. Members can form connected social groups by sending invitations to other users. After invitations are accepted, the invitee is added to the inviter's Friend List. Friends can leave each other messages on their homepages, essentially forming mini forums on each page. Since messages are uploaded as HTML, posters enrich their messages with active content like animations, links and embedded objects. Even the least tech-savvy users have become surprisingly talented HTML editors.

After discussing XSS exploits, it's obvious that sites like MySpace.com are prime candidates for Web 2.0 vulnerabilities. Being the 6th most popular website worldwide (http://www.alexa.com/site/ds/top_sites?ts_mode=global&lang=none), many assume that most of the security holes have been plugged. No site, however, is totally secure, and in a site with such a large user population vulnerabilities are highly leveraged. MySpace has been hacked in the past due to poor input validation routines that resulted in XSS holes. Since a submitted post to a profile is stored on MySpace's servers and displayed to any visiting user, poor input validation would allow a stored XSS attack. The nature of the submitted malicious post would define the attack. If the payload was harmless, the attack would be annoying but ultimately insignificant. If the payload was outright malicious, for example self-replicating code or cookie stealing attacks, the result could be devastating. The following will describe two MySpace attacks, one on the tamer side and the other being serious in nature.

The Samy worm is credited with being the first Cross-Site Scripting worm to be released in the wild (Mook, 2005, para. 2). Its author, Samy Kamkar, is a programmer from the Los Angeles area. His motivation was simply to secure some online celebrity status amongst his friends. At the time of its release, the Samy worm was the fastest growing worm ever and infected over 1 million hosts in 24 hours (Mook, 2008, para. 2). The payload was harmless and only caused the phrase "and most of all, samy is my hero" to be appended to the profile of an infected host. It also would create a friend request from the target to Samy's account, an action that usually requires authorization but was done automatically.

The security hole that was exploited to allow the code to be posted on Samy's profile was not straightforward. MySpace has always employed a sophisticated set of input filtering on all entries. The hole existed because of weaknesses in MySpace's input validation strings and the client's browsers. Samy discovered that terms like <SCRIPT>, <BODY>, 'onClick' and 'javascript' were stripped from anywhere in the input. The filters didn't pick up variations of some terms. For instance, 'javascript' was filtered out but java\nscript ('\n' is the escape character meaning 'newline') was not. Browsers like Internet Explorer and Firefox would interpret 'java\nscript' as 'javascript'. MySpace would not allow javascript to be included in straight HTML in <SCRIPT> tags, but browsers would allow javascript to be executed within Cascading Style Sheets tags. So, <SCRIPT>"alert(1)"</SCRIPT> would not be allowed, but <div style="background:url('java\nscript:alert(1)')"> was valid. The full exploit code was written in HTML, using embedded JavaScript in <div> tags and Ajax XMLHttpRequest objects. Other roadblocks that were overcome included limits on characters for entries, alternate encoding tricks and reusable functions. The full source code of the actual exploit is available at http://namb.la/popular/tech.html. It was posted by Samy himself after the security breach was patched on MySpace's end.

The Samy worm was released on October 4th, 2005 at 12:34PM. Within 24 hours, Samy had received 1,005,831 friend requests, 1/35th of MySpace's active members at the time, according to a recount by the worm's author himself, which is archived at http://web.archive.org/web/20060208090906/namb.la/popular/. Samy first posted the code on his profile. Every visitor to his page would request him as a friend and add him as one of their heroes. The phrase "but most of all, samy is my hero" would now appear in the visitor's profile as well. The visitor's homepage also would contain the code so everyone that viewed this newly hacked page would also get infected in the same manner.

This worm, also known as JS.Spacehero, eclipsed the infection rate of previous worms that exploited more conventional vulnerabilities, such as unpatched operating systems. In comparison, Code Red infected 359,000 hosts, Slammer 55,000 hosts, and Blaster 336,000 hosts within the first 24 hours (Wasserman and Su, 2008, Table 1). The main difference between past worms and Samy occurs after the first 24 hours. After discovering the vulnerability, MySpace brought down its entire network to fix the issue. Once the patches were applied to the input filters and the profiles were scrubbed clean of the exploit code, the worm's propagation was terminated. With the other worms, the vulnerabilities existed solely in the clients. The software vendors released patches to their buggy software, but hosts were not safe until the users applied the patches themselves. The minimal effort taken to mitigate the result of the Samy worm should not suggest that Web 2.0 vulnerabilities are benign in nature. The threats may seem simple to prevent, and we shall see that to be true. These security holes, however, are easy to exploit once a website is seen to contain an XSS flaw. Moreover, Web 2.0 attacks can quickly affect vast number of users on a site like MySpace or Facebook and then can take advantage of the extraordinary computational power available on these sites for nefarious activities.

On January 31st, 2007, Samy Kamkar plead guilty in Los Angeles Superior Court to a felony charge in violation of California State Penal Code section 502(c)(8) (Mann, 2007, para. 2), which states that any person that knowingly introduces any computer contaminant into any computer, computer system, or computer network has committed a public offense. He was sentenced to three years of formal probation, ordered to perform 90 days of community service, pay restitution to MySpace, and had computer restrictions placed on the manner and means he could use a computer.

Although the longevity of the attack seems to depend largely on how long it takes a host site to discover the hole, XSS worms still have the potential to be just as devastating as traditional worms, if not more so. What if Samy had malicious intentions? If his included payload attempted to download Trojan malware, steal cookies or instruct browsers to participate in Puppetnet attacks such as site penetration testing or DDoS, the attack would have infected one million hosts, a huge surface area. Another MySpace worm that was released in 2006 did have such a malicious intent and resulted in many MySpace users having their cookies stolen.

Browser Plug-ins Attacked

Over the July 15th, 2006 weekend, a worm was released on MySpace that exploited a vulnerability in the Adobe Flash Player (Chien, 2006). Adobe had announced the vulnerability the week before and advised all users to upgrade to version 9 of their player. Since this worm was released deliberately with malicious intent, researchers did not have the luxury of a fully disclosed explanation from the worm's author. It seems, however, that MySpace's vulnerability was that it allowed users to embed .swf (Adobe Flash video) files regardless of the code included. Flash animations, unlike many other video formats, allow actions called Action Scripting. The included actionscript was used to hijack the email address and password from the infected user's cookie and store them on a different site.

Two exploited vulnerabilities made the SWF MySpace attack possible: an XSS hole in MySpace's input filters, and an undesirable side effect of the Adobe Flash Player browser plug-in. The XSS hole inherent on MySpace was not so different from the hole that allowed the Samy worm to propagate. Although the previous weakness was patched and the input filters hardened against such an attack, MySpace still allows undesirable scripts to run when executed by an embedded object.

The weakness in the Adobe Flash Player extension, however, was the primary culprit. Adobe SWF objects allow embedded script code to be executed by the browser upon load. This actionscript occurs automatically in the background. The actionscript used in this hack loaded another actionscript that was located on another page on MySpace, which in turn loaded another. The three scripts ran recursively in the context of the victim's MySpace profile, resulting in the victim's email address and password being gleaned from the browser's cookies. These valuable pieces of information were redirected to a blog post that the attacker controlled. Adobe corrected this undesirable side effect in its Flash Player version 9 by disallowing untrusted scripts from automatically loading any URL in the victim's browser. Unfortunately, most users did not update their browser plug-ins by the time the attack was unleashed.

This presents another facet of Web 2.0 security concerns. All the dynamic and interactive content that Web 2.0 technologies like Ajax have enabled needs to be rendered on the client's machine. Many of these rendering engines come in the form of browser plug-ins. These plug-ins, like the browsers and operating systems they run in, need to be kept up-to-date with patches to prevent against known exploits. The way Ajax uses Javascript, extensively and in complex ways, creates many issues. The client-side processing of asynchronous events allows the requests to be sniffed and gives attackers the opportunity to inject code and propagate XSS and CSRF attacks (Di Paola and Fedon, 2006).

<u>**Mitigation – What Can Be Done?**</u>

Thus far, I have talked about the attack surface that Web 2.0 technology presents to the hacker community. We have seen where the vulnerabilities arise, the empirical attack vectors used by hackers, how these exploits are used to create malicious infrastructures and the attacks that are occurring everyday in the wild. But just like the malware that has existed in the past, there are safeguards that can both prevent and eventually eliminate this new wave of threats.

There are many elements that contribute to the overall attack surface and each one needs to be hardened. In addition to security practices targeted at the threats presented by Web 2.0, standard security practices are as important as ever. End-users should surf the Internet with fully-patched systems and updated browser add-ons. Anti-virus and anti-spyware software must be deployed and up-to-date. System administrators charged with protecting the web servers must deploy up-to-date patching techniques across the all data center servers. In addition to the standard practices, network security gurus must now employ perimeter appliances that monitor both ingress and egress traffic. As mentioned earlier, conventional firewalls are not very effective against web-based attacks because firewalls generally allow standard HTTP traffic through. Monitoring ingress and egress traffic for unusual activity, e.g., frequent communications with sites not normally visited by the organization, can help administrators determine if the organization's network resources are being used for unauthorized purposes. Lastly, application developers and programmers are possibly the most important link in the chain. They are expected to build applications to specifications under a deadline and at the same time guarantee the security of their products. All too often, getting the project done on time wins over getting it done right.

In the following sections, we will look at preventive methods for end-users, system administrators, network security professionals and application developers. Web surfers need to follow safe browsing techniques, as well as utilize tools that can give guidance for relative risk factors. System administrators should proactively ensure that web servers are fully patched, while providing their own users a safe browsing environment. Many times, system administrators must save the users from themselves. Network security personnel work alongside the system administrators monitoring the traffic that is flowing in and out of the web servers. They can make use of gateway appliances built specifically to identify Web 2.0 attacks on-the-fly. Developers need to tune their programming processes to be aware of the threats that developing Web 2.0 applications present. They should realize the importance of secure programming by understanding the risk and staying up-to-date on the web-based threat landscape.

Protection for End-Users

When it comes to protecting the end-user, Web 2.0 vulnerabilities are very different from past security concerns. Web-based threats do not discriminate against the client's operating system or browser. It is no longer sufficient for users just to be leery of unknown downloads and email attachments. Merely surfing trusted websites with an XSS vulnerability is enough to invite infection. Unfortunately, the end-user has no protection from a trusted website if such a hole exists. There is no client-side detection mechanism in use today that can warn users if their banking site, for example, is suspect.

There are, however, best practices that can lessen the possibility of falling prey to web-based attacks. Some present tradeoffs that users may or may not be willing to accept. We know that the scripting engine resident in the client's web browser executes all scripts, including the malicious ones injected by hackers to vulnerable websites. Disabling scripts from running altogether, a long recognized and widely deployed security practice, would make browsing much safer. Unfortunately, in the current Web 2.0 environment blindly disabling all scripts would render most websites useless.

Selective Script Prevention

Internet Explorer, Safari and Firefox all have available settings to prevent scripts from running in the browser. In addition, there are available add-ons and extensions that can give the user more control over what scripts are allowed to run on certain sites. Since most sites today employ scripting, disabling scripting in the browser would make web-surfing a dull experience at best and most likely make most current sites unusable. Total disabling of scripts is no longer an option.

NoScript (v1.7.6) (http://noscript.net/) is a browser extension tool that allows for selective script execution. This browser extension is an open-source tool developed for Mozilla Firefox. It installs directly into the browser and by default disables all scripts from running. The user will notice NoScript's effect within the first one or two websites visited because so many sites use some form of scripting to display banner ads or to control the login mechanism. NoScript presents a pop-up at the bottom of the browser screen that notifies the user when scripts are being blocked. Right-clicking on the NoScript icon in the browser toolbar shows a list of the currently visited domains and the particular scripts in each domain that are being blocked. The user then has the option of allowing a specific script to run temporarily. In addition, the user can indicate the domains that are trusted, which allows all scripts in those domains to run. Indeed, the list of blocked scripts may grow to as many as ten or fifteen scripts, making this tool seem daunting to non-savvy users. It may even take users multiple tries before enabling the correct script to run in order to see the desired content. There is also an option to turn on all scripts globally, essentially disabling the extension altogether. Of course, any script given permission to run can have that permission revoked as well.

Disabling macros in word processors and scripts in webpages has long been a simple way to enhance security. Security analysts have warned for some time that the integration of data and code creates numerous security loopholes. Unfortunately, the functionality of Web 2.0 sites depends on the use of methodologies like Ajax, which employs Javascripts extensively and in complex ways. Developers rely on such methodologies to display content in a manner that allows users to be more productive and also to provide the browsing experience many users now expect. The tight integration of code and content prevalent in modern websites makes it almost impossible to disable scripting entirely and difficult to disable scripting selectively.

Using NoScript, it is interesting to see how crippled the Web is when scripting is diasbled. For instance, every website in the first 20 sites listed in Alexa.com's top 100 for the United States, which includes YouTube.com, MySpace.com and Wikipedia.com, present scripts that NoScript blocks. Even Alexa.com can't display the web tracking graphs present on the front page of their site. Text-based sites like Wikipedia were not severely impacted because scripting is not used as extensively. YouTube, however, is rendered useless. All videos and many search options are driven by scripting. Once the correct allow permissions are set for the scripts belonging to the domain, all functionality returns. Banking sites are also impacted by this plug-in. The login mechanism for many sites requiring authentication utilizes scripting and is affected as well.

A few observations are evident from this non-scripting browsing experience. For casual browsing, disabling scripting takes most of the fun out of websites. The use of scripting, especially to develop dynamically updating Ajax-based applications on web forms represents the evolution of the Internet and the content it serves. Moreover, we can't expect casual Internet users to be able to pick the exact script they want to allow to run. Most users likely will default to the Allow All Scripts Globally option in NoScript just to be able to make use of the website.

Requiring Certificates for Scripts

Successful web-based attacks exploit the user's trust in a website. If a website is trusted, e.g., the user's bank site, a user should configure an extension like NoScript to allow all scripts from that site. If the bank site turns out to have a XSS hole that an attacker exploited, NoScript is not going to help. In this sense, it may be beneficial to develop an extension that does both selective script blocking to help against malicious scripting on random websites and also performs some kind of signature-based checking against scripts run from a trusted site, similar to the way Microsoft requires drivers to be digitally signed in its 64-bit versions of the Vista operating system (Relmer, 2006). If a high profile target for a web-based attack, like a bank site, wants to use scripting as a transport mechanism, they should have their scripts signed by a reputable authority. Now, either the browser or the extension would be able to check the validity of each script and choose whether or not to allow it to run. Nefariously injected scripts would be rejected because no valid signature would be included in the script code.

Signature-based scripting may be a viable solution against CSRF attacks. However, implementation would be difficult. Both browser manufacturers and web developers would have to conform to this new standard. Browsers would have to include support for an extension of the HTTP standard that enables websites to identify a script with a signature. The browser would need to read the signature, identify the encryption algorithm and check it against a predefined list of certificate authorities. The procedure can be similar to methods already employed for setting up a Secure Sockets Layer (SSL) tunnel for encrypted communication between the browser and a website. This browser modification would be made easier if all operating systems supported a modular kernel where security extensions can be loaded as libraries. The National Security Agency (NSA) has been developing such a security-enhanced architecture into the Linux operating system. The project has been named Security-Enhanced Linux (SELinux) and allows for security-centric modules to be loaded directly into the kernel as they are developed (Loscocco and Smalley, 2001).

Web application developers would need to conform to this standard as well. Obviously, all scripts that need to be executed on the client-side must include a certificate that identifies them as belonging to the website. These certificates would have to be purchased from a trusted certificate authority (CA), such as VeriSign (www.verisign.com) or Thawte (www.thawte.com). A valid certificate is purchased for a yearly fee that many websites, especially smaller ones, might not be willing to incur.

Visual Notifications of Patch Levels

Although our focus has been on web-based attacks, we have seen that the combination of Web 2.0 vulnerabilities and traditional exploitations such as Trojans allow for powerful attacks as well. Therefore, keeping systems fully patched and up-to-date is just as important today as in the past. We have seen that web-based attacks can come as hybrid exploits that use security flaws in both browsers and their extensions to gain higher access to the victim machine. It has taken the operating system industry longer than it should have to configure their systems to have tools such as automatic updating turned on by default. Non tech-savvy users are just recently realizing the importance of keeping their operating system (OS) fully patched and their antivirus software updated. And then there are the browser extensions. Most users may not even know that these extensions should be patched. In fact, it is estimated that only 59.1% of all Internet users globally surf the web with fully patched browsers and extensions (Frei, Dübendorfer, Ollmann and May, 2008, table 2).

One study (Frei et al., 2008) proposes a very interesting idea for browser and extension updates. The idea of the "best-before" date as it applies to the food industry is also applied to browser security. Just as a consumer would be hard-pressed to drink milk that displays an expiration date of 7 days prior, a user should be visually prompted that the system they are using is stale and unsafe. Computer users all became more aware of the state of security on the home computers when Microsoft released Service Pack 2 for XP. If the computer was out-of-date with OS patches or antivirus signatures, a red shield would display in the taskbar warning the user of the less-secure state. This visual red flag could be just as effective for surfing the web. The study proposes to have a message box docked in the browser that displays how many patches are missing and the number of days expired. The researchers also propose that popular websites can deploy such a tool on their websites, which would check the USER-AGENT information of visiting browsers and display a similar message at the top of the webpage.

We propose to take this one step further by creating a Network Access Control (NAC) gateway into high profile websites such as bank sites. A NAC gateway usually sits just outside the front-end web server and checks the client against a set list of requirements it must have in order to be allowed access to the site. These requirements could be antivirus version or definition file age, OS patch level, client firewall, etc. A website can run similar checks against a visiting user's browser version, patch level and extension patch level. Sensitive sites can employ two-fold vigilance by both hardening their website against web-based attack, while also making sure their web users are doing everything they can to protect themselves.

Website Report Cards

A number of antivirus companies have a free browser extension that visually makes web surfers aware of websites that are known to distribute malware. The plug-in displays a green, yellow or red icon next to the returned links on search engine results pages. McAfee's SiteAdvisor (http://www.siteadvisor.com/) and AVG's Safe Search (http://www.grisoft.com/ww.special-download-new-avg-8-software) are two such recently released products. Both tools display the risk level of websites by keeping a huge database of crawled sites and known malware. For instance, a Google search results page will display a red icon next to the link for a site because it is known to host downloads that install Trojans. This same idea can be applied to web-based vulnerabilities. Many security sites, such as securityfocus.com and ha.ckers.org, stay current on websites that are known to have XSS vulnerabilities. The vulnerable sites may or may not fix the problem immediately following a security disclosure. The browser tool can keep a database of known sites that have had these kinds of vulnerabilities and give users a visual warning that browsing with scripting enabled may be hazardous. Tips about safe browsing can also be displayed when the user places the mouse over the icon.

User Education

User education about Web 2.0 vulnerabilities can aid in reducing web-based attacks. This advice is by no means new, but should be put into context with current vulnerabilities. For example, a least-trust model should be followed when making associations on social networking sites. Any online interactions with unknown people should be considered suspect, especially if inquiries are made about online financial interests. As we have seen with CSRF attacks, these queries may be from cyber criminals casing potential suspects. Although targeted attacks are harder to establish, they are often the most devastating when successful. Users should be constantly reminded to only work in one browser window when establishing authenticated sessions to sites such as online stock trading or banking sites. A successful XSS hole will make your browser believe that the injected malicious script came from the trusted site and can potentially execute any action allowable on that site, including transferring funds or making withdrawals.

Protection and Prevention for System Administrators

System administrators and network security professionals have an extremely tough job. They have to ensure that their users employ safe browsing techniques. Additionally, it is their job to secure the websites they host to help protect the online clientele. Certain security procedures can be followed using existing tools in-house, sometimes, however, it is necessary to purchase additional third-party solutions to implement security policies to help mitigate the risk.

Centralized Management of Endpoint Security Solutions

The practices that should be employed to prevent company users from inviting unwanted malware into a corporate network are not new – OS patches must be current and antivirus/antispware must be deployed and kept up-to-date. Keeping thousands of company computers current as far as OS patches and antivirus/antispyware definitions are concerned is not an easy task. Centralized management of security levels across the network allow administrators to simplify patch rollouts, as well as react quickly to non-compliance events. Machines can have non-compliance events when OS patches are not up-to-date, antivirus signatures are old, local firewalls are disabled, browser plug-ins need to be updated, etc. With the increasing number of security applications needed to keep the enterprise safe from these threats, a single display that shows a number of widgets dedicated to different threat prevention tools is essential.

There are enterprise level management tools available to keep statistics on the safety level of all client machines. One such tool is McAfee's EPolicy Orchastrator v4.0 (http://www.mcafee.com/us/enterprise/products/system_security_management/epolicy_orchestrator.html). The McAfee tool provides a display that gives a dashboard view of all compliant and non-compliant systems in the network for both wired and mobile machines. Many applications, including non-McAfee tools, can be managed in this central location and thresholds can be set for each, including patch levels and signature file age. This way, administrators have an overview of all systems, and red flags alert them to potentially vulnerable systems immediately.

Inventory of Approved Browser Plug-ins

Browser extensions are a different story. Since most extensions are developed independently of both the browser and the operating system, there is no available tool that can monitor the patch levels of each one. This can present an administrative nightmare for system administrators. Limiting the amount of extensions available to corporate users is the key, as well as employing privilege levels for the users so that they cannot install rogue extensions themselves. Even still, users require many extensions that render web content needed in their day-to-day work. Examples of such extensions include Adobe Reader, Apple Quicktime, RealPlayer, Java Runtime Environment (JRE) and many more. System administrators should keep an inventory of all allowable extensions and monitor the extension's websites for updates, especially those with security implications. Keeping a consistent and up-to-date software inventory across all machines company-wide not only keeps the internal network safer, but makes deploying security patches easier.

Blacklist High-Risk Websites

We have seen that websites that offer Web 2.0 technologies, e.g. social networking sites, tend to attract web-based attacks. System administrators should limit access to such websites for corporate users. Targeted web-based attacks like CSRF can bypass firewalls and access any server in the corporation's enterprise. Because of this, whitelists and blacklists should be kept and updated for internal web surfing. In an ideal environment, the principle of least privilege should be followed, where employees are just given access to resources needed to do the job. Of course, what web resources an employee will need is very difficult to determine due to the vast amount of information available on the Internet. This is discussed further in (Bishop, 2003).

Tools like McAfee's SiteAdvisor have corporate versions that not only display warnings to users, but also can give the system administrator the power to deny all high risk sites. When users try to navigate to these sites, they are presented with a page that explains why they are being blocked. The page also allows them to notify the administrator of the unsuccessful attempt. Social networking sites should be considered for corporate blacklists as well. Often access to these sites is not required for work related purposes. In addition, since these sites accept user-supplied input, they tend to be breeding grounds for Web 2.0 vulnerabilities. Not allowing access at all will most definitely limit the threats that they present.

Hash Lists for Known Scripts on the Web Servers

With respect to the web servers, system administrators should be concerned with the state of known scripts that run in response to web requests. It is true that Persistent XSS attacks generally inject their malicious payloads into areas that accept user-supplied content, as opposed to system scripts. However, any script that the system administrator knows the content of a priori should have a hash value in an inventory list. Thus, a weekly scan of all known scripts can be taken, new hashes computed for each one and then a comparison against known and expected values can be done. Any anomaly should be considered a red flag and proper reactive methods should be followed. Keeping a hash value inventory is an effective way to detect tampering and is discussed in Pavlou and Snodgrass (2006).

Appliances for Network Security Professionals

Network security specialists will need to be concerned with the traffic that is coming in and out of their network, especially in regards to web-facing applications in the company's demilitarized zone (DMZ). The DMZ is an area of the corporate network that is firewalled off, or otherwise separated from the internal network for security reasons. Many times, the DMZ also has a firewall in between it and the Internet cloud for added security. Here, network address translation (NAT) and port forwarding techniques are used to translate the incoming web requests to the proper web server. Unfortunately, firewalls do not give the needed security against web-based attacks because most firewalls are configured to allow HTTP traffic running over TCP port 80.

In this case, additional network appliances should be considered to observe and track all incoming and outgoing web requests. Appliances are closed systems, many of which are Linux-based, that specialize in a particular activity. For instance, an Intrusion Prevention System (IPS) appliance would sit on the network edge, monitoring all traffic that passes through its interface. The nature of the traffic is analyzed for attack signatures; such as excessive SYN requests within ingress traffic, which is indicative of a DDoS attack, or excessive calls to many different IP addresses within egress traffic, a common trait of Botnet applications looking to call home. Proper reactive measures can then be executed, like denying traffic to or from a host on the network once suspect traffic matches a preconfigured signature.

An appliance that monitors network traffic for XSS or CSRF attacks can help to prevent Web 2.0 attacks at the network perimeter. The scanning can be either behavior-based or signature-based. Behavior scanners can be taught what normal request/reply activity looks like for certain webpages and then raise an alarm when any deviation from the norm occurs. For instance, if a message board page is only supposed to accept a text entry and reply with a success or failure message, but document properties like cookies are being redirected to another site instead, an alarm can warn the network security team that a possible XSS attack is on-going. Likewise, egress traffic, or traffic leaving the web server, can be monitored for behavior like many SYN requests or image source requests being sent back to the visiting browser. Signature-based appliances can serve as a backup for user-supplied input filtering. Just as code should be included on the website itself that scrubs the input for unwanted characters or methods, this scrubbing can go on before it even gets to the website. The manufacturer of the appliance can maintain signature files for all known variations of unwanted input and make these signatures available for daily updates, like antivirus definitions. The network security team can work with the development team to tune what methods are allowed and which should be rejected right at the perimeter. Of course, like many IPS devices, these appliances really need to be tuned in order to lesson the amount of false positives they raise. Ultimately, stopping web-based attacks before they reach the internal network is ideal.

Preventive Methods for Developers

Software developers always seem to be the scapegoat when it comes to disclosed web-based vulnerabilities. Granted it is the coders that leave the hole open in the first place; however, the responsibility should be shared between both the developers and the supervisors that manage them. Secure coding is not a new practice by any means, but the threat landscape that developers should be concerned with changes everyday. In regards to Web 2.0 vulnerabilities, developers need to be well-trained in the use of user-supplied input scrubbing and non-verbose error messages.

Current application developers need to employ secure coding practices in every project. Unfortunately, developers are always working under a deadline that needs to be met, lest the company as a whole suffers the consequences, for example, lost contracts, breach of contract lawsuits, loss of performance bonds, etc. The last thing a developer is going to worry about when coming down to the wire is security - they are more concerned simply with making the application work. Secure design and coding has to be considered from day one in every project. According to CERT's *Build Security In* initiative for secure coding, “software security is fundamentally a software engineering problem and must be addressed in a systematic way throughout the software development life cycle.” (DHS, 2008).

Instilling a security-centric mindset into the programmer is much harder than requiring that the programmer already works in this fashion. This quality is no longer looked at as a recommended skill; rather it is a requirement. There are many classes that are available to developers to help improve their overall programming skills, with an emphasis on secure coding practices. Often these classes are taught by the programming language developers themselves, e.g., Microsoft for .NET or Sun for Java. More recently, organizations are providing developers with classes centered on secure Ajax development to help mitigate the current problem Web 2.0 technologies are facing, such as the Secure Ajax Development class offered by Ciphent (http://www.ciphent.com/training/courses/secure_Ajax_development). Supervisors should demand that developers become certified in these skills before hiring. Existing developers should be made to attend these classes to become certified. Unfortunately, training budgets are all too often the first to go when it comes to budget cuts. In this case, developers should be encouraged to foot the bill themselves with the promise of future incentives. Ensuring that the entire development team is security-savvy from the start of a project will always pay off in the end.

Web Application Penetration Testing

Third-party penetration testing for web applications is an extremely effective tool for discovering Web 2.0 vulnerabilities. Just like many of the mitigation techniques discussed so far, penetration tests are not a new idea. However, it has become standard practice to use penetration testing techniques against web applications in order to preemptively search out

XSS vulnerabilities. They are less intrusive than network security pen tests because it is easy to test for a XSS hole without causing any damage, as we have seen with pop-up box techniques.

There may not be a better way to test the solidarity of your web server's security against hackers than having a third-party, an impartial group of "hackers," attempt to exploit overlooked vulnerabilities. Such a security firm will try to exploit all the attack vectors discussed in this paper against a company's Internet- and intranet-facing web applications. The firm would test for XSS holes on pages that accept user-supplied input, especially any page that echoes this input back to the user. Automated tools will try simple Reflective attacks by entering code within <SCRIPT> tags, as well as complex HTML variants to attempt to evade the sites input filters. If the site allows transactions, the attacker will attempt possible CSRF attacks by embedding them in static URL actions. After the tests are concluded, the firm will present the owner of the tested site a progress report detailing all discovered vulnerabilities. The company will advise on comprehensive procedures toward fixing all uncovered vulnerabilities. Prevention of similar security flaws in the future is the desired effect.

Conclusions

The nature of the World Wide Web has changed. As technology grew faster, it has allowed web developers to expand the capabilities of their offerings. No longer are Internet users bound by the static characteristics of Web 1.0. Web-based applications now allow users to fully interact with online content by adding and changing their own input in whatever way they see fit. In many of today's most popular websites, end-users are as likely to provide content as developers. Often end-users provide not only text but active content, scripts that run on the browsers of anyone who visits the site. The tight integration of static and active content in current websites and the ability of users to contribute both types of content have led to a new generation of novel security vulnerabilities know as Web 2.0 vulnerabilities.

As with every evolution of computer technology thus far, this new collaborative Web is under attack. We have seen what the vulnerabilities are and why it is so hard to get a handle on solutions for the Web 2.0 problem. XSS holes are a common phenomenon - built into websites both natively and by plugging-in smaller applications with existing vulnerabilities. These XSS holes are being exploited by attackers to accomplish malicious activities such as website defacement, cookie stealing, browser hijacking and script injection. Hackers also combine XSS with other attack vectors, like social engineering, to accomplish more sophisticated attacks. Hybrid attacks, such as CSRF, enable hackers to target victims that use online services hosted on sites with known XSS vulnerabilities to commit identity theft or execute transactions in the context of the victim's account. Additionally, we have seen evidence to suggest that the nature of Web 2.0 vulnerabilities can allow for distributed attack infrastructures called Puppetnets to take control of large numbers of unsuspecting web surfers and use their browsers for nefarious means. Puppetnets are of exceptional interest because of their likeness to Botnets, but without the requirement to install a bot binary on the victim.

Most Web 2.0 vulnerabilities stem from the common XSS hole. All that is needed to protect against XSS vulnerabilities is solid input filtering to prohibit anyone from injecting script code to the website that later gets rendered by a visiting user's browser. However, an average of 10 XSS flaws is disclosed every month on the Bugtraq mailing list alone.

We have tried to identify the cause for the alarming number of vulnerabilities exhibited by Web 2.0 technology. It begins with the developers who need consistently to practice secure coding techniques. IT supervisors should demand this skill of all newly hired programmers, as well as ensure that the existing development team is encouraged to keep current with new techniques. If the development process becomes inherently more secure, it may be sufficient to prevent new vulnerabilities from cropping up.

While the development process is trying to mitigate the problem on the back-end, system administrators and end-users need to do all they can to protect themselves on the front-end. Because many Web 2.0-type attacks also exploit known vulnerabilities in OSs and applications, staying up-to-date as far as security patching for flaws in the OS, the browser and its plug-ins is as important as ever. Most Web 2.0 problems are out of the control of the users, so best practices for safe web surfing must be followed.

Although Web 2.0 vulnerabilities are a serious and widespread problem, we believe it is possible to limit them. Researchers are diligently working on fixing the problem while developing tools to keep users safe until that time arrives. These tools should be embraced by the Internet community as a whole and built-in to both the browsers and the websites.

Users should be educated about the problem and given the opportunity to surf safely. They should not be asked to shy away from what this new technology has brought to the table.

This paper has identified the problems facing the new trend in the Internet. We have given an overview of existing mitigation techniques, as well as offered many variations and extensions to further protect the Internet from the exploits Web 2.0 technology allows. Innovation cannot be hindered by vulnerabilities created out of carelessness and lack of foresight. Granted, not all vulnerabilities allowed by new technology can be seen ahead of time. When they are discovered, however, haste needs to be made to both understand and mitigate the threat so that all future efforts are toward the next evolution.

References

Almeida, M. & Fernandez, K. (2008, June 26). *ICANN and IANA Domains Hijacked by Turkish Crackers.* Zone-h.org. Retrieved July 1st, 2008 from http://www.zone-h.oeg/content/view/14973/30/

Auger, R. (2008, April 17). *The Cross-Site Request Forgery (CSRF/XSRF) FAQ.* CGISecurity Inc. Retrieved May 12th, 2008 from http://www.cgisecurity.com /articles/csrf-faq.shtml

Bishop, M. (2003). *Computer Security: Art and Science.* Pearson Education, Inc.

Breen, R. (2006, December 18). Circumventing Browser Connection Limits for Fun and Profit. Message posted to blogsite Ajaxperformance.com. Retrieved June 24th, 2008 at http://www.Ajaxperformance.com/2006/12/18/circumventing-browser-connection-limits-for-fun-and-profit/

Chien, E. (2006, July 18). MySpace Shockwave Flash Hack. Message posted to Symantec's Vulnerabilities & Exploits blogsite. Retrieved June 2nd, 2008 from https://forums.symantec.com/syment/blog/article?message.uid=305415

Di Paola, S. & Fedon, G. (2006, December). *Subverting Ajax.* 23rd Chaos Communication Congress Conference. Retrieved on July 18th, 2008 from http://events.ccc.de/congress/2006/Fahrplan/attachments/1158-Subverting_Ajax.pdf

Frei, S., Dübendorfer, T., Ollmann, G., & May, M. (2008, July 1). *Understanding the Web Browser Threat: Examination of Vulnerable Online Web Browser Populations and the "Insecurity Iceberg".* ETH Zurich Tech Report No. 288. Published by author.

Retrieved on July 3rd, 2008 from http://www.techzoom.net/publications/insecurity-iceberg/index.en

Handy, N. (1988, October). *The Confused Deputy (or why capabilities might have been invented).* ACM SIGOPS Operating Systems Review, Vol. 22, Iss. 4, pgs. 36 - 38. Retrieved May 12th, 2008 from ACM Digital Library database.

Jackson-Higgins, K. (2007, January 4th). *Botnets Don Invisibility Cloaks*. United Business Media LTD. Retrieved June 20th, 2008 from http://www.darkreading.com/document.asp?doc_id=113849.

Jackson-Higgins, K. (2007, March 29). *Killer Combo: XSS + CSRF*. United Business Media LTD. Retrieved May 16th, 2008 from http://www.darkreading.com /document.asp?doc_id=120801

Keizer, G. (2005, October 21). *Dutch Botnet Suspects Ran 1.5 Million Machines*. TechWeb Technology News. Retrieved June 20th, 2008 from http://www.techweb.com /wire/security/172303160

Keizer, G. (2007, August 12). *'Hackers' Deface UN Site*. Computerworld Inc. Retrieved June 3rd, 2008 from http://www.computerworld.com/action/article.do? command=viewArticleBasic&articleId=9030318

Kyrnin, J. (n.d.) *What is Ajax?: Building Web Applications Just Got More Fun*. Retrieved June 29, 2008 from http://webdesign.about.com/od/Ajax/a/aa101705.htm

Lal, E. (2005, October 17), *Teen Uses Worm to Boost Ratings on MySpace.com*. Computerworld Inc. Retrieved May 15th, 2008 from http://www.computerworld.com /securitytopics/security/holes/story/0,10801,105484,00.html

Lam, V, T., Antonatos, S., Akritidis, P., & Anagnostakis, K. G. (2006, Novenmber) *Puppetnets: Misusing Web Browsers as a Distributed Attack Infrastructure.* Conference on Computer and Communications Security, Proceedings of the 13th ACM Conference on Computer and Communications Security. Ses. Attacks and cryptanalysis, pgs. 221 - 234. Retrieved June 23rd, 2008 from the ACM Digital Library database.

Leyden, J. (2005, July 6th). *MS UK Defaced in Hacking Attack*. The Register. Retrieved June 3rd, 2008 from http://www.theregister.co.uk/2005/07/06/msuk_hacked/

Loscocco, P. & Smalley, S. (2001, June). *Integrating Flexible Support for Security Policies into the Linux Operating System.* Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference. Retrieved July 18th, 2008 from the National Security Agency Website: http://www.nsa.gov/selinux/papers/freenix01-abs.cfm

Mann, J. (2007, January 31). *MySpace speaks about Samy Kamkar's sentencing.* TechSpot.com. Retrieved April 20th, 2008 from http://www.techspot.com /news/24226-myspace-speaks-about-samy-kamkars-sentencing.html

Mook, N. (2005, October 13). *Cross-Site Scripting Worm Hits MySpace.* BetaNews Inc. Retrieved April 13, 2008 from http://www.betanews.com/article/ CrossSite_Scripting_Worm_Hits_MySpace/1129232391

Nadir, D. (2007, March 21), *How to Protect Against Web 2.0 Threats*. HayMarket Media. Retrieved June 7th, 2008 from http://www.scmagazineus.com/How-to-protect-against-Web-20-threats/article/34711/

Department of Homeland Security (DHS). National Cyber Security Division. Build Security In Initiative (n.d.). Retrieved July 18th, 2008 from the United States Computer

Emergency Response Team website: https://buildsecurityin.us-cert.gov/daisy/bsi/home.html

OpenDocMan Cross Site Scripting (2008, June 17). Retrieved July 5th, 2008 from the SucuirtyFocus Archive site: http://www.securityfocus.com/archive/1/493390

O'Reilly, T. (2005, September 30). *What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software.* O'Reilly Media Inc. Retrieved March 4, 2008 from http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=1

Pavlou, K. & Snodgrass, R. T. (2006, June). *Forensic Analysis of Database Tampering.* International Conference on Management of Data. Proceedings of the 2006 ACM SIGMOD international conference on Management of data. Retrieved July 18th, 2008 from the ACM Digital Library Database.

Relmer, J. (2006, February 1). *Microsoft to Require Signed Drivers for 64-bit Vista.* Ars Technica, LLC. Retrieved June 28th, 2008 from http://arstechnica.com/news.ars/post/20060201-6098.html

SQL Injection Attacks Against Banks on the Rise (2006, July 19). HNS Consulting LTD. Retrieved June 1st, 2008 from http://www.net-security.org/secworld.php?id=4076

Swartz, J. (2008, February 10). *Social-Networking Sites Going Global.* Gannett Co. Inc. Retrieved April 11, 2008 from http://www.usatoday.com/money/industries/technology/2008-02-10-social-networking-global_N.htm

Tanaka, K. (2008, June 1). *Botnet Cyber Attack Costs Company 300 Million Yen.* The Yomiuri Shimbun, Inc. Translation retrieved June 20th, from a blog site at http://fergdawg.blogspot.com/2008/06/botnet-cyber-attack-costs-company-300.html

Teenage Hacker Facing Court Case for Data Theft (2006, January, 22). The Taipai Times. Retrieved June 1st, 2008 from http://www.taipeitimes.com/News/front /archives/2006/01/22/2003290158

Wasserman, G. & Su, Z. (2008, May). *Static Detection of Cross-Site Scripting Vulnerabilities.* International Conference on Software Engineering, Proceedings of the 30th International Conference on Software Engineering, Sess. Testing II, pgs. 171-180. Retrieved July 1st, 2008 from the ACM Digital Library database.

Watkins, P. (2001, June 13). Cross-Site Request Forgeries (Re: The Dangers of Allowing Users to Post Images). Message posted to bugtraq@securityfocus.com archived at http://www.tux.org/~peterw/csrf.txt

WEBAlbum v2.0 Remote Stored Cross Site Scripting Vulnerability (2008, June 5). Retrieved July 5th, 2008 from the SecurityFocus Archive site: http://www.securityfocus.com/ archive/1/493143