PANDA
CLOUDINTERNETPROTECTION

Simply... Evolution

YOUR BROWSER WEARS NO CLOTHES: WHY FULLY PATCHED BROWSERS REMAIN VULNERABLE

PANDA CLOUD
OFFICE PROTECTION

PANDA CLOUD
EMAIL PROTECTION

PANDA CLOUD
INTERNET PROTECTION

PANDA
SECURITY

Index

1. Introduction

As users of technology, we have been taught that the Internet is not always a safe place, but that we can protect ourselves by patching and hardening our systems. While patch management and system hardening have long been the basics for enterprise security, shifts in technology and attack patterns are changing the rules. Today it is not just possible, but common, for a user's fully-secured machine to become compromised. At times, this occurs because of increasingly-sophisticated social engineering attacks or newly-discovered (so-called zero-day) vulnerabilities. However, it is increasingly resulting from exploitation, which does not target a specific vulnerability on an individual platform but is instead abusing the functionality and structure of the Internet itself. This fundamental shift to naked browser attacks changes everything. Enterprises must adapt their approach to security to stay ahead in the never-ending arms race of web security.

We are now firmly entrenched in the era of web applications. It is no longer simply desirable but expected that the majority of enterprise development be architected as web applications, no matter whether those applications are to be used internally or externally. This shift has brought a degree of uniformity to the IT landscape. No matter what operating system you choose and regardless of the hardware you select, mobile or otherwise, it will have a web browser and that browser will adhere to at least a basic set of operational standards.

By 2009, a JavaScript engine had become the norm on even mobile phone browsers, and the majority of platforms can handle at least the most popular Rich Internet Application technologies such as Adobe Flash. This subtle and voluntary standardization has not only made possible productive Web 2.0 technologies such as AJAX, but has unfortunately also created a broad attack base for those looking to do harm. If attacks can abuse JavaScript, for example, as opposed to a specific version of Internet Explorer, the potential population for attack rises from some finite percentage of Internet users to virtually 100%.

Attackers once viewed browsers themselves as targets for attack. Now, browsers are instead becoming facilitators of attacks. Browsers are simply doors that enable access to the data the attacker is after. The difference from earlier attack types is that vulnerability does not have to be identified and exploited at the browser.

Today, many attacks work cross-browser and cross-platform because they don't target the browser itself; they target functionality that remains the same regardless of the browser platform. The web was designed to be open – not secure. This ideal has not been lost on attackers, so they dedicate much of their time to bending the rules of the web to work in their favor. This has in turn led to a surge in successful attacks on fully-patched machines. We call such attacks naked browser attacks because no patch to protect end users is forthcoming. In this new world order, we must revisit our approach to web security if we are to protect ourselves and our businesses effectively.

2. A Little History

The attack cycle used to be simple – attackers would uncover vulnerability within an application or operating system, exploit it, and continue to do so until the appropriate vendor released a patch to address the problem. This is, of course, an oversimplification of the multitude of variables that could be involved in any specific instance, but it does capture the essence of what has driven the anti-malware industry for some time. One encouraging change has been the shrinking window of time during which an attacker could take advantage of a given vulnerability. A decade ago, it was not uncommon for enterprises to spend weeks or even months conducting regression testing before determining it was "safe" to deploy vendor patches. This allowed for a substantial period of vulnerability during which millions of machines were open to attak.

Fortunately, as understanding of the risks of exposure and improved processes for disseminating patches and communicating the risks associated with individual vulnerabilities has evolved, the window of opportunity for attack has now shrunk to a matter of days or even hours following the publication of news regarding a given vulnerability.

Looking back over the past decade, as illustrated in Figure 1, three distinct eras existed for attackers. Up until 2004, most attacks focused on vulnerable Internet-facing services such as popular web, mail and FTP servers. Over time, critical vulnerabilities in such services were exploited by attackers, which resulted in fast-spreading worms.

As Web 2.0 became more widespread and server security improved, attackers switched their target to browsers. A plethora of vulnerabilities in all browsers led to increased attacks on end users. While such attacks generally required a social engineering component to convince a user to view a page or click on a link, these challenges were typically minimal.

Today, we are entering the era of naked browser attacks. The attacks are naked, in that there is no specific vulnerability in the browser itself, but the attacks target end users.

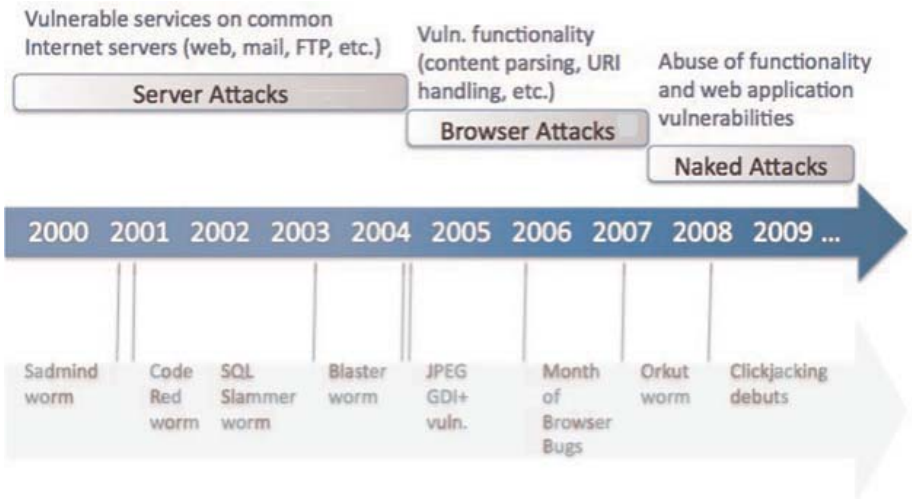


Figure 1 - Evolution of Attacks

3. Naked Browser Attacks

Naked browser attacks against secured browsers succeed either because they abuse trust established between the browser and a vulnerable web application or because they abuse functionality of the web itself. Several types of attack fall into these categories and, while it is not the aim of this paper to go into detail on all of them, the process of cross-site scripting (XSS), which abuses browser/server trust and click-jacking, leveraging intended functionality in an unintended way, is described below.

3.1. Cross-Site

Cross-site scripting remains one of the most prevalent attack methods that web users face today, despite its relatively high profile over the past several years. It has been a fixture in the Open Web Application Security Project (OWASP) Top Ten[1] list of common web application vulnerabilities since the list was first introduced in 2004. Additionally, the December 2008 White Hat Website Security Statistics Report[2] indicated that two thirds of all websites probably include XSS flaws. This is a truly frightening statistic that leaves users at risk each and every time they browse the web.

XSS illustrates one of the fundamental changes in security brought about by the interconnected nature of the web. With XSS, the vulnerability which is abused resides not within a user's browser but in a third party web application which the user accesses. However, the user is still the victim of the attack because the browser responds to the injected malicious JavaScript as it should, by interpreting the code. In this type of attack, the browser has no way of distinguishing between intentional user-supplied content and content which may have been injected through an XSS attack.

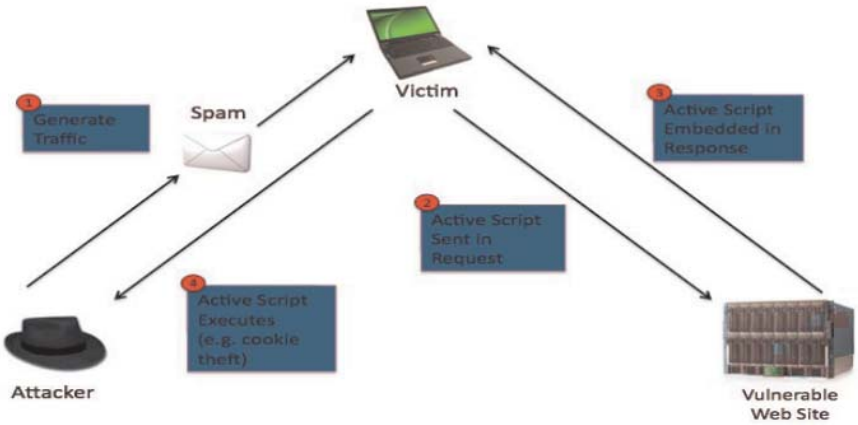


Figure 2 - Typical XSS Attack Scenario

3.1.1. A Typical XSS Attack

Figure 2 shows a typical XSS attack scenario. The following walk-through details why XSS succeeds without the need for browser vulnerability.

1. **Generate Traffic:** To be successful, an XSS attack requires that a user send a specially-crafted request to a vulnerable web server. The request contains embedded active content (usually JavaScript), which is designed to perform an action of the attacker's choosing. Often, the script will attempt to send the user's cookies for the targeted website to the attacker. Sending spam email is a common way to get victims to send the predefined request, typically by

embedding a link within an HTML-formatted email message that includes a message to entice the victim to click on the link.

2. **Active Script Sent in Request:** If the user clicks on the link in the spam email message, a request is sent to the vulnerable web server. Rather than a simple request for the URL of a web page, the request will also include the injected JavaScript, either as parameters in the URL itself (GET request) or within the body of the request (POST request).

3. **Active Script Embedded in Response:** The vulnerable web page includes functionality which will accept user-supplied input and include it in the dynamically-generated page returned to the user. Such behavior is fine as long as the user-supplied input is appropriately sanitized to ensure that the content received was the content expected. The absence of such controls is what permits XSS attacks.

4. **Active Script Executes:** When the browser receives the response, the page content will include the injected malicious JavaScript, which will then be interpreted and acted upon by the browser.

3.1.2. Impact

While this simple attack scenario involves a single attacker and victim, XSS is commonly used in more complex attack scenarios. In January 2008, it was revealed that attackers had used XSS vulnerability on the login page of Banca Fideuram, a major Italian bank, to inject a fake login form within an IFRAME[3]. The attack sent a user's authentication credentials to an attacker-controlled server and was particularly dangerous as it was hosted on a trusted, SSL-protected webpage. XSS attacks are quickly evolving and are no longer static in nature. XSS worms have now started to appear on popular social networking sites such as Orkut[4] and MySpace[5].

It is important to remember that XSS attacks will succeed regardless of whether users have applied up-to-date security patches. XSS attacks succeed because browsers are designed to interpret JavaScript. With XSS, the attacker is abusing input validation vulnerability on a web application, and the user viewing the page becomes the victim. The idea that users need to surf only 'reputable' pages to remain safe simply does not apply any longer. Almost all major websites have experienced some form of XSS vulnerability and, given the social engineering component of an XSS attack, sites with high volumes of traffic tend to be targeted in the most successful attacks.

3.2. Clickjacking

Clickjacking leapt into the media spotlight in the summer of 2009 when researchers were asked by Adobe to pull a talk at a well-known security conference on the subject just days before the presentation was to be made. This type of attack layers good content over bad, sprinkled with a little social engineering to trick users into performing an action they did not intend.

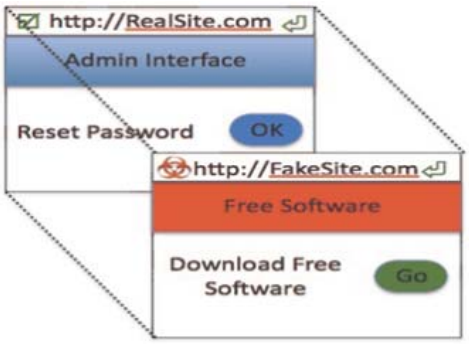


Figure 3 – Clickjacking

Figure 3 shows an administrative interface permitting a password reset being layered together with a fake site, which obfuscates everything. The content of the fake page is designed to hide what is really going on and convince an unsuspecting user to, in this case, reset his password, not download free software. Successful clickjacking requires the following three components:

1. **Embedded Content:** The targeted action (e.g. password reset), which is on a page not controlled by the attacker, is embedded within a page controlled by the attacker. This is typically accomplished by using an IFRAME on the attacker-controlled page. This is why 'frame busting' code – which prevents content from being displayed in an IFRAME - is commonly recommended as a server-side defense against clickjacking.

2. **Obfuscation:** The third-party content, including the password reset button, will not be visible despite being on the page, because its opacity value is set to zero. Opacity is used to adjust the transparency of an object.

3. **Layering:** Attacker-controlled content is actually layered below the third party content and positioned to ensure that the two buttons line up precisely. However, the 'Go' button for downloading software is visible instead of the 'OK' button for the password reset due to the opacity settings, as noted above. Layering is accomplished by leveraging z-index properties, which set the depth value of web page elements. In this case, the third-party content would have a higher z-index value than the attacker-controlled content. The result is that, although the victim sees the 'Go' button, what's actually clicked is the 'OK' button.

3.2.1. Impact

Clickjacking is an enabler for social engineering attacks. As with XSS attacks, users will not fall victim simply by viewing a malicious web page. Instead, they must click on a link to trigger the attack. By combining a variety of legitimate HTML formatting techniques, clickjacking facilitates the necessary social engineering by making it appear to the end user that they are clicking on a link other than the one the browser interacts with. Once an attacker can influence the mouse clicks made by a user, the potential attacks that can be conducted are virtually limitless. Having a user unknowingly upload data to an attacker-controlled location, for example, can compromise privacy and breach compliance regulations. Authentication can be bypassed by adding a rogue user account or lowering predefined security settings. Complete system compromise can be achieved if a user's mouseclicks download a malicious binary. In short, the potential for damage in a successful clickjacking attack is limited only by the imagination of the attacker.

Adobe was heavily affected by clickjacking, not because they produce a web browser that permits it but because the Flash Player Settings Manager could be abused by it. Settings for Flash Player are managed by accessing embedded Flash content on an Adobe web page[6]. In the Website Privacy Settings tab (see Figure 4), it is possible to set a site's permissions so that the site is always trusted and can access a machine's webcam and microphone without first requesting permission.

By obfuscating this console with fake content and social engineering a victim into clicking on a few specific areas of the screen, an attacker could hijack a the webcam and microphone of an unsuspecting victim and physically spy on them. This type of attack was demonstrated on the Guya.net blog

and was the catalyst that ultimately forced the public dissemination of clickjacking details[7]. The Guya.net proof of concept created a game, implemented using JavaScript, which asked a user to follow a button moving around the screen and click on it. While some clicks were innocuous, others were altering the user's Flash Manager settings and ultimately selecting 'Always Allow', as shown by the arrow in Figure 4.

Figure 4 - Adobe Flash Player - Website Privacy Settings

While Adobe has addressed this specific attack, by adding framebusting code to the Settings Manger, clickjacking still works just fine in the majority of modern browsers. It works because IFRAMEs and properties such as z-index values and opacity are standards that are respected by most common browsers. Individually, they are powerful tools, which allow for the design of some truly impressive web applications. However, when combined and used by a malicious attacker, they can be made into a viable attack vector. Once again, this type of attack does not leverage individual vulnerabilities but instead abuse intended functionality by using it in ways other than it was intended.

3.3. Other Attacks

XSS and clickjacking attacks are certainly not the only client-side attacks that don't rely on client-side vulnerabilities, simply two of the better-known. There are numerous other types of attack with similar characteristics; cross-site request forgery, content spoofing, URL redirection, HTTP response splitting, and others all demonstrate elements of naked browser attacks. Moreover, many of these attacks represent emerging issues which are just starting to be seen in wide-ranging attacks.

3.4. Challenges

You can't stop what you don't know about. An attack that blends in with legitimate web traffic will always be harder to detect. A browser exploit lends itself to traditional signature-based detection, as an attack generally requires that anomalous traffic be sent. In the case of a buffer overflow in a web browser, web content will need to be created which includes data to trigger the attack, shellcode to execute once control has been gained and some data for padding to ensure that everything lands in the right place. None of this is standard content on a web page, so it's relatively easy to detect. Clickjacking, by contrast, is much harder to detect using signatures, because b none of the components of clickjacking is nefarious individually. All are legitimate properties available to web application developers and will commonly be seen on a variety of web pages. It is the combination of a variety of legitimate attributes which makes an attack possible and the likelihood of a single preventative silver bullet remote.

4. Defense in Depth

Defending against attacks which succeed regardless of diligent patching and hardening browsers is an unnerving prospect. Tightening patch management procedures had been set as a first line of defense, but here are increasing volumes of attacks that bypass the entire process. What's worse is that naked browser attacks typically involve elements of social engineering and it is difficult, if not impossible, to prevent an attack that involves an employee serving as an unknowing accomplice.

Almost any discussion of how to defend against attacks such as XSS or CSRF will concentrate on how to secure the web application, not how to protect the browser affected by the attack. The focus of the majority of our security capital to date has been on defending servers, not browsers. However, typical enterprises have hundreds of browser installations for every server, and many of those browsers reside on laptops that leave the confines of the enterprise on a regular basis. Vulnerability is exacerbated by the fact that individuals with limited security knowledge operate those browsers. When looking at enterprise security from that perspective, it is easy to see why priorities need to be shifted.

4.1. Existing Solutions

Browser patches cannot protect against these attacks because the browsers themselves are not the point of vulnerability; they are behaving as intended. That said, there are some client-side applications that can help to protect browser abuse. , for example, is an excellent extension for Firefox and other Mozilla-based browsers that permits granular control over the execution of active content such as JavaScript, Java, Flash, and other plug-ins. It also includes specific controls to identify and block XSS and clickjacking attacks. Administrators should be aware that NoScript is designed for sophisticated users and many of the options may be confusing to an average user. While some administrators recommend disabling script engines altogether within browsers, this is no longer a viable option given the heavy reliance on JavaScript by modern web applications.

In the long run, it is to be hoped that browser vendors will begin to expand security functionality to combat against attacks despite the fact that the vulnerabilities leverage weaknesses in web applications as opposed to the browsers themselves. Microsoft's Upcoming Internet Explorer 8, for example, will include functionality to detect reflected XSS attacks[8].

Intrusion Detection/Prevention (IDS/IPS) systems often have signatures to detect attacks such as XSS, but they tend to identify exploitation of specific popular web applications. As noted earlier, signature-based detection of the attacks discussed in this paper is not a trivial matter, as the actual attacks can take many forms. Signatures that are too specific will miss the attacks, and those that are too generic will result in time-wasting high false positive rates.

4.2. Defending Against Naked Browser Attacks

Not surprisingly, there is no silver bullet to protect against naked browser attacks. Patches cannot address the weaknesses that permit such attacks and, with plenty of finger-pointing going on to place blame elsewhere, quick fixes will not be forthcoming. With that in mind, it is important that enterprises implement a range of detection and prevention measures to combat naked browser attacks.

4.2.1. Monitor

Effectively monitoring and logging activity on the network can ensure that naked browser attacks are isolated for follow-up when they do occur. Logs should be consolidated and not just maintained separately at each individual web gateway, enabling incidents to be correlated across physical locations. Web logs can be analyzed for anomalous traffic patterns such as large spikes in traffic to a particular page as attackers herd users to a particular attack target location. Sudden drops in expected traffic volume could also raise suspicion, as infected machines may be blocked from going to particular sites. This often happens in order to prevent the downloading of new anti-virus signatures that might identify an infected machine. But monitoring alone is not sufficient. Someone must own the process to ensure that that reports are produced, analyzed and escalated when necessary.

4.2.2. Manage

A common mantra in security is that users should only be given the appropriate level of access necessary to do their job. Why is it then that when it comes to web access, enterprises typically let users do whatever they want, with the possible exception of blocking objectionable content through URL filtering? Web applications have been given that name for a reason – they are applications and as such access should be restricted based on functionality, not just destination. For example, while it may be fine for users to view content on Facebook, it's probably not appropriate to permit unfettered uploading of content in order to protect against data leakage. Look for solutions that enable control over not just where users are going, but also what they're doing.

4.2.3. Merge

There are a number of commercial and free data feeds (e.g. Phishtank, Google Safe Browsing, OpenDNS, etc.) that identify potentially malicious content. Such feeds can be incorporated into web filtering solutions to block access to sites that may involved in browser-based attacks such as phishing scams or botnet attacks. When making use of such content, it is important to also regularly review metrics to ensure that the lists are adding value and not creating unnecessary levels of false positives.

4.2.4. Educate

User education should not be overlooked. While user diligence will never replace technical control, users should be provided with the knowledge not just of how to avoid attacks but also how to escalate issues when needed. When establishing programs, ensure that education is delivered on a continual basis and in a variety of formats. People learn in different ways but repetition is essential if the knowledge is to be retained.

5. Conclusion

Attackers once focused their efforts on targeting corporate servers, looking for gaping holes that would give them the keys to the kingdom. As enterprise servers became more secure, the clients became the target, especially web browsers - which have a questionable security track record at best. Today, many of the attacks targeting browsers are naked attacks, requiring no browser vulnerabilities to be effective and thus immune to standard protective measures like patching. The interconnected reality of the web ensures that risk is not isolated between web browser and server.

Some of the attacks discussed in this paper expose user data due to vulnerabilities in the web applications that they have been exposed to. Other attacks simply abuse intended functionality by using it in an unintended way. For this reason, an increasing number of attacks on web browsers will succeed, even when the browser in question is fully patched and hardened. As a result, enterprises must take revisit how they implement web security. Patch management is no longer enough to protect the browser and prevent unauthorized access to the keys to the kingdom.

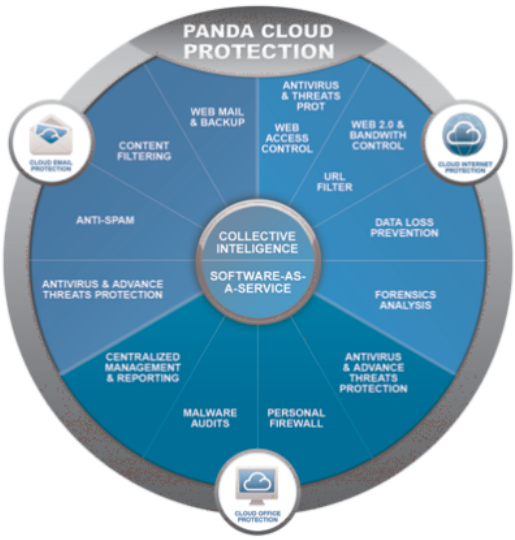
6. Panda Cloud Protection suite

Panda Cloud Internet Protection is part of the Panda Cloud Protection suite which is a complete SaaS security solution that protects all the main threat entry points: endpoint, email and Web traffic, against malware, spam, phishing, cross-site scripting and other advanced Web 2.0 attacks, through a light, secure and simple solution.

As the security suite is based in the cloud it offers maximum protection, while optimizing costs and productivity. Startup is immediate and the solution is simple to manage through an intuitive Web console.

The Panda Cloud Protection suite harnesses the power of Collective Intelligence. Panda's cloud-based Collective Intelligence leverages 21 terabytes of knowledge and experience drawn directly from millions of users to deliver comprehensive, instantaneous, non-intrusive real-world protection against known and unknown malware to all users.

Panda Cloud Protection leverages the power of the cloud to not only provide up-to-the-minute protection against known and unknown threats but also to streamline the delivery of that protection through the anytime, anywhere power of the Cloud Management Console.



7. References

1.
http://www.owasp.org/index.php/OWASP_Top_Ten_Project
2.
http://www.whitehatsec.com/home/resource/stats.html
3.
http://news.nefcraft.com/archives/2008/01/08/italian_banks_xss_oportunity_seized_by_fraudsters.tml
4.
http://www.washingtonpost.com/wp-dyn/content/article/2007/12/19/AR2007121900781_pf.html
5.
http://en.wikipedia.org/wiki/Samy_(XSS)
6.
http://www.macromedia.com/support/documentation/en/flashpalyer/help/settings_manager.html
7.
http://ha.ckers.org/blog/20081007/clickjacking-details/
8.
http://msdn.microsoft.com/en-us/library/cc004337(VS85).aspx

PANDA SECURITY

EUROPE
Ronda de Poniente, 17
28760 Tres Cantos. Madrid. SPAIN

Phone: +34 91 806 37 00

USA
230 N. Maryland, Suite 303
P.O. Box 10578. Glendale, CA 91209 - USA

Phone: +1 (818) 5436 901

www.pandasecurity.com