

JavaScript Infection Model

By Aditya K Sood and Richard J. Enbody

Join the Discussion
Connect

Advancements in Web 2.0 technologies have enhanced Internet functionality but at the same time have created numerous threats to the World Wide Web. This paper talks about the negative nature of JavaScript, which is exploited heavily by malware writers to spread infections throughout the online world.

Abstract

Advancements in Web 2.0 technologies have enhanced Internet functionality but at the same time have created numerous threats to the World Wide Web. The biggest issue the online world is grappling with is web malware, which is an outcome of intensive exploitation of web vulnerabilities. This paper talks about the negative nature of JavaScript, which is exploited heavily by malware writers to spread infections throughout the online world.

Exploitation Shift

There is always an element of discrepancy present between current and upcoming technologies. With the advent of powerful operating system protection mechanisms, the attack surface has shifted to web exploitation vectors because memory exploitation is becoming tougher for the attackers. Technologies such as Microsoft Data Execution Protection (DEP),¹ Address Space Layout Randomization (ASLR),² and GS cookies³ have circumvented the attack and exploitation of system-level vulnerabilities. The use of string functions is completely isolated from systems as they are considered as a base for buffer overflow attacks. Exploitation has shifted from system vulnerabilities to web vulnerabilities.

The attack landscape of the Web has a panorama of exploitations that are proliferating day by day. With the rise of blogs, wikis, atom feeds, RSS, and others, the insecurity level is in-

creasing⁴ in spite of versatile functions. These new technologies have made the Web flexible and robust by allowing the inclusion of content from third-party sites and sending content to other domains. In reality, data from the third parties cannot be verified against presence of potential malware. As a result malware can accompany the data back into the parent website without restriction and continue spreading across the Web. Security considerations have to be undertaken in the best possible manner to combat web exploitation.

New technologies

With the advent of new technologies, the sphere of attack surface vulnerability has widened. The Web is getting exposed to identity theft, exploitation, scams, phishing, redirection vulnerabilities, cross site scripting (XSS), and cross site request forgery (CSRF).⁵ CSRF, for example, is a type of attack in which HTTP requests are sent in a stealth manner without the knowledge of user. This type of attack allows the attacker to execute commands and requests on user's behalf. The inherent vulnerabilities in web applications are exploited by various application injections such as PHP, ASP, LDAP, SQL, and DOM (Document Object Model).⁶ The injections are widely used to manipulate the content, steal information, and spread malware. One step ahead is HTTP Protocol manipulation comprising of attack type Response Splitting,⁷ which bypasses browser protection mechanisms by splitting the HTTP response from the server thereby fooling the browser to interpret two responses instead of one.

1 Data Execution Prevention, <http://support.microsoft.com/kb/875352>.

2 Address Space Layout Randomization, <http://blogs.technet.com/b/security/archive/2006/05/26/430538.aspx>.

3 GS, <http://blogs.technet.com/b/srd/archive/2009/03/20/enhanced-gs-in-visual-studio-2010.aspx>.

4 RSS Attacks, <http://www.techspot.com/news/20098-increased-rss-malware-attacks-predicted.html>.

5 Cross Site Request Forgery, https://www.isecpartners.com/files/CSRF_Paper.pdf.

6 DOM XSS, <http://www.webappsec.org/projects/articles/071105.shtml>.

7 HTTP Response Splitting, <http://www.securiteam.com/securityreviews/5WP0E2KFGK.html>.

The cropping up of Web 2.0 and AJAX with JSON (JavaScript Object Notation) has transformed the structure of Web. Primarily, AJAX is used to interact with a specific set of events in a webpage through asynchronous JavaScript calls, which are usually scattered throughout the webpage. AJAX allows a number of events to be executed in a single page from different domains. Content validation is a big problem because content is fetched from various sources. Further, AJAX serializes all types of data elements into strings, which are used by programming interfaces such as ASP, ASP.NET, etc. Basically, serialization converts data into a stream. It also helps build dynamic scripts in a backend channel, passing information from third-party servers to the browser DOM for execution. These factors can be utilized collectively to misuse the AJAX technology⁸ and the techniques are proliferating in the wild. Numerous attacks can be initiated with new parameters and web technologies. The educational community, financial institutions, banks, companies, etc., are all structured over the Web. Browsers are interdependent on a number of components and sometimes it is hard to determine the impact of a vulnerability. Are we ready to handle such attacks? Are our defense mechanisms ingenious enough to thwart Web 2.0 attacks through AJAX and applied policies?

Web malware – real world scenario

Web malware is infecting websites at a rapid pace. The reasons can be security negligence, unpatched vulnerabilities, administration problems, etc., but the impact is growing exponentially. In the last few years, the security community has detected an overwhelming increase in malware using JavaScript.^{9 10} The malware problem, however, is not restricted to one business entity but has encompassed all the spheres, thereby resulting in loss of business and compromising the stability and robustness of organizations. Dasient¹¹ has published stats which present the sphere of infection by web malware. Business is the most exploited entity because of the inherent money element involved in it – there is money to steal.

Understanding the JavaScript Infection Model

JavaScript is one of the most susceptible scripting language used for malware infection by most of the attackers,¹² especially since JavaScript is used extensively in websites and applications for accessing various HTML elements and objects dynamically. JavaScript provides more robust control and dy-

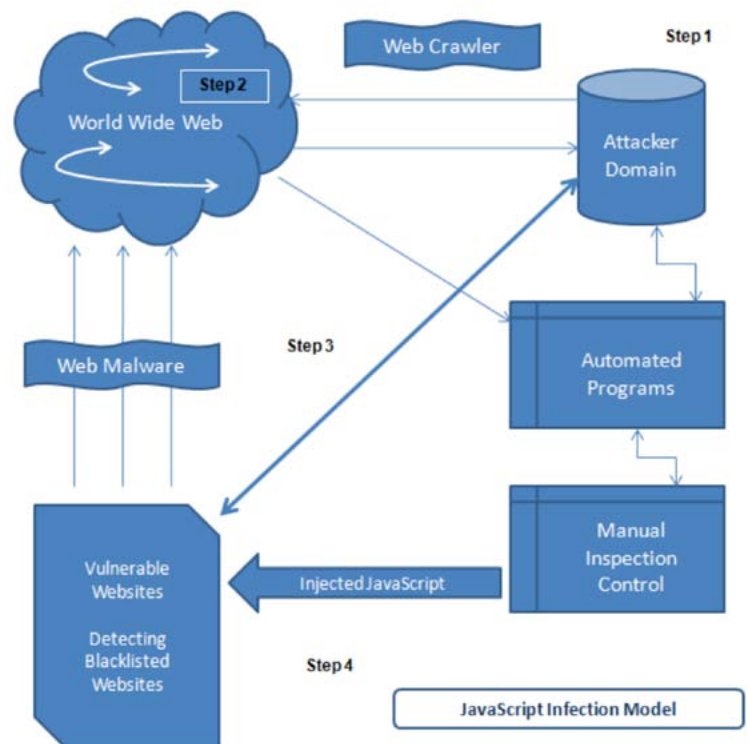


Figure 1 – JavaScript Infection Model

namic operations, which can be performed on the vulnerable websites and applications. Every browser uses JavaScript as a default scripting language and content is rendered without any notification because this scripting language is a working standard in Web 2.0. That's the main reason malware writers use JavaScript to conduct stealth attacks by exploiting the in-built functionality of browsers.

It is always advisable to understand the benchmark of infection strategies used by the attackers to launch extensive attacks on the public websites or social networking environments. The JavaScript Infection Model (JIM) reflects the generic methods opted by the attackers to launch malware. The model itself clarifies the structure and basic thinking of the attacker to inject malware in the Web, thereby resulting in large-scale infection. The overall model is presented in Figure 1. The following steps describe the model in detail.

1. Attacker controlled domain

The base is the attacker's controlled domain, which hosts a number of malware/programs that are used for malicious purposes. This step is undertaken to ensure that there is a centralized repository of JavaScript malware. The infection vector can be a single website or a number of websites in the domain. The attacker's controlled domain also has custom designed web spiders for collecting information from the Web. Attackers always look for vulnerabilities in websites to conduct injections so that malicious content can be included into the victimized websites.

2. Detecting blacklisted and vulnerable websites

The second step is to scrutinize vulnerable websites and domains which possess inherent application vulnerabilities that

8 Malware using AJAX, <http://blogs.securiteam.com/index.php/archives/734>.

9 Growth in Web Malware, http://blog.dasient.com/2010/09/continued-growth-in-web-based-malware_9357.html.

10 Malware Stats, http://wam.dasient.com/wam/infection_library_index.

11 Dasient, http://blog.dasient.com/2010/09/continued-growth-in-web-based-malware_9357.html.

12 JavaScript opens doors to browser-based attacks, http://news.cnet.com/JavaScript-opens-doors-to-browser-based-attacks/2100-7349_3-6099891.html.

can be exploited to spread malware. As most malware spreading attacks are automated, an attacker should have the information of blacklisted websites in order to make the malware attacks more successful. Automated spiders accumulate information about a domain. This is a very important step from malware writer's perspective because if the website is blacklisted, it gives an indication of the fact that most browsers will not visit that website. As a result, it hampers the malware infections through JavaScript. This information required by the attacker depends on the capability of that spider. As one knows the Web is an open forest: there is no stringency in accessing the front end of websites directly because it is the Web's default nature until and unless custom security controls are implemented.

3. Designing attack vector

After detecting the blacklisted domains, the crawlers push the information back to the attacker's controlled domain for scrutinizing the contents. During this process malicious scripts are not injected into blacklisted domains; rather new sets of websites having vulnerabilities are used as an attack point. The crawling process is repeated to find new domains having security vulnerabilities. The information can be scanned in an automated manner or manually; it depends on the attacker's strategy how the process will be carried out. For example, user agent strings provide information about the type of browsers used and other custom software that are used in line with the browsers. This indirectly helps the attackers a lot in crafting an attack by exploiting web vulnerabilities in the new domains. If we talk about the manual ways, then the attacker himself can look into the vulnerable websites for latent vulnerabilities. Again, it depends upon the attack vector, which requires a specific set of vulnerabilities to be exploited for spreading malware.

4. Exploiting targets

The information is further looked upon by matching the indexed vulnerabilities on the attacker's domain to explore direct vulnerabilities that can be exploited. If not, the attacker tries to inject malicious iFrames in the context of the domain with the source pointing to malicious JavaScript hosted on the attacker's domain or third-party infected domain. Once the JavaScript is injected, the code is changed as per requirements to infect the systems in a versatile manner. Lastly, the attacker controls the vulnerable websites hosted on a domain and changes them into a malware spreading entities. The attacker's surface becomes diversified when more victims fall into the trap.

Many of the online tools used by malware writers primarily test the weaknesses and entry points through which JavaScript can be injected. Because JavaScript is dynamic and robust in its working on the Web, it is the preferential choice of malware writers to conduct attacks and spread malware.

Spreading malware: JavaScript functionality

JavaScript is used extensively in spreading malware¹³ due to its inherent nature to provide direct access to browser components. A JavaScript infection pattern is followed by most malware whether system-resident or web-based to hide the objects used for infection. The concept of hidden infection is not new but is used in a randomized manner. This is a very effective technique to launch malware attacks in a stealthy manner to exploit the user parlance to understand what is happening at the backend. There are definitive methods that are used incessantly by web malware to infect victim machines. Let's have a look at the hidden structure which is used intensively.

Remote scripting with hidden iFrames

The HTTP specification allows the effective use of <iframe> in a substantial manner to embed one webpage into another webpage, irrespective of the domain to which a page belongs, and can be used in cross-domain context. This functionality of iFrames is exploited by malware writers in order to infect systems. Since these are interactive in nature, it is possible to bypass Same Origin Policy (SOP) easily to launch cross-domain attacks if a certain set of vulnerabilities exist in the base software or in web applications. SOP defines a control policy on scripts that are originating in a same domain to access properties and objects on various webpages in that domain. SOP applies restrictions on scripts in order to prevent access to HTML objects and properties of webpages on different domains and websites. SOP is completely browser-based, and due to inherent vulnerabilities¹⁴ in browsers it can be bypassed easily. This factor relates not only to the HTTP specification but also to the browsers as to how effectively SOP is implemented. It also depends on the rendering engine when the objects are rendered and access is required by them to perform various functions.

```
<iframe src="http://www.malicious.com" width="1"
height="1" style="visibility:hidden;position:absol
ute"></iframe>
```

```
<iframe src=" http://www.malicious.com" width="0"
height="0"></iframe>
```

The above presented iFrame code is used heavily in spreading malware. The beauty of this code is the fact that the iFrame becomes hidden and cannot be displayed on the infected website. The hidden iFrame sends a request to malicious.com for downloading malware or executing rogue JavaScript.

Malware writers exploit this functionality to a greater extent to serve malware by infecting websites with iFrames which can be used to serve hidden malware through infected websites. It is also possible to update databases with malicious iFrames as stored elements to make iFrames persistent in na-

13 JavaScript Malware, <http://www.darkreading.com/security/app-security/showArticle.jhtml?articleID=208803932>.

14 Mozilla SOP bypass Vulnerability, <http://securitytracker.com/alerts/2010/Jul/1024228.html>.

ture. This type of attack depends on the XSS vulnerability being present in the website, depending on the design of the application and website which makes the XSS persistent or reflective in nature. Nowadays, SQL-injection attacks have been used in conjunction with XSS to upload malicious iFrames into a database through a vulnerable website. The iFrames are passed as payloads in the form of hexadecimal strings,¹⁵ which get stored in the web database. It remains persistent in the database and is retrieved when a website issues some database query. This tactic is used heavily in mass SQL-injection attacks that spread malware through drive-by-download attacks.

JavaScript heap spraying

The inherent software vulnerabilities that can be exploited through JavaScript heap spraying¹⁶ are also a major source of malware infection. The browser-based exploits are used extensively to trigger malware infection on the client side. If a vulnerable version of the browser is running or there is presence of vulnerable software, the malware writers try to exploit the vulnerability in order to execute custom shellcode.

Heap spraying is a type of attack which exploits memory corruption vulnerabilities through browsers without any hassles. The target is unsafe applications. The functional part is to create and allocate arbitrary objects in the heap by using a type-safe language. The objects used to fill heaps contain dangerous exploit codes to be run against the target system. Specific notions used for these types of exploits are mentioned below:

- Design a string which grows exponentially by concatenating with itself
- Keep string format as Unicode to avoid any type of stringency in the code
- Exhaust the maximum length that is allowed by the scripting engine
- Shellcode is placed at the end of the string
- The set of code is copied over a large sum of arrays which can control the execution
- The attack surface is potentially created so that it can have enough memory for exploitation
- Heap blocks are considered to be on the same locations every time

The above mentioned steps are used to exploit the target through the browser. The security community has witnessed exorbitant growth of these browser-based exploits in the recent years.¹⁷ These attacks exploit the vulnerability in browsers by manipulating the default heap using JavaScript, used to create heap blocks within a particular memory range and dynamic image objects generated to trigger the vulnerabil-

ity. Image objects have a “src” parameter and are pointed to URLs that exploit the vulnerability in Firefox. JavaScript heap blocks constitute the shellcode with call and jmp instructions. The shellcode may result in downloading of malware.¹⁸

Obfuscation and hybrid codes

Code obfuscation is used at a very large scale for spreading malware. The preferable scripting language is, of course, JavaScript. But obfuscation is not restricted to only JavaScript and using only escape functionality and generic encoders. Obfuscation is used extensively to make code hybrids nowadays. There can be a scenario in which two scripting languages are used together. The malware spreading can be done very easily through hybrid codes because it becomes hard to analyze the code which is encoded with custom encoder and using JavaScript functions to get downloaded into the system. For example, custom JavaScript decoders can be used to decode the JavaScript-related functions, but in order to decipher the payloads other extensive tools are required. PERL is a good tool that is used in a wide manner to decode hybrid codes to understand the actual payload which is exploiting the system.

Widgets

The Web 2.0 working model uses widgets at a very large scale. Generally, these small chunks of code are used for enhanced functionality, primarily for advertisement purposes, including content from third parties, notification alerts, etc. They are based on the concept of code reuse. Widgets are provided by the advertisement agencies, news portals, companies, etc., to include content for the user website experience. The vendor offers widgets that can be added manually or automatically in the user websites or blogs, depending on the feasibility of environment. In order to accept data from third parties, the widget provides a communication and content transfer interface between the parties. Once installed in the user website, it opens the communication channel between the parent node and the child node. JavaScript widgets are used heavily for infecting websites because the chunked code sits as HTML in the parent page and follows the same hierarchy functions as the parent node.¹⁹ It means the code becomes in line with the main module and is executed as the webpage is loaded into the browser. Widgets may contain malware which redirects the website to a malware domain or starts downloading the malware executables into the system. So JavaScript code in the form of widgets is used to spread malware through the Web thereby impacting the large sets of users who are using that specific widget. The widgets can be uploaded to malware domains or content delivery networks to follow the chain process of infecting large number of websites.

15 SQLXSSI Attacks, <http://www.slideshare.net/adityaks/owasp-app-sec-us-2010>.

16 Heap Spraying, <http://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf>.

17 IE Object Memory Corruption, <http://www.exploit-db.com/exploits/930>.

18 Mozilla “Host: Heap Buffer Overflow Exploit,” <http://www.exploit-db.com/exploits/1224>.

19 Web Widget Infection, <http://blog.dasient.com/2010/06/third-party-javascript-widget.html>.

Drive-by downloads

Drive-by Download²⁰ is one of the most long-lived techniques of spreading malware through JavaScript by forcing victims to download malware without their consent and knowledge. The malicious website usually hosts a number of hidden iFrames that point to the malicious program or malicious Active X control. Essentially, the website forces the user's browser to install the plug-in directly on the system. How does the drive-by download work? Let's consider as generic case. The website offers a browser plug-in to install directly into the context of system. There is a lot of functional difference between a normal plug-in and Program Loader Stub (PLS) which is installed as result of the plug-in. Program loader stub defines the nature of execution of plug-in. The main function of program loader stub is to call functions in a hierarchical manner and synchronously. In general terms, the PLS is responsible for calling API's appropriately that are used by plug-ins to download malware. The PLS loads each DLL that is required by the plug-in into the memory space. It fills in the structures in the import directory of the executable in the memory. If the DLL is not loaded in the memory, the PLS fails to load the malicious executable. The stub actually downloads a lot of malicious content into the user's system by remaining silent and infecting the system at large in a stealth manner. The plug-in offers fake information to convince the user it is valid:

1. Fake certification and end user license agreements (EULA) links aiming to exploit the user's ability to make a decision about the software.
2. Incessant and rogue security warnings to exploit the user's inability to decide the working of software plug-in.

If the required plug-in is allowed to install in the system, a stub is installed simultaneously, which further downloads a number of programs without user consent. The installed programs infect the system as mentioned below:

- Infection at the folder level by placing malicious files. The malware programs follow the concept of cross linking in which one malicious binary is interfaced with another malicious binary with randomized names. This becomes hard for a normal user to detect the real working of the binary present in the requisite program files folder.
- Inserting malicious, e.g., porn website, links in the browser bookmarks and shortcuts to be placed in the program bars on the desktop that further links malicious website and malware programs respectively. The main criterion of this type of functionality is to infect the generic places in the systems which are used by the victims easily.
- It primarily hijacks the default browser links and web-pages and points to a malicious website or custom-

designed search engine. If a victim uses that search portal to run a query, the resultant links are the most vulnerable links and directed towards malicious web servers. This happens quite often.

- Generation of advertisement pop ups in the system continuously. This can be a time-based logic to generate a pop up after a requisite interval and running with the same behavior for a longer duration.

The infection does not stop here as such. If a victim tries to remove the required set of files, a cleanup link is also provided which further downloads another plug-in having well-defined EULA and privacy restrictions. The attackers follow the same technique but the overall functional perspective is different. It is kind of dual entrapment and the directed website looks like a cleaning solution vendor website, but typically it is fake. As a result, the victim is trapped again and the system gets infected with a different layout.

Conclusion

Malware running rampant in the wild is an outcome of flaws and vulnerabilities that exist in web applications and websites that make up the Web. The same language tools are used for development AND exploitation. This paper has presented JavaScript from the malicious infection point of view. In order to defend against these web malware we have to design a number of defenses and protections in the same pattern. Since JavaScript is the major scripting language to add dynamic functionality to webpages, the same is used for dynamic generation of malware. So understanding the diverse nature of JavaScript, we can design protection mechanisms to combat its misuse and exploitation by malware writers.

About the Authors

Aditya K Sood is a security researcher, consultant, and Ph.D. candidate at Michigan State University. He has worked in the security domain for Armorize, COSEINC, and KPMG and founded SecNiche Security. He has been an active speaker at conferences like RSA, TRISC, Hacker Halted, ExCaliburCon, EuSecwest, XCON, OWASP AppSec, Security-Byte, CERT-IN and has written content for HITB Ezine, Hackin9, Usenix Login. He may be reached at adi_ks@secniche.org.



*Dr. Richard Enbody is an Associate Professor in the Department of Computer Science and Engineering, Michigan State University. He joined the faculty in 1987 after earning his Ph.D. in Computer Science from the University of Minnesota. Richard's research interests are in computer security, computer architecture, web-based distance education, and parallel processing. He has two patents pending on hardware buffer-overflow protection, which will prevent most computer worms and viruses. He recently co-authored a CS1 Python book, *The Practice of Computing using Python*. He may be reached at enbody@cse.msu.edu.*



²⁰ Drive by Downloads, <http://www.spywarewarrior.com/uiuc/dbd-anatomy.htm>.