Defeating Drive-by Downloads

Robert Charles Metzger

November 15, 2016

The term “Drive-by downloads” refers to the unintentional download of a virus or malicious software onto a client system.

There are several ways to categorize defenses against drive-by downloads. The first is by the indicators that they look for to distinguish benign web pages from malignant ones. The papers reviewed here use the following indicators:

1) Javascript string constants with unprintable characters,
2) Javascript strings constants interpreted as x86 instructions,
3) Javascript internal representation (bytecode),
4) Javascript language constructs,
5) Browser file download events,
6) Browser environment changes,
7) Communication between browser modules,
8) Browser file download reasons,
9) DOM change methods,
10) HTML tags used in drive-by-downloads,
11) URL chains, and
12) URLs of malware distribution networks.

The second way to categorize defenses is by the algorithms they use for matching the indicators that they look for with the same indicators found in known drive-by-download pages.

The third way to categorize defenses by whether they execute offline, in pseudo real-time, or in true real-time. We take this approach.

Offline solutions analyze potential threats by crawling web pages or analyzing network traffic completely independently of user working with web browsers. The first benefit of an offline solution is that it can employ all the server and network resources you wish. The second benefit is that it can be proactive, searching out potential attack sites before users even become aware of them. The first drawback of offline approaches is that they don’t have the entirety of the HTML and Javascript that will be processed by the browser. Thus, they can’t find certain types of problems. The second drawback is that they need a suspect list or guidance on where to look for problems.

Pseudo real-time solutions run in a server infrastructure or web proxy. They analyze web pages as they are requested and the requests are serviced. They don’t forward malicious pages to the clients who request them. There are two benefits of pseudo real-time

approaches. First, the cost of performing the protection analysis can be amortized over multiple requests for a web page, since it is performed once on a server or proxy. Second, the analysis is not performed on the client, which means that there will be no incremental processing time requirement on the client. The drawback of the pseudo real-time approach is that it doesn't have the entirety of the HTML and Javascript that will be processed by the browser. Thus, it can't find certain types of problems.

A true real-time approach analyzes each page as it arrives at the client that requested it. Static methods can analyze the parts of the web page as it arrives, and dynamic methods can evaluate the final page and associated scripts as they are interpreted by the browser. The benefit of the pure real-time approach is that the analysis can work on the final version of the HTML and Javascript that will be processed by the browser. The drawback of this approach is that all of the resources expended for the analysis must be on the client. If an analysis method takes too much time, either it will reduce the user's productivity, or it will have to be abandoned.

Real Time Defenses

1. Egele, M., Kirda, E., & Kruegel, C. (2009). Mitigating drive-by download attacks: Challenges and open problems. In *iNetSec 2009–Open Research Problems in Network Security* (pp. 52-62). Springer Berlin Heidelberg.

This work describes a client-based, dynamic tool that analyzes Javascript when a browser loads a page. They created a prototype and integrated it into Firefox browser and its Javascript processor. They believe that the best distinguishing characteristic of drive-by-downloads is Javascript string constants. In a drive-by download, these constants are filled with characters which can be interpreted as machine code.

To detect an attempt to exploit a memory corruption vulnerability, their system monitors the values of string constants allocated by the Javascript processor. Their implementation adds code to each point in the Javascript interpreter where it creates string variables. They make a memo of the memory location and length of the values used to construct the string. After capturing the strings, a library that performs x86 instruction emulation and shellcode detection processes them.

The authors evaluated the effectiveness of their system by applying it to 4500 pages known to be benign, from the Alexa ranking of global web sites. It did not classify any of these as malicious, thus false positives were zero. Then they evaluated their system with 11,900 URL's from websites known to implement drive-by-downloads, which were packaged as 1187 traces. Of these, 956 were classified as having shellcode, which meant it was 81% effective.

2. Cova, M., Kruegel, C., & Vigna, G. (2010, April). Detection and analysis of drive-by-download attacks and malicious JavaScript code. *In Proceedings of the 19th international Conference on World wide web* (pp. 281-290). ACM.

This work describes a client-based, dynamic tool that analyzes web pages as possible drive-by-downloads. They created a prototype and integrated it into HtmlUnit, which is a set of Java classes that emulate a browser. They support Javascript with the Mozilla Rhino interpreter. They believe ten features distinguish Javascript which implements a drive-by-download. They categorize these features into two kinds: five are necessary to exploit browser vulnerabilities, and five are merely useful. The features that they consider essential are these:

1) The number of bytes allocated in string operations,
2) The number of string constants that have non-printable characters,
3) The number of instantiated browser components (plugins),
4) The values and types of parameters to methods of instantiated components, and
5) The sequences of calls to methods of instantiated components.

The authors implemented their approach in a tool they call JSAND and validated it using more than 140,000 web pages. The tool identifies malicious pages by comparing an "anomaly score" to that 20% more than the anomaly score of a test data set. They ran their tool on a set of over 115,000 web pages, and it indicated that 137 were malicious. The authors verified that 122 of these did start drive-by-downloads, and the remaining were false positives. They ran their tool on a set of 823 known drive-by-download pages. It failed to identify two of them, which were false negatives.

3. Song, C., Zhuge, J., Han, X., & Ye, Z. (2010, April). Preventing drive-by download via inter-module communication monitoring. In *Proceedings of the 5th ACM symposium on information, computer and communications security*(pp. 124-134). ACM.

This work identifies malignant web pages using intermodule communications within the browser. The authors say that exploitations involve invoking vulnerable browser modules, and that when those modules are invoked, the obfuscation of the malignant Javascript code has been resolved. They built a prototype and integrated it into Microsoft Internet Explorer. During a browsing session, it monitors the communications between "vulnerable" modules. Then it checks the communication patterns against vulnerability signatures, which they derived manually.

The authors implemented their approach in a tool they call JSAND and validated it using more than 140,000 web pages. The tool identifies malicious pages by comparing an "anomaly score" to that 20% more than the anomaly score of a test data set. They ran their tool on a set of over 115,000 web pages, and it indicated that 137 were malicious. The authors verified that 122 of these did start drive-by-downloads, and the remaining were false positives. They ran their tool on a set of 823 known drive-by-download pages. It failed to identify two of them, which were false negatives.

4. Hsu, F. H., Tso, C. K., Yeh, Y. C., Wang, W. J., & Chen, L. H. (2011). Browserguard: A behavior-based solution to drive-by-download attacks. *IEEE Journal on Selected areas in communications,* 29(7), 1461-1468.

This work identifies drive-by downloads by collecting every file that is downloaded to a client through a browser, and analyzing the reason why the file was fetched. The authors implemented their ideas as a Microsoft Browser Help Object and integrated it into Internet Explorer.

The valid reasons for downloads include:

1) Fetching files needed to display the page.
2) Fetching the source of a different web page, after the user clicks a link.
3) Fetching a file after the user clicks a download button in a download dialog.
4) Fetching a file generated after the user clicks the link to an ASP/PHP/JSP file.

If the download did not occur for one of these reasons, it is malignant. The system has a process that maintains a "whitelist" and "blacklist" of processed files. Initially, it puts downloaded files on the blacklist, to prevent execution. After confirming a valid reason for the download, it is released for execution.

The authors evaluated the system by visiting the top 500 websites from the Alexa list, using the IE browser with BrowserGuard active. There were no false positives on these sites. They used the MetaSploit framework to generate 10 malicious web pages, using 9 known exploits for Internet Explorer 7.0. BrowserGuard prevented all of them from executing, yielding zero false negatives.

5. Cherukuri, M., Mukkamala, S., & Shin, D. (2012, October). Similarity analysis of shellcodes in drive-by download attack kits. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2012 8th International Conference on (pp. 687-694). IEEE.

This work identifies drive-by-downloads by comparing the shellcodes in known attack kits with the string constants in unknown web pages. They disassemble the string constants used as shellcodes, and count the number of unique x86 opcodes in each shellcode sequence. These counts are used as a vector of features. They use three statistical measures to compare the shellcodes from attack kits with strings found in suspect web pages: Cosine Similarity, Extended Jaqard Similarity, and Pearson Correlation.

The authors collected 15 attack kits in use at the time the paper was written. They collected the JavaScript strings from these kits which were 1500 characters or longer, based on the mean length of those strings. They collected benign strings from the top 10,000 websites from the Alexa list. First, the authors compared the attack code strings with each other. They found that those shellcodes were similar to one or more shellcodes from a different shellcode from a different attack kit 68% of the time. They found only minor variations in the similarity measures for attack kits released in different years. To evaluate their hypothesis, they compared 100 randomly selected strings from the large benign set to

the malicious set. The mean of the maximum similarity measures in these comparisons was 25%, which indicates that the shellcodes are very different from benign character constants.

6. Cao, Y., Pan, X., Chen, Y., & Zhuge, J. (2014, December). Jshield: towards real-time and vulnerability-based detection of polluted drive-by download attacks. In *Proceedings of the 30th Annual Computer Security Applications Conference* (pp. 466-475). ACM.

This work identifies exploits used in drive-by-downloads by matching the internal representation generated by the Javascript processor from the Javascript written by the attackers. This internal representation is a sequence of opcodes and parameters. They manually encode signatures of known attacks into two formats. The first encoding is used for initial screening using regular expressions, and the second encapsulates control flow and data flow. They use two different mechanisms to match their indicators with problematic web pages: regular expressions for initial matching, and a deterministic finite automaton (DFA) with recording of control and data flow state changes for final matching.

The authors see two uses for their system, named JShield. It can run on Web Application Firewalls or Web IDS/IPS. It can also be used to provide a malware scanning service. It is robust enough to be deployed at Huawei, a large telecommunication company, inspecting traffic between their switches and routers for malicious content. The authors evaluated the top 100 websites from the Alexa list, and there were no false positives.

7. Cherukuri, M., Mukkamala, S., & Shin, D. (2014). Detection of shellcodes in drive-by attacks using kernel machines. *Journal of Computer Virology and Hacking Techniques*, 10(3), 189-203.

This work identifies exploits by building signatures from x64 opcodes found in character constant strings in Javascript. They represent these as vectors of counts of each unique opcode. They identify Javascript that is attempting exploit a defect by applying a Support Vector Machine (SVM). The vectors are counts of opcodes, and the samples used for training came from exploit shellcode examples and benign strings from regular Javascript code

The authors performed experiments with SVM classifiers constructed by training with samples of 50%, 60% and 70% of the total samples. The SVM accuracies for these training sets were 99.47%, 99.55%, and 99.50%. The false positives were 29, 20, and 13 respectively, and the false negatives were 24, 16, and 17 respectively.

8. Cherukuri, M., Mukkamala, S., & Shin, D. (2014, October). Detection of plugin misuse drive-by download attacks using kernel machines. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom),* 2014 International Conference on (pp. 546-553). IEEE.

This work identifies problematic Javascript using a vector of 18 features that characterize calls to browser plugin functions. These features include number of times a method is called,

number of arguments passed, types of arguments passed, values of various argument types, counts of printable and non-printable characters in character string arguments, etc. They identify malignant Javascript by applying a Support Vector Machine. The vectors which are split into sets contain the 18 features. The benign training sets came from web pages which are on the Alexa frequently visited list and the malicious training sets came from known malware. The authors implement tracking of browser plugin functions using the Netscape Plugin Application Programming Interface (NPAPI). The work was embedded in HTMLUnit, which is a set of Java classes that emulate a browser.

The authors collected 22,000 web pages listed on Alexa which caused the browser to use plugins. They collected malicious use cases from exploit kits, Metasploit, and various active malicious web pages. The dataset contained 37,369 benign uses and 10,239 uses. The authors performed experiments with SVM classifiers constructed by training with 50% of their use cases and testing with the other 50%. The recognizer achieved an accuracy of 99.4%, with a false positive rate of 0.03% and a false negative rate of 1.15%.

9. Jayasinghe, G. K., Culpepper, J. S., & Bertok, P. (2014). Efficient and effective realtime prediction of drive-by download attacks. *Journal of Network and Computer Applications*, 38, 135-149.

This work identifies malicious Javascript using the Javascript internal representation, called bytecodes. They generate the entire bytecode sequence, and then create fixed length vectors by moving a window over the bytecode stream. The implementation extracts the function call from each bytecode command, and generates a new stream, eliminating the variability due to the arguments in the bytecode. Then they pass a sliding window over the stream, and extract fixed length vectors of symbols. These n-grams are represented as a vector space, in which each distinct n-gram is a dimension. The system uses a Support Vector Machine to determine whether the bytecodes implement and exploit.

The authors created a set of benign landing pages from 10,620 sites listed at the top of the Alexa list. They created the malicious pages from known drive-by-download sites, and by synthesizing samples based on examples in various malware forums and publications. There were 57 unique attack vectors in the set. The size of the n-grams was best between 1 and 4, depending on the machine learning algorithm used. The accuracy rate was 98.25% and the false positive rate was 0.03%.

10. Kishore, K. R., Mallesh, M., Jyostna, G., Eswari, P. R. L., & Sarma, S. S. (2014, February). Browser JS Guard: Detects and defends against Malicious JavaScript injection based drive by download attacks. In *Applications of Digital Information and Web Technologies (ICADIWT), 2014 Fifth International Conference on the* (pp. 92-100). IEEE.

This work uses three groups of indicators to identify malicious Javascript: static attributes from key HTML tags used, DOM change methods used, and dynamic code execution functions. The authors implemented a set of extensions to a browser. They use a custom set of rules to

evaluate the features that they collect to determine whether a Javascript code is malicious or benign. The rules were derived by inspecting examples of Javascript used in drive-by-downloads. When the modified browser receives an HTTP response, the JS Guard evaluates all of the incoming files required to render the page. Static monitoring extracts the properties of the vulnerable HTML tags, and determines if there are hidden behaviors or unauthorized redirections. Dynamic monitoring catches DOM changes and dynamic code execution in Javascript. It extracts the arguments and determines if there is malicious behavior. If either the static or dynamic monitoring finds a problem, it tells the user about the problem and offers the opportunity to continue or abort.

The authors report a false positive rate of 0.72% and a false negative rate of 9.04%, but give no information on their test dataset.

Pseudo-realtime Solutions

11. Rieck, K., Krueger, T., & Dewald, A. (2010, December). Cujo: efficient detection and prevention of drive-by-download attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference* (pp. 31-39). ACM.

The authors implemented an intermediary in a web proxy, which analyzes web pages and only delivers those it considers benign to the client. They believe that static and dynamic features must be used together to identify malignant web pages. The static analysis performs standard compiler lexical analysis. The dynamic analysis takes place in the Firefox Javascript processor. It records browser environment changes and execution of external events. They use machine learning for identifying problem pages. They map the syntactic analysis result into a series of fixed length windows. These are mapped into a vector space of unique strings, and analyzed with a Support Vector Machine algorithm.

They evaluated their system with 200,000 URL's derived from the Alexa list of most popular websites. It only identified two benign pages as attacks, which is a .001% false positive rate. They also evaluated CUJO using essentially the same set of drive-by-downloads pages as Cova. Their system achieved a 94.4% identification rate.

12. Lu, L., Yegneswaran, V., Porras, P., & Lee, W. (2010, October). Blade: an attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM conference on Computer and communications security* (pp. 440-450). ACM.

Unlike other approaches, this system executes outside of the browser, even though it runs on the client. They believe one essential characteristic identifies a drive-by-download. Their approach intercepts and prevents execution of any downloaded file, unless the user has consented to do so through the browser. Their system includes a Screen Parser, Hardware Event Tracer, Supervisor, Correlator, and I/O Redirector. The Screen Parser signals the Supervisor when the user is sees a dialog that will authorize a download. The Hardware Event Tracer connects mouse clicks or keyboard events to download authorization dialogs. The

Correlator links the download authorizations with downloaded files. The Supervisor puts the downloaded files in a secure zone until correlation. The I/O Redirector intercepts I/O operations from browser processes and sends them to the file in the secure zone, until it is known to be authorized.

The authors tested their system using 3,992 unique malicious URL's. These web pages were accessed multiple times over several days. BLADE blocked all attempted drive-by-download installs.

Offline Solutions

13. Matsunaka, T., Kubota, A., & Kasama, T. (2014, September). An Approach to Detect Drive-By Download by Observing the Web Page Transition Behaviors. In *Information Security (ASIA JCIS), 2014 Ninth Asia Joint Conference on* (pp. 19-25). IEEE.

This work identifies drive-by-downloads by finding suspicious link transitions. Their system deploys sensors in the browser on the client, in web servers, and in DNS servers. These sensors report transition information to a central server, which applies a custom algorithm to detect suspicious link transitions, presented in the paper.

They evaluated their work with the D3M 2013 dataset, created by the Anti Malware Engineering Workshop 2013. Their system has no false positives on this data, but a false negative rate of 24.2%.

14. Zhang, J., Seifert, C., Stokes, J. W., & Lee, W. (2011, March). Arrow: Generating signatures to detect drive-by downloads. In *Proceedings of the 20th international conference on World wide web* (pp. 187-196). ACM.

This work identifies drive-by-downloads by tracking the URL chains that are generated by the browser. They generated traces of URL information for the central servers in malware distribution networks, and use these chains to identify sequences that a browser would perform to fetch a drive-by-download. The authors built a system, which generates regular expressions to identify paths .

The authors generated the HTTP traces used for evaluation with a production cluster of high interaction client honeypots. The system only had 2 false positives when processing 10,733,282 HTTP traces.