Close encounters of the third kind

A look at the prevalence of client-side JavaScript vulnerabilities in web applications

Contents

Executive summary

In the past ten years, many whitepapers, research articles, and blog posts have been published on the subject of server-side web application vulnerabilities such as SQL injection, Cross-site scripting, and HTTP response splitting. In addition, several projects such as the WASC Web Hacking Incident Database[1] or the WASC Statistics[2] projects have tried to estimate the incidence of such issues in the real world. On the other hand, there is a dearth of information and statistics on the incidence of client-side JavaScript™ vulnerabilities in web applications, even though these vulnerabilities are just as severe as their server-side counterparts. We suspect that the main reason for this lack of information is that client-side vulnerabilities are harder to locate, and require deep knowledge of JavaScript and the ability to perform code review for HTML pages and JavaScript files.

As Web 2.0, AJAX applications and rich internet applications (RIAs) become more common, client-side JavaScript vulnerabilities will probably become more relevant, and we foresee a rise in the amount of such issues being exploited by malicious hackers.

This whitepaper presents the results of a research recently performed by the IBM® Rational® application security group into the prevalence of client-side JavaScript vulnerabilities. For this research, we used a new IBM technology called JavaScript Security Analyzer (JSA), which performs static taint analysis on JavaScript code that was collected from web pages extracted by an automated deep web crawl process. This kind of analysis is superior to and more accurate than regular static taint analysis of JavaScript code, as it includes the entire JavaScript codebase in its natural environment: fully rendered HTML pages and the browser's Document Object Model (DOM).

The research used a sample group of approximately 675 websites, consisting of all the Fortune 500 companies and another 175 handpicked websites, including IT, web application security vendors, and social networking sites. In order to avoid damage to the sites or interference with their regular behavior, we used a non-intrusive web crawler, similar to that of a web search engine, which retrieved approximately 200 web pages and JavaScript files per site from the application into a repository. These pages were then analyzed offline for client-side JavaScript vulnerabilities, using the JavaScript Security Analyzer, concentrating on two main types of issues: DOM-based Cross-site scripting, and Open redirects.
The results of our research were quite disturbing: about 14 percent (98 sites) of the 675 sites suffer from many severe client-side JavaScript issues, which could allow malicious hackers to perform attacks such as:

- Infect users of these sites with Malware and viruses.
- Hijack users' web sessions and perform actions on their behalf.
- Perform Phishing attacks on users of these sites.
- Spoof web contents

The troubling fact about these statistics is that most organizations have no efficient process or automated solution to assist them with the task of locating these types of issues.

Our research also showed that 38 percent of the vulnerable sites suffered from these vulnerabilities as a result of using third party JavaScript code such as:

- Marketing campaign JavaScript snippets.
- Flash embedding JavaScript snippets.
- Deep linking JavaScript libraries for Adobe® Flash and AJAX applications.
- Social networking JavaScript snippets.

Of the 98 vulnerable sites, 92 sites (94 percent) suffered from DOM-based cross-site scripting issues, whereas only 11 sites (11 percent) suffered from open redirects. The total amount of DOM-based cross-site scripting issues found was 2370, while only 221 open redirects were found.

Lastly, based on the dataset that we analyzed, we may extrapolate that the likelihood that a random page on the internet contains a client-side JavaScript vulnerability is approximately one in 55.

We would like to stress the fact that our research concentrated on only two issue types (DOM-based cross-site scripting and Open redirects), and was performed using the first version of the JavaScript Security Analyzer technology. Our analysis was run on a relatively small number of web pages, and was performed without digging deeply into each site (for example, no credentials were used to log in to the sites). We are quite certain that a more thorough web crawl and a longer list of client-side JavaScript issues to look for would reveal significantly more security vulnerabilities.

Technical details

In order to understand the difficulties involved in assessing web applications for client-side JavaScript issues, we must first understand how these types of issues differ from server-side web application vulnerabilities.

In 2005, Amit Klein, a distinguished security researcher, published a whitepaper called "DOM Based Cross Site Scripting or XSS of the Third Kind."[3] The paper discussed a unique variant of Cross-site scripting which, unlike "Stored" and "Reflected" Cross-site scripting, did not rely on user input being sent to the application and then reflected back in a web page, but instead exploited the fact that the vulnerable HTML page used information from JavaScript objects such as document.URL, document.location or document.referrer, all of which could be controlled by a malicious attacker in some way.

The paper presented the following vulnerable example HTML page, with the URL address: http://www.vulnerable.site/welcome.html

```
<HTML>
  <TITLE>Welcome!</TITLE>
  Hi
  <SCRIPT>
     var pos=document.URL.indexOf('name=')+5;
     document.write(document.URL.substring(pos,document.URL.length))
  </SCRIPT>
  <BR>
  Welcome to our system
  …
</HTML>
```

According to the original paper, a typical access to this web page would be via the following URL:
http://www.vulnerable.site/welcome.html?name=Joe
However, if this web page is retrieved via the following malicious URL: http://www.vulnerable.site/welcome.html?name=<script>alert(document.cookie)</script>
A Cross-site scripting condition occurs.

The whitepaper then described how this Cross-site scripting vulnerability works:

"The victim's browser receives this link, sends an HTTP request to www.vulnerable.site, and receives the above (static!) HTML page. The victim's browser then starts parsing this HTML into DOM. The DOM contains an object called document, which contains a property called URL, and this property is populated with the URL of the current page, as part of DOM creation. When the parser arrives to the JavaScript code, it executes it and it modifies the raw HTML of the page. In this case, the code references document.URL, and so, a part of this string is embedded at parsing time in the HTML, which is then immediately parsed and the JavaScript code found (alert(…)) is executed in the context of the same page, hence the XSS condition."[4]

The whitepaper also discussed how this third kind of Cross-site scripting could be used to mount attacks that evade server-side detection and prevention mechanisms, such as web application firewalls, by using the HTML fragment identifier (#), a fact that makes this type of vulnerability particularly dangerous.

Many security experts believe[5] that the task of locating client-side JavaScript issues such as DOM-based Cross-site scripting is a daunting one, often requiring that a penetration tester perform thorough code review of both the HTML and the JavaScript source code that is included with it. In addition, many believe that current automated methods for performing dynamic and static security analysis of web applications fall short, and are incapable of accurately locating most client-side JavaScript issues.

For our research we used a new technology, developed by the IBM Rational application security group, which is available as part of IBM Rational AppScan® Standard Edition software v8.0.[6] This technology is called JavaScript Security Analyzer (JSA), and works in the following way:

JSA goes over all URLs visited by the web crawler of Rational AppScan Standard Edition software, one by one. For each URL, JSA saves the entire HTTP response stream. JSA then looks for JavaScript entry points in the current visited URL, and applies a set of JavaScript-specific taint analysis rules. These rules include specifications of source, sink, and sanitizer functions. JSA reports on data flows from source to sink that do not go through a sanitizer. JSA reports on six different issue types. Issues reported by JSA appear in Rational AppScan Standard Edition software. Trace information for each issue is displayed in the issue information pane in Rational AppScan Standard Edition software.

Note that JSA runs entirely on the local machine, pulling visited URLs from the current scan, and performing no communication with the site at all. This makes it possible to run JSA on existing scan files, even if the scanned host is not available. The engine of JSA uses a sophisticated taint analysis algorithm, and is based on a static analysis platform developed by IBM research.

Modern websites, which use Web 2.0 and AJAX, often generate HTML and JavaScript code on the fly. This means that standard static code analyzers cannot fully scan the source code and locate client-side JavaScript issues, since the source code itself does not yet include the entire HTML and JavaScript code. On the other hand, because the input for the assessment done by JSA includes both the fully rendered HTML and the JavaScript code (both extracted by deep crawling of the website), client-side issues can be detected with superior accuracy.

In essence, JSA enjoys the best of both dynamic and static analysis, amalgamating the two approaches, in order to accurately assess JavaScript code in its natural environment.

For this research, we used a sample group of 675 websites, including all 500 of the Fortune 500 companies, plus 175 handpicked websites including IT security companies, web application security companies, social networking sites and other popular websites. We retrieved approximately 200 pages and JavaScript files per site, using a non-intrusive web crawling process that only follows HTML links, and executes JavaScript code in each page to find dynamically generated links and simulate real user interaction with the site, which is necessary for AJAX-type sites. To avoid damage to or any non-standard interaction with the sites, our web crawler did not fill out any HTML forms, did not log in to the application, and did not submit any additional HTTP requests to the sites. In essence, our web crawler merely performed an indexing of each application similar to that of web search engines.

Each application was tested for two main client-side JavaScript issues: DOM-based Cross-site scripting, and Open redirects[7] a vulnerability which allows a malicious attacker to force the victim's browser to automatically redirect to a site he/she owns, and which can be used for Phishing purposes.

Our research found that of the 675 websites analyzed, 98 (14.5 percent) were infested with DOM-based Cross-site scripting and open redirects (Figure 1).

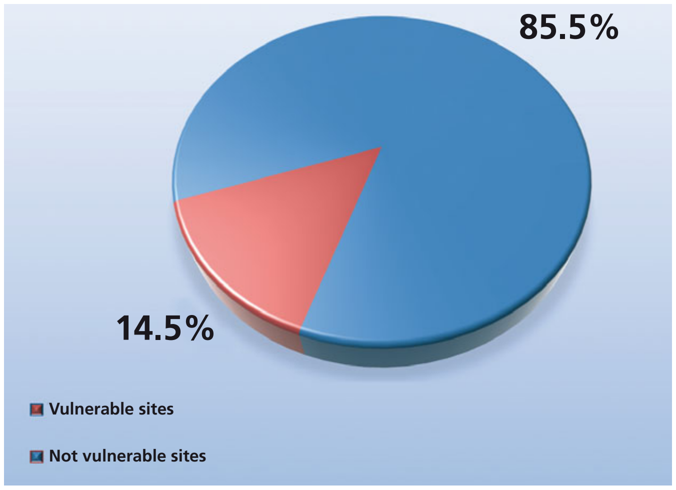


Figure 1: Percentage of sites vulnerable to client-side JavaScript issues

Another interesting piece of information was that out of the 98 vulnerable sites, 38 percent suffered from a vulnerability introduced by a third-party JavaScript code snippet (Figure 2). These snippets were included for adding one of the following capabilities:

- Marketing campaign JavaScript snippets.
- Flash embedding JavaScript snippets.
- Deep linking JavaScript libraries for Flash and AJAX applications.
- Social networking JavaScript snippets.

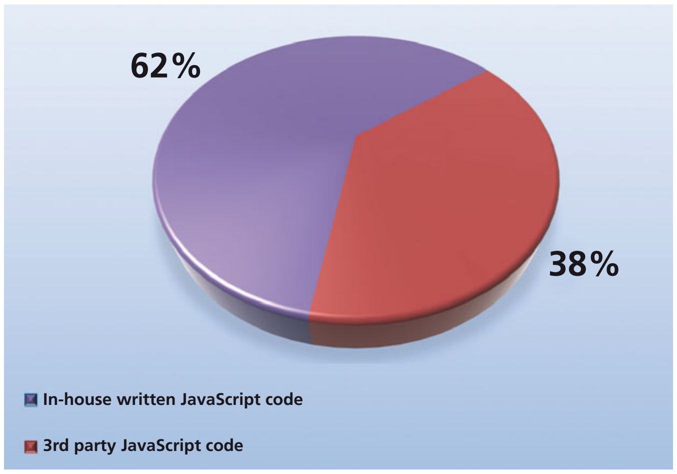


Figure 2: Vulnerable third party JavaScript code vs. in-house written code

Snippets of this kind are quite common in web applications these days, and we see a growing use of such third party JavaScript code, especially in Web 2.0 and AJAX web applications. Moreover, web developers often add these snippets blindly, without performing any security verification on them—they are unaware of the hazards they introduce to what could have been a reasonably secure application.

Looking at the distribution of vulnerability types (Figure 3), we see that of the 98 vulnerable sites, 92 (94 percent) suffered from DOM-based Cross-site scripting issues, and only 11 (11 percent) suffered from Open redirects. The total amount of DOM-based Cross-site scripting issues that were found was 2370, versus only 221 Open redirects.

In total, our scan included 169,443 web pages, out of which 90,929 were unique. Out of the unique pages, we have found that 1659 web pages had a verified client-side JavaScript vulnerability, which means that the approximate likelihood for a random web page on the internet to contain a client-side JavaScript vulnerability is one in 55.

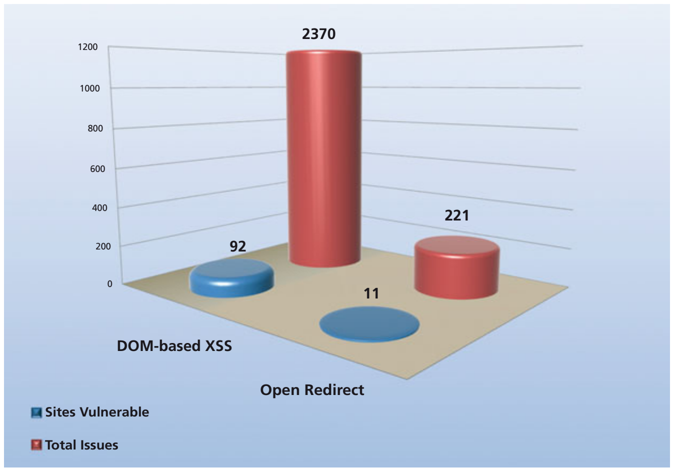
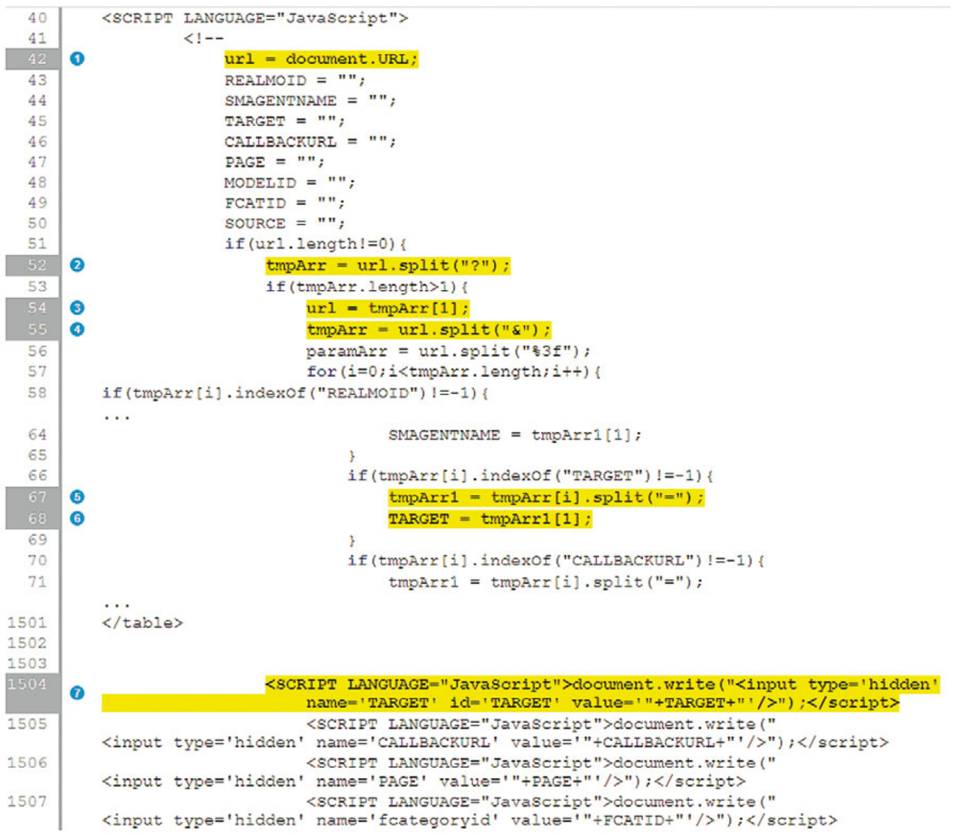


Figure 3: Distribution of vulnerability types (DOM-based Cross-site Scripting vs. Open redirects)

In order to validate each of the issues found, and to avoid false positive results, we used the trace information provided by JSA, and manually verified each vulnerability. Below (Figure 4) you can see an example of such JavaScript trace information of a real vulnerability found during our research:



```
40        <SCRIPT LANGUAGE="JavaScript">
41                <!--
42  ①                 url = document.URL;
43                    REALMOID = "";
44                    SMAGENTNAME = "";
45                    TARGET = "";
46                    CALLBACKURL = "";
47                    PAGE = "";
48                    MODELID = "";
49                    FCATID = "";
50                    SOURCE = "";
51                    if(url.length!=0){
52  ②                     tmpArr = url.split("?");
53                        if(tmpArr.length>1){
54  ③                         url = tmpArr[1];
55  ④                         tmpArr = url.split("&");
56                            paramArr = url.split("%3f");
57                            for(i=0;i<tmpArr.length;i++){
58        if(tmpArr[i].indexOf("REALMOID")!=-1){
          ...
64                                    SMAGENTNAME = tmpArr1[1];
65                                }
66                                if(tmpArr[i].indexOf("TARGET")!=-1){
67  ⑤                                 tmpArr1 = tmpArr[i].split("=");
68  ⑥                                 TARGET = tmpArr1[1];
69                                }
70                                if(tmpArr[i].indexOf("CALLBACKURL")!=-1){
71                                    tmpArr1 = tmpArr[i].split("=");
          ...
1501      </table>
1502
1503
1504                          <SCRIPT LANGUAGE="JavaScript">document.write("<input type='hidden'
    ⑦                             name='TARGET' id='TARGET' value='"+TARGET+"'/>");</script>
1505                              <SCRIPT LANGUAGE="JavaScript">document.write("
          <input type='hidden' name='CALLBACKURL' value='"+CALLBACKURL+"'/>");</script>
1506                              <SCRIPT LANGUAGE="JavaScript">document.write("
          <input type='hidden' name='PAGE' value='"+PAGE+"'/>");</script>
1507                              <SCRIPT LANGUAGE="JavaScript">document.write("
          <input type='hidden' name='fcategoryid' value='"+FCATID+"'/>");</script>
```

Figure 4: JavaScript taint analysis trace information

The example above is quite common. As can be seen, hacker-controlled data is first used in line #42, through the usage of the document.URL object, and is later used in the HTML code in line #1504. The various steps of the malicious data flow can also be observed.

Summary

Our research, which ran on a modest-sized sample group of 675 websites, showed that client-side JavaScript issues such as DOM-based Cross-site scripting and Open redirects are far more common than previously thought. Moreover, as Web 2.0 and AJAX design patterns that rely on untrusted third party JavaScript code gain popularity, it is likely that client-side security issues will become more and more common.

We suggest that the dearth of accurate statistics on the prevalence of such issues in public discussions, projects, and whitepapers on web application vulnerabilities is due to their complex nature and the difficulty involved in manually or automatically locating them. However, our research has demonstrated a new automated and accurate approach for locating client-side JavaScript issues, by amalgamating two separate security analysis approaches: static taint analysis of JavaScript code, and deep dynamic web crawling of running web applications. Our approach harnesses the best of both techniques to locate vulnerabilities in web applications with precision.

Vulnerability disclosure

IBM has notified the third party JavaScript vendors whose code was found by our research to contain vulnerabilities about the severe issues found by IBM and offered assistance in solving them.

About the authors

Ory Segal, security products architect and IBM Rational AppScan product manager
Ory Segal is a leading expert in web application security and an experienced product manager with more than 12 years of security and research experience. Ory is responsible for researching technologies and recommending strategic directions for IBM Rational's application security product line. Ory holds a degree in computer science from the Open University of Israel, and recently received an IBM Outstanding Technical Achievement Award. Ory is also an officer of the Web Application Security Consortium (WASC).

Omri Weisman, software development manager
Omri Weisman is a software development manager in IBM. For the past nine years, Omri has been leading software development projects in the field of application security and vulnerability assessment. In his current position Omri manages the Static Analysis Group in IBM Rational, responsible for building technologies for detecting security vulnerabilities through code scanning. Omri holds a B.Sc. in mathematics and computer science from the Ben Gurion University.

Adi Sharabani, cross-Rational security strategy and architecture
Adi Sharabani is in charge of the cross-Rational security strategy. As part of his role, Adi is responsible for leading, designing, and deploying overall security processes within the Rational development groups. Adi was formerly head the IBM Rational Application Security Research, responsible for research activities on web application security. Adi holds a B.A. in physics and in mathematics (both cum laude) from the Tel Aviv University and was a researcher at the University's Astrophysics Lab

Yair Amit, security and research group manager, Rational
Yair Amit is the manager of the Rational Application Security and Research group. Yair manages technological and security research and is responsible for the security content of IBM Rational's application security product line. Yair is recognized for his rich web and network security background; his research has found numerous security vulnerabilities and has been presented in various security events over the years. Yair holds a double major B.A. degree in computer science and life sciences with specialization in Bioinformatics (both summa-cum laude) from the Tel-Aviv University.

Lotem Guy
Lotem Guy is a senior security researcher at the Rational Application Security and Research group. Lotem is responsible for researching new web application vulnerabilities, performing application security audits and developing security related features for the Rational AppScan products family. Lotem holds a B.A. in computer science and computational biology from the Hebrew University in Jerusalem.

For more information

To learn more about IBM Rational AppScan products, contact your IBM representative or IBM Business Partner, or visit: **ibm.com**/software/rational/offerings/testing/webapplicationsecurity

Additionally, financing solutions from IBM Global Financing can enable effective cash management, protection from technology obsolescence, improved total cost of ownership and return on investment. Also, our Global Asset Recovery Services help address environmental concerns with new, more energy-efficient solutions. For more information on IBM Global Financing, visit: **ibm.com**/financing

[1] **Web Application Security Consortium**. Web Hacking Incidents Database (WHID). http://projects.webappsec.org/Web-Hacking-Incident-Database

[2] **Web Application Security Consortium**. WASC Statistics Project. http://projects.webappsec.org/Web-Application-Security-Statistics

[3] **Klein, Amit. 2005**. DOM Based Cross Site Scripting or XSS of the Third Kind. http://www.webappsec.org/projects/articles/071105.shtml

[4] **Klein, Amit. 2005**. DOM Based Cross Site Scripting or XSS of the Third Kind. http://www.webappsec.org/projects/articles/071105.shtml

[5] **Open Web Application Security Project**. OWASP Testing Guide: Testing for DOM-based Cross site scripting (OWASP-DV-003). http://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OWASP-DV-003)

[6] **Rational AppScan Standard Edition.** http://www-01.ibm.com/software/awdtools/appscan/standard/

[7] **MITRE. CWE-601**: URL Redirection to Untrusted Site ('Open Redirect'). http://cwe.mitre.org/data/definitions/601.html

Rational® software

RAW14252-USEN-00