The Challenges of Automated Application Assessments in a Web 2.0 World

Table of contents

HP Introduction

This white paper was written by two security experts from Stach & Liu, Rob Ragan and Vincent Liu, to describe the challenges of automated penetration testing or application scanning of Web 2.0 applications. In this white paper, they outline, the features that security experts should expect in a black box or dynamic application scanner.

Stach & Liu is one of the preferred partners of HP for delivering application security services. They are experienced users of HP Application Security software. Stach & Liu is a professional security consulting firm founded in 2005 to help companies secure their networks and applications. Services include network and application penetration testing, application source code review, network risk assessment, security policy and compliance review, and strategic security consulting.

The team of security professionals includes industry leading experts in network and application security who possess over 60 years of combined security experience and more than 50 years of security assessment and consulting experience with government intelligence, the Fortune 500, and Big 4 consulting.

In addition to authoring several best-selling security books, being published in industry journals, and profiled in multiple industry articles, the team has been presenting their security research for over a decade and has spoken at top conferences, including BlackHat, DefCon, InfoSecWorld, OWASP, SANS, and Microsoft BlueHat. Stach & Liu has strategic partnerships with Hewlett-Packard, Computer Sciences Corporation, and Checkmarx. Stach & Liu is headquartered in Phoenix, Arizona with locations in Los Angeles, Seattle, Atlanta, and Tokyo.

For more information, on Stach & Liu, visit http://www.stachliu.com/

About HP Application Security Center

HP Application Security Center is a suite of software, SaaS and professional services designed to help companies improve the security of their Web applications by helping to discover, fix, and prevent vulnerabilities that can be exploited by hackers. To learn more about HP application security offerings, visit www.hp.com/go/securitysoftware

The remainder of the paper is authored by Stach & Liu.

Introduction

Each new advance in Web application technology is invariably followed by a period of catch-up by black hats and white hats alike. White papers are published, conferences are held, and lessons are (hopefully) learned. Yet the race to identify and exploit application vulnerabilities is never ending. Keeping up with the latest terms, technologies, and techniques can be a full-time job in itself. It should come as no surprise that as technology changes, the tools we use to help us secure our applications must also evolve or risk irrelevancy.

For Web application vulnerability scanners, one of the most popular types of application security tools, one of the prevailing challenges has been keeping up with the migration of server-side application logic to client-side scripting—most often in the form of JavaScript and Flash objects. This offloading of functionality was initially triggered by technologies like JavaScript and XML (Ajax), which have evolved from their humble beginnings as an oft-mocked buzzword into foundational components of RIA "Rich Internet Applications" and the next-generation of Web 2.0 applications. Although the fundamental security issues surrounding Ajax applications are the same, these applications represent a paradigm shift in how Web applications are designed and developed, and have thusly impacted how we must approach Web application security assessments.

No longer is it sufficient to focus the majority of our efforts on assessing server-side functionality; now we are required to direct more attention and resources to identifying and analyzing vulnerabilities within client-side scripting code and embedded objects.

This movement of logic and functionality into the client has had major implications for automated Web application scanners. A Web scanner's ability to properly crawl the application has and always will be essential to testing it for security issues. Crawling is another way of saying automatically interacting with the application in the same way a user would. Before the RIA movement interaction was easy as it was sufficient to perform link discovery using simple screen scraping. Lexical parsing was used to identify and evaluate actions based on form tags and input values, but with a modern application these basic techniques will no longer suffice. A thorough crawler is necessary to build a proper attack surface and maximize testability. Automatic interaction is more difficult in a RIA environment because of more complex script actions such as timers expiring, mouse movement, dragging, and dropping. At the end of the day, a Web scanner is still testing for the same vulnerabilities, but it needs to explore all the client-side entry points to make sure it has covered the exposed features. Thus, link and feature discovery is one of the main challenges facing Web scanners today. Underlying the issue of link and features discovery are script event executions, DOM state analysis, and user emulation efficiency.

Script event execution

Today's users are accustomed to having their Web sites behave like their desktop applications—fast and interactive. The reason modern applications are able to provide a seamless application experience lies in the ubiquitous use of script events. These script events are used within a browser to trigger actions that update the page, make asynchronous calls, and generally make Web applications feel more interactive. Increased application responsiveness is also achieved through the use of client-side code by reducing the number of roundtrips to the server. A preliminary set of script code is loaded during the initialization of the page and this code is then executed within the browser. Subsequent code execution occurs asynchronously and behind the scenes to allow for fluidity in the user interface. At its core, Web 2.0 design strives for a smoother user experience and highly-usable and intuitive interfaces. Developers leverage scripts to achieve simple navigation and to bring rich content in their applications; avoiding the clunky experience of Web pages in the past.

Figure 1: Dynamic menu generated using JavaScript

```
1.  function createLinks(){
2.      for(var i=0;i<this.menuTexts.length;i++){
3.            var a=document.createElement('a');
4.            a.appendChild(document.createTextNode(this.menuTexts[i]));
5.            a.href=this.menuLinks[i];
6.            a.title=this.menuTexts[i];
7.            this.loc.indexOf(a.href,0)!=-1?a.className='activebutton':a.className='regbutton';
8.
9.            this.d.appendChild(a);
10.     }
11. }
12.
13. window.onload=function(){
14.     if(document.body&&document.body.firstChild){
15.           menuTexts=new Array('About Us','Products','Services','Links','Contact Us');
16.           menuLinks=new Array('about.htm','products.htm','services.htm','links.htm','contact.htm');
17.
18.           var menu=new Menu(menuTexts,menuLinks);
19.           document.body.insertBefore(menu,document.body.firstChild);
20.     }
21. }
```

An example script event is the onmouseover event that fires whenever a user moves the mouse pointer into an object. Figure 1 is an example from devarticles.com that describes how most dynamic menu bars are implemented in modern Web sites. It is common for a tag to have an onmouseover event call a function such as createLinks that writes multiple new tags containing hyperlinks to the document. When the window loads the JavaScript in Figure 1 it initializes arrays (lines 15 and 16) that contain a list of menu titles and corresponding HTML files. When the createLinks function is executed it creates an <a href> tag (line 3) for each value in the menu texts array and adds it to the DOM (line 9).

A versatile Web scanner needs a crawler intelligent enough to discover and audit such links to provide maximum application coverage. It is necessary to emulate the onmouseover event to discover the new links generated by the script event and subsequent function calls. Otherwise the new links generated by the script wouldn't be discovered and could not be audited. This is why emulating a user is so important to triggering functionality that may lead to other parts of the application.

Leading the industry in this regard, HP WebInspect relies on a custom script execution engine to emulate user behavior. Enabling the JavaScript and VBScript analyzer will allow WebInspect to discover any resources created or exposed through the execution of Ajax, and thus allowing for a more comprehensive analysis of the target application. Nevertheless, even with extensive and thorough mapping of an application's script code, the Web application scanner must also be precise with its analysis of the various states that the active content will produce, which introduces the issue of DOM state analysis.

Scripting Attack Vector

Interesting examples of a JavaScript events that executes without user interactions are onload and onerror. The onload event fires when a page or image is loaded. The onerror event triggers when an error occurs loading a document or image. Imagine a scenario where one of these events is calling a function that utilizes unvalidated data. If the data contains malicious input then it could lead to XSS without any user interaction.

DOM state analysis

In the same way script events can be used to create links, they can also be utilized to create and build page content. In a Web browser, rendered content that comes from sources such as HTML, XHTML, and DHTML files are represented in the DOM (Document Object Model) format. By representing the loaded page in this fashion, it becomes much easier to programmatically access the various elements under the DOM. In an HTML Web page, JavaScript inspects and manipulates the state of a page through the DOM. As the Web scanner will be emulating the JavaScript events, it becomes necessary to track the resultant effects and state changes to the DOM in order to provide full coverage. Pages might also contain external script references that retrieve content from other sites and dynamically render the content, which now must also be assessed. This can cause issues for a Web scanner because it needs to be able to download and parse script includes; potentially from multiple hosts. Again, this may not happen properly unless the scan has been configured to allow offsite script references to be downloaded and its execution then emulated by the Web scanner.

Dynamic drop-downs

Dynamic input generation is a form of script-based DOM manipulation often seen in applications that involve conditional searches. Shopping for cars online usually involves selecting a make, model, and color. Figure 2 demonstrates an example of DOM manipulation that dynamically populates drop-down inputs with values based on the previously selected value. Once the "make" of the car is selected the onchange JavaScript event fires to populate the Model drop down with corresponding values and enabling it for the user's next selection.

Figure 2: Cascading drop-down list

If the Web scanner doesn't trigger the onchange JavaScript event, the call to populate the next drop-down wouldn't be made. If the call to get the list of Models for the Audi manufacturer was a Web service then the Web scanner would never discover that feature and would be auditing a limited attack surface. Imagine if there were a critical vulnerability in that Web service such as XSS or SQL Injection and it was missed because the Web scanner wasn't able to properly emulate user behavior.

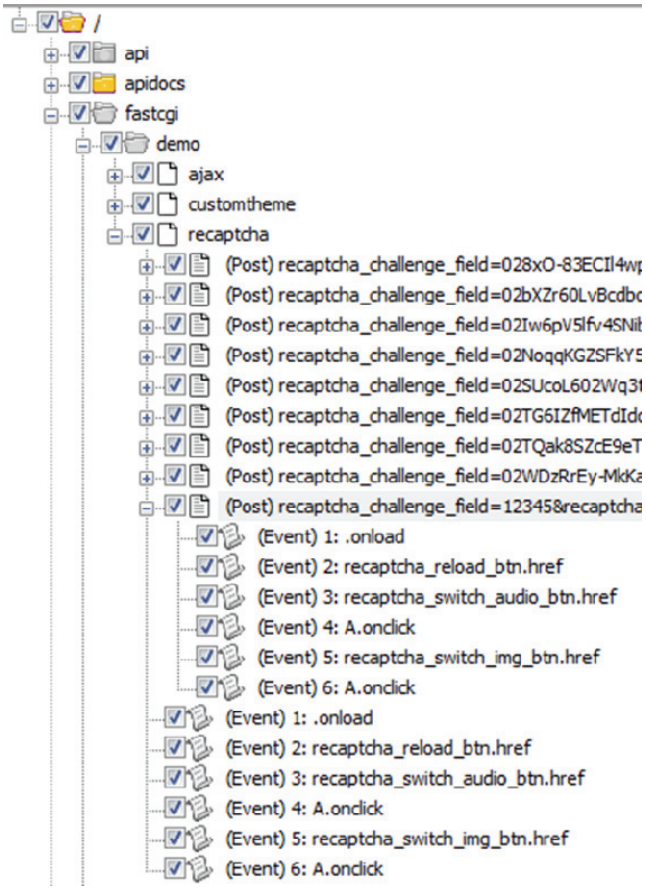
Web services

Implementing a third party Web service or utilizing an external API is characteristic of RIA and Web 2.0 applications. Dynamically generated content such as forms, images, and audio clips are a more complex example of script-based DOM manipulation. An example of a real-world scenario involving such content can be found in sites that implement the reCaptcha free CAPTCHA service. As depicted in Figure 3, a Web page that uses the reCaptcha service will include an iframe tag that loads a dynamically generated form and image to display to the user. There is also an audio clip dynamically loaded for visually impaired users.

Figure 3: reCaptcha—a human friendly dynamic form that is complex for a machine to process

Dynamically introducing content into a page creates considerably more potential application states. A Web scanner that fails to capture each unique state it discovers will not be able to accurately model the behavior of the application. HP WebInspect creates script event sessions for each change to the DOM. Tracking each of the different states and unique sessions enables better accuracy when testing for security vulnerabilities because the Web scanner can perform checks on each of the states. Figure 4 demonstrates a WebInspect crawl of reCaptcha. In this example there are six events that can be executed and a unique session is recorded for each of them. HP WebInspect is then able to audit the dynamically generated form and record the state of the application after emulating the execution of each of the form's script events. The site tree will be populated with newly discovered content and new entry points will be added to the audit queue ensuring better test accuracy.

Figure 4: HP WebInspect—JavaScript event sessions for reCaptcha



By tracking the DOM state changes for each of these events the Web scanner is better equipped to find things like new links, other dynamically generated content, and security vulnerabilities. However, with better coverage and accuracy there is a possibility of reducing performance.

User emulation efficiency

There exists an inherent risk of high resource utilization when tracking all of the script events and DOM state changes as well as the interactions between them. Controlling this issue, user emulation efficiency, remains an essential part of providing an acceptable performance baseline for any Web scanner. A performance baseline includes the reliability of the process, amount of system memory used, and the time to completion. The more client-side code present in the Web 2.0 application the more user emulation inefficiency becomes a problem. The more events there are to execute then the more DOM state changes there are to record. If the Web scanner's script emulation engine crashes because it encountered an out-of-memory error or some other unexpected condition it should not affect the rest of the application assessment. The emulation engine may reach some code that it doesn't interpret properly which may also cause it to halt or seemingly run forever. Especially, if the script contains a loop that waits for a condition that the emulation engine may not implement properly thus never achieve.

Figure 5: Dynamic calendar that generates new DOM elements

When the calendar thumbnail image is clicked, as shown in Figure 5, the onclick event fires rendering the current month under the text box. There are onclick events defined for each day along with the previous and next arrows in the top corners of the calendar. Making a selection on the arrow elements causes a new month to be dynamically added to the DOM. When a day is selected the text box is populated with the date value and the calendar is rendered invisible. This example demonstrates a single rich calendar feature that can possibly change the state of the DOM from hundreds of unique JavaScript events. This is exactly the type of complex feature that can lead to exhausting system resources or significantly delay a Web scanner assessment.

Web scanners need to be able to handle executing complex or poorly written code. In addition to being reliable and scalable, the script emulation should be configurable to complete processing in a practical amount of time. Often complex or poorly written code takes a very long time to process by an automated analysis tool. Web scanners are time saving tools used during a security assessment. The security professional using the tool needs the ability to tweak the amount of time spent on each application assessment. If a Web scanner does not have the ability to limit the number of script events it executes per page, it may appear to be scanning a site indefinitely. If the limit is set too high the analysis engine may exhaust system resources such as memory. HP WebInspect solves these issues by running the script emulation engine out-of-process from the other assessment tasks and by limiting the number of events executed per page to 1000 by default. WebInspect also provides a setting to adjust the number of events executed per page from 1 to 9999. This setting may require adjustment if scripts are endlessly executing the same events. Achieving the correct amount of user emulation efficiency may involve a trial and error approach.

Conclusion

We have seen how the features of RIA applications that provide such a rich user experience can limit the effectiveness of less-sophisticated security scanners. Reviewing the following examples described should make it apparent how script event execution, DOM state analysis, and user emulation efficiency are necessary problems that must be overcome by modern Web application scanners. If these problems are not addressed then there is possibility to miss links and features or not test the various states of the application. If the script analyzer uses too many system resources then there is a chance the Web scanners can go from being a time saver to being a time sink. HP WebInspect solves these problems by emulating script execution in an isolated process and recording the states the client-side code creates within the application. By solving these problems properly it enables maximum coverage, accuracy, and efficiency. Be sure to consider how these issues are addressed when choosing a Web application scanner.

About the authors

Rob Ragan is a Security Associate at Stach & Liu, a security consulting firm providing IT security services to the Fortune 500 and global financial institutions as well as U.S. and foreign governments.

Before joining Stach & Liu, Rob served as a software engineer at Hewlett-Packard Application Security Center, where he developed Web application security testing tools and conducted application penetration testing. Prior to the acquisition by HP, Rob was with SPI Dynamics where he led QA and testing for DevInspect, the flagship static source code analysis tool. Rob has also worked as a Web application developer for Pennsylvania State University. In these roles, Rob gained significant experience developing vulnerability identification techniques, improving automated assessment processes, and troubleshooting static source code analysis engines.

Rob actively conducts Web application security research and has presented the following: Filter Evasion: Houdini on the Wire, Static Analysis: The Art of Fighting Without Fighting, and Introduction to Web Application Security.

Rob holds a Bachelor of Science from Pennsylvania State University with a major in Information Sciences and Technology and a focus on System Development. While at Penn State, Rob was an active member of the Information Assurance Club where he provided training on Web application security.

Vincent Liu, CISSP, is the Managing Director at Stach & Liu, a security consulting firm providing IT security services to the Fortune 500 and global financial institutions as well as U.S. and foreign governments.

Before founding Stach & Liu, Vincent led the Attack and Penetration, and Reverse Engineering teams for the Global Security unit at Honeywell International. Prior to that, he was a consultant with the Ernst & Young Advanced Security Centers and an analyst at the National Security Agency. In these roles, he gained extensive experience conducting risk assessments, performing application code reviews, and supporting incident response situations.

Vincent is a contributor for the Metasploit Project and a respected member of the security community. He is an experienced speaker and has presented his research at conferences including BlackHat, ToorCon, and Microsoft BlueHat. Vincent has been published in interviews, journals, and books with highlights including: Penetration Tester's Open Source Toolkit; Writing Security Tools and Exploits; Sockets, Shellcode, Porting, and Coding; and Hacking Exposed: Wireless. Most recently, he led the technical editing of the critically-acclaimed Ajax Security, published by Addison-Wesley.

Vincent holds Bachelor of Science and Engineering degree from the University of Pennsylvania with major in Computer Science and Engineering and minor in Psychology.

4AA2-8419ENW, September 2009

STACH&LIU