

If You Can't Crawl It, You Can't Test It

Why A Little Understood, Challenging Technology May be the Key to Application Security

The collection of accurate data is central to the identification and remediation of web application vulnerabilities. Performing vulnerability assessments and penetration tests of Web applications is the traditional method of collecting the data needed by enterprise to ensure the application meets internal security and risk mitigation guidelines. But collecting accurate and usable data has been a much more difficult challenge than expected and continues to plague security teams responsible for Application Security.

This white paper covers in detail not only the technical challenges facing Application Vulnerability Scanners, but goes on to explain how these challenges have often adversely affected the scanners ability to properly perform accurate application assessments of targeted Applications.

Why it may seem obvious that crawling a website is a necessary precursor to testing it, many reviews of web application VA tools gloss over this crucial technology. To a great extent, the difficulty of building an accurate, comprehensive crawler is overlooked because the difficulty of the problem is not understood. This oversight is not only evident in published reviews of web application scanners but also in most bake-offs.

This issue is particularly troubling because imperfect crawlers render all other considerations about scanners irrelevant if not implemented properly. All the sophisticated attacks in the world are meaningless if the pages that need to be assessed have not been discovered. Pages can not be tested unless they have first been identified.

What is most concerning is that developers and security professionals may not even know if the scanner does not crawl all links on a site, unless they manually check the site against a list of crawled links. This may result in hundreds or thousands of false negatives.

This white paper will attempt to cover some of the major difficulties in application crawling and suggest strategies for testing automated crawlers.

Defining the Problem Space

The first true vulnerability assessment tools were network scanners. While one should not underestimate the technical complexities involved in creating network scanners, the actual definition of the problem space was fairly simple. Users simply input an IP and port range and the scanners created a list of targets in this a space of dimensions xy (IP addresses times ports). They started at the beginning and marched inexorably to the end of the queue. Additionally, there were a relatively finite number of outcomes; there are only so many platforms for a network scanner to assess.

The problem with web applications could not be more different. Users enter one page: the start page. There are hundreds or thousands of pages behind that that must be discovered before the scanner can even begin its work of testing for vulnerabilities. These pages are not laid out in a defined increasing sequence like an IP or port range. The scanner must extract links and follow them to find more links until it has discovered the entire website.

This problem would be easy enough if all links were obvious hrefs (e.g. a href='calendar.php?') in the HTML. A novice programmer can parse through HTML source and grab anything following an href= and add it to a crawl list. It is perhaps this misconception that this is all that is involved that has led so many analysts and security professionals to overlook this problem.

Alas, the web has evolved a wide range of technologies that have been grafted on to the original HTML backbone. While the major browser vendors have invested tens or hundreds of millions of dollars in man hours to make the user's interaction with these technologies seamless, replicating them in an automated crawler is nontrivial.

Lies, Damn Lies and Crawling JavaScript

Mark Twain once famously wrote that there are three kinds of lies: "lies, damn lies and statistics." If he were alive and writing on crawling technology today, he might amend that statement to "lies, damn lies and claims to crawling JavaScript."

At first glance, crawling JavaScript is a no-brainer. Many JavaScript links are merely hrefs embedded in JavaScript code. Scanners can use their existing grepping engines to search for these links and add them to the list of links to crawl and test for vulnerabilities. However, beyond simple grepping for plain links, crawling more complex forms of JavaScript can be quite difficult. Any crawler that searches for hrefs within JavaScript tags can certainly make the claim to crawl JavaScript. They can – just not all or even most JavaScript.

This is because most JavaScript links are not so easy to crawl.

Concatenated Links

Many websites actually use JavaScript to create the client-side Web pages that users see. For example, it is common to use JavaScript functions to actually create the links by concatenating the component parts of the link.

For example, the link
www.shoppingsite.com/shirts/bluepolo.html

could be expressed

```
function goto_product(department, name) {
```

```
'http://www.shopping.com/' + department + '/' + name + '.html';  
}
```

Used in this way

```
<a href="javascript: goto_product('mens','bluepoloshirt');">
```

Obviously, creating this link would require the scanner to obtain the JavaScript code and then execute it to create the link. Safeway's shopping website (shop.safeway.com) is a very good example of this. Almost all of the links are created from JavaScript concatenations.

Event-Driven Links

Additional complexity is created by websites that have event-driven links. For example, OnMouseOver, OnClick and OnChange attribute/events can result in links when a user does something. The links need to be crawled but they are not really there until the user executes an action. Comprehensive scanners have the ability to predict these user events and the links that they create in order to crawl and test them.

Combining JavaScript with Forms and Cookies

JavaScript is increasingly used with form submission, adding to the complexity of crawling. For example, JavaScript is frequently used to modify or supplement input data in forms. The scanners must know how to execute the JavaScript in combination with an automated input. The most notable example of this is logins that use forms. If the submission of the username and password is not properly handled, the scanner will not log in.

Microsoft's Passport (used in Hotmail and other popular Microsoft sites), for example, has a sophisticated implementation of authentication that uses JavaScript to submit the username and password.

JavaScript can also be used for other enabling technologies, like setting cookies or creating redirects. If these are not executed properly, the scanner's session can be lost, effectively terminating the scan before the application is completely scanned.

Authentication

Basic or http authentication and NTLM authentication are well-defined, established protocols. Unfortunately, most websites have gravitated towards form authentication, which is not based on a defined protocol and as such, has a near infinite number of possible implementations.

Form authentication presents several challenges to automated crawlers. First, the crawler must find and recognize the form used to authenticate. Because there is no specification, there are a large number of possible implementations. Additionally, the crawler cannot rely solely on variable names (e.g. username and password) because many developers use

their own shorthand for login variables. Crawlers need to be able to effectively reconstruct the tables that the users see and tie the descriptions to the variable names to be sure that they are identifying and populating login forms properly. Crawlers must also be able to identify and avoid logout links.

Through the Wormhole

The number of steps and tricks that web developers use to authenticate can be truly impressive. Users are sent back and forth across different pages, ports (http to https and back) and given multiple, changing cookies. In many cases, these schemes are designed to prevent bruteforce authentication programs from working against a site. They have the unfortunate added side effect of disabling most crawlers as well.

Any one of the technologies is fairly simple to handle in isolation. Crawlers must have highly sophisticated state models to be able to handle them all in combination. Any error in any part of the process will result in the authentication failing.

This issue is particularly important for single-signon schemes where a user will leave the domain being crawled and send to a central location to authenticate. A critical crawler function involves knowing what NOT to crawl – this is how crawlers avoid crawling the entire web; they are restricted to a defined domain or subdomain. In single-sign-on implementations, crawlers must know to go outside of the domain they are tasked with crawling but only to the domain where authentication takes place.

Session Management

Once users authenticate into a website, the server will keep track of them using a session cookie, obviously to avoid users having to re-authenticate on every page. Many change cookies periodically for security reasons. Automated crawlers must keep track of these cookies in a centralized module to make sure that they stay logged in. Given the large number of attacks being made in a typical scan (hundreds of thousands are not atypical) on a multi-threaded application, this can become quite difficult.

Infinite Links

Many applications have code that can create infinite or near-infinite links. The classic example is a calendar. Each day (or even each hour) can be a unique link. Web applications must have built in logic to identify these resources and make sure to limit them so that the application does not attack the same link endlessly.

Form Autopopulation: The Shopping Cart

As discussed in the section on authentication, identifying and populating forms can be quite challenging. This problem extends beyond authentication into standard crawling. The most interesting and challenging implementation is the shopping cart and user signup processes. Typically, these will be spread out over multiple pages (e.g. 1) basic information/username password, 2) shipping address and 3) billing address and credit

card). In order to get to the next page, all form fields must be identified, filled in properly and submitted. An additional source of confusion is that these pages will often have the same name (e.g. checkout.asp or signup.asp). Users will be directed to the second and third pages based on cookie information. Automated crawlers need to be able to identify that this is occurring and know to re-crawl the same page name in order to get to the next page.

Although this may seem a trivial matter (after all, a failure may result in a miss of 2-3 pages), it is not. The database information accessed by this part of the web application is typically the most sensitive and may include social security numbers and credit card data.

NTOSpider

The overriding design consideration for NTOSpider has been to automate the crawling and assessment process sufficiently so that users do not have to manually intervene to ensure that applications have been thoroughly and accurately tested. Many web application security professionals have hundreds or thousands of applications to audit on a monthly basis. Manual interaction with applications is simply not an option. Additionally, many security teams leverage less well trained personnel to run scans. Manually crawling an application requires skill and training to ensure complete results.

NTOSpider has the most advanced JavaScript parsing engine in the industry including support for concatenated links and event-driven links as well as JavaScript-driven form submissions. NTO has tested its crawler against some of the most JavaScript-intensive sites on the Web to ensure that it works in the real world.

NTO has also created a highly sophisticated state management model that handles any combination of redirects, port changes and cookie changes. Some of NTO's customers include companies that have created incredibly complex authentication and session management schemes to foil bruteforce programs. NTOSpider was the only scanner that could automate the crawling of their applications.

The Bake-off: Tips for Testing Scanners

As noted above, all vendors will claim that they handle these technologies...and they do to a greater or lesser extent.

The issue is that in the complex reality of the web, many websites contain a hodgepodge of solutions cobbled together. Automated crawling in the real world is quite difficult.

First, here is what you should not do.

- 1) Do not install a web hacking tutorial like Webgoat and see if your scanner can crawl it. Webgoat is a very powerful educational tool that contains a series of puzzles that a user must solve to crawl the site. Real world websites are not constructed this way.

- 2) Do not install a simplistic application and test it. When you buy the tool, you will be testing highly complex, large applications with multiple technologies. Many buyers have satisfactorily tested tools against simple applications only to find that once they purchased the tool and began real testing, the tool couldn't crawl their actual applications.

Fortunately, proper testing is fairly simple. While attacking websites is illegal (unless you are the owner), crawling is perfectly fine. Search engines like Google and Yahoo do it all day, every day. Select a group of websites with a range of technologies (like JavaScript, authentication, shopping carts and anything else you feel that you are likely to run into) and set the scanner to crawl only mode. Some simple hand checking of some deep links on the site should enable you to determine how well the crawler is doing. NTO can provide you with a list of interesting sites if you wish to expedite this process.

We would also highly recommend that you attack one or more of your own applications to verify that session management works during the attack phase (as well as to verify vulnerability discovery and false positive rates). To be safe, you may wish to copy them and install them on offline servers. Be sure that these servers have enough capacity to handle the load from the scanners. As mentioned above, scanners are all multithreaded; while production servers will be configured to handle the load, VMWare images of applications may be overloaded.

About the Author

As Director of Engineering at NT OBJECTives, Inc., Dan Kuykendall focuses on new threats and attack automation strategies for all aspects of Web application/services security.

Prior to joining NT OBJECTives, Mr. Kuykendall worked as a Web Application Software Engineer at [Foundstone](#) where he developed a web application for managing the FoundScan network scanning software. During this time Mr. Kuykendall was responsible for securing the web application from exploit from the outside and in its communication with the scanning engine.

Mr. Kuykendall previously worked at one of the top [20 European financial institutions](#) as a Network Engineer and part of its emerging Network Security Team. As part of his duties he was responsible for managing and securing a mixed network of Novell, Microsoft and Tandem servers and mainframes, conducting setup, maintenance and security audits.