by Dave van Stein

Web Vulnerability Scanners: Tools or Toys?

Executing a web application vulnerability scan can be a difficult and exhaustive process when performed manually. Automating this process is very welcome from a tester's point of view, hence the availability of many commercially and open-source tools nowadays.

Open-source tools are often specifically created to aid in manual testing and perform one task very well, or combine several tasks with a GUI and reporting functions (e.g. W3AF), whereas commercial web vulnerability scanners, like e.g. IBM Rational Appscan, HP Webinspect, Cenzic Hailstorm, and Acunetix Web Vulnerability Scanner, are all-in-one test automation tools designed for saving time and improving coverage.

Web Application Vulnerability scanners basically combine a spidering engine for mapping a web application, a communication protocol scanner, a scanner for user input fields, a database with attack vectors and a heuristic response scanner combined with a professional GUI and advanced reporting functions. Commercial vendors also provide frequent updates to the scanning engines and attack vector database.

Evaluating web vulnerability scanners

Over the past years many vulnerability scanner comparisons have been performed [1, 2, 3] and the most common conclusion is that the results are not consistent.

This great diversity in results can partially be explained by the lack in common testing criteria. Vulnerability scanners will typically be used by testers with various backgrounds like functional testers, network security testers, pen-testers, and developers, each of them having a different view on how to review these tools, causing different result interpretations. Sometimes this leads to comparing web vulnerability scanners with other security-related products, which is like comparing apples and oranges [1].

Another explanation is the diversity in a test basis. The vast amount of technologies and ways to implement these in web applications make it difficult to define a common test strategy. Each application requires a different approach, which is not always easy to achieve, making it hard to compare results.

Finally, like testers, vendors also have different views on how to achieve their goals. Although vulnerability scanners might look the same on the outside, the different underlying technologies can make interpretation and comparison of results more difficult than they appear to be.

The Web Application Security Consortium (WASC) started a project in 2007 to construct *"a set of guidelines to evaluate web application security scanners on their identification of web application vulnerabilities and its completeness."*[5]. Unfortunately this project has not reached its final stage yet, although a draft version has been recently presented[6].

This article focuses on the difficulties reviewing and using vulnerability scanners. It does not provide the best vulnerability scanner available, but discusses some of the strengths and weaknesses of these tools and gives insight in how to use them in a vulnerability analysis.

Using a web vulnerability scanner

Vulnerability scanners are like drills. Although the first are designed to find holes and the latter for creating them, their usage is similar.

Using drills out-of-the-box will possibly yield some results, but most likely not the desired ones. Without doing some research into the several configurations of the machine, the possible drillheads, and the material you are drilling into, you are more than likely to fail in drilling a good hole and may come across some surprises. Likewise, running vulnerability scanners out-of-the-box will probably show some results, but without reviewing the many configuration options and structure of the test object, the results will not be optimal. Also the 'optimal configuration' differs in each situation. Before using a vulnerability scanner efficiently, it is necessary to understand how scanners operate and what can be tested.

In essence, scanners work in the 3 following steps:

1. identify a possible vulnerability
2. try to exploit the vulnerability
3. search for evidence of a successful exploit

For each of these steps to be executed in an efficient way, the scanner needs to be configured for the specific situation. Failing in configuring one of these steps properly will cause the scanner to report incomplete and untrustworthy results, regardless of the success rate of the other two steps.

Knowing what to test

Before a vulnerability scanner can start looking for potential problems, it first has to know what to test. Mapping a website is essential to be able to efficiently scan for vulnerabilities. A scanner has to be able to log into the application, stay authenticated, discover technologies in use, and find all the pages, scripts, and other elements required for the functionality or security of the application.

Most vulnerability scanners provide several options for logging in and website spidering that work for standard web applications, but when a combination of (custom) technologies are used, additional parameterization is needed.

Another parameter is the ability to choose or modify the user agent the spider uses. When a web application provides different functionality for different browsers or contains a mobile version, the spider has to be able to detect this. A scanner should also be able to detect when a website requires a certain browser for the functionality to function properly.

After a successful login, a scanner has to be able to stay authenticated and be able to keep track of the state of a website. While this is no problem when standard mechanisms (e.g. cookies) are used, custom mechanisms in the URI can easily cause problems. Although most scanners are able to identify the (possible) existence of these problem-causing techniques, an automatic solution is rarely provided. When a tester does not know or understand the application, used techniques and possible existence of problems, the coverage of the test can be severely limited.

The Good

Vulnerability scanners are able to identify a wide range of vulnerabilities, each requiring a different approach. These vulnerabilities can roughly be divided into **four aspects:**

- information disclosure
- user input independent
- user input dependent
- logic errors

Information disclosure handles all errors that provide sensitive information about the system under test or the owner and user(s) of the application. Error pages that reveal too much information can lead to identification of used technologies and insecure configuration settings. Standard install pages can help an attacker successfully attacking a web application whereas information like e-mail addresses, user names, and phone numbers can help a social engineering attack. Some commercial vendors also check for entries in the Google Hacking Database[7]. Its use, however, is limited in the development or acceptance testing stage.
User-independent vulnerabilities cover insecure communications (e.g. sending passwords in clear text), storing passwords in an unencrypted or weakly encrypted cookie, predictable session identifiers, hidden fields, and having enabled debugging options in the web server.

Checking for both information disclosure problems and user-independent vulnerabilities manually can be very time-consuming and strenuous. Vulnerability scanners identify these types of errors efficiently almost by default.

The Bad

Bigger problems arise when testing for user-dependent vulnerabilities. These problems occur due to insecure processing of user input. The most known vulnerabilities of this kind are Cross-site scripting (XSS)[8, 9] , the closely related Cross-site request forgery (CSRF)[10, 11], and SQL injection[12, 13].

The challenge for automated scanning tools, when testing for these vulnerabilities, lie in detecting a potential vulnerability, exploiting the vulnerability and detecting the results of a successful exploit.

SQL injection

SQL injections are probably the best known vulnerabilities at the moment. This attack already caused many website defacements and hacked databases. Although the most simple attack vectors are no longer a problem for most web applications, the more sophisticated variants can still pose a threat. Even when an application does not reveal any error messages or feedback on the attack, it can still be vulnerable to so-called blind SQL injections. Although some blind injections can be detected by vulnerability scanners, they cannot be used for complete coverage, mainly due to performance reasons. Blind SQL injections typically take a long time to complete, especially when every field in an application is tested for these vulnerabilities. Most vendors acknowledge this limitation and provide a separate blind SQL injection tool to test a specific location in an application.

XSS and CSRF attacks

Cross-site scripting (and cross-site request forgery) attacks are probably the most underestimated vulnerabilities at this moment. The consequences of these errors might look relatively harmless or localized, but more sophisticated uses are discovered each day like hijacking VPN connections, firewall bypassing, and gaining complete control over a victim's machine. The main problem with XSS is that possible attack vectors run into millions (if not more). For example; an XSS thread on sla.cker.org[14] has been running since September 2007, contains close to 22,000 posts so far and new vectors are posted almost daily.

There are several causes that contribute to the vast amount of possible attack vectors:

- It is possible to exploit almost anything a browser can interpret, so not only via the traditional SCRIPT and HTML tags, but also CSS templates, iframes, embedded objects, etc.
- It is possible to use tags recursively (e.g. <SCR<SCRIPT>IPT>) for applications that are known to filter out statements.
- It is possible to use all sorts of encoding (e.g. unicode[15]) in attack vectors.
- It is possible to combine two or more of these vectors, creating a new vector that is possibly not properly handled by an application or filtering mechanism.

With AJAX the possibilities increase exponentially. Obviously it is impossible to test all these combinations in one lifetime, not in the least for the performance drop this would cause. Vulnerability scanners therefore provide a subset of the most common attack vectors, sometimes combined with fuzzing technologies. This list is, however, insufficient by nature, so additional attack vectors should be added or manually tested.

Another problem is the diversity of XSS attacks; the two most known types are reflective and stored attacks. With reflective attacks the result of the attacks is transferred immediately back to the client, making analysis relatively simple. Stored or persistent attacks on the other hand are stored at some place and not immediately visible. The result might even not be visible to the logged-on user and may require logging in as another user and understanding the application logic to detect them.

Stored or persistent user input vulnerabilities can basically be checked in two ways:

- Exploit all user input fields in a web application and scan the application completely again afterwards for indications of successful exploits
- Check what is stored on the server after filtering and sanitizing

Most scanners opt for an implementation of the first method, but detecting all successful exploits is a difficult task. Especially when an application has different user roles or an extensive data-flow, successful exploits can be hard to detect without understanding and taking into account the application logic.

Acunetix uses an implementation of the second method in a technology called AcuSensor[16]. Although this technology shows good results in detecting for example stored XSS and blind SQL injection attacks, the biggest drawback is that it has to be installed on the web-server in order to use it. This might not be problematic in a development or even acceptance environment, but in a production environment this is often not an option or even allowed.

The Ugly

The most difficult errors to find in a web-application are application and business logic errors. Although these errors are usually a combination of other vulnerabilities, they also contain a functional element contributing to the problem. Examples of logic errors are password resets without proper authentication or the possibility to order items in a webshop and bypassing the payment page. Since logic errors are a combination of security problems and flaws in the functional design of an application, practically all vulnerability scanners have problems detecting them. Commercial vendors like IBM and Cenzic do have a module for defining application logic attacks, but these are very basic modules and require extensive parameterization.

When testing for logic errors, manual testing still is necessary, although vulnerability scanners can be used for the repetitive or strenuous parts of the test. Practically all commercial vendors, but also e.g. Burp Suite Pro, have an option to use the vulnerability scanner as a browser. Hereby the tester chooses the route to test the application, while the scanner can perform automatic checks in the background.

Conclusions

Vulnerability scanners can be very useful tools in improving the security and quality of web applications. However, like any other testing tool, being aware of the limitations is essential for a proper use. With their efficient scanning of communication problems and bad practices, they can save time and improve the quality and security early in the development of web applications. When used for testing user input filtering and sanitizing, they can save time by rapidly injecting various attacks. However, manual reviewing of the results is essential and, due to the limited amount of attack vectors, additional manual testing remains necessary.

Fully automated testing of business and application logic is not possible with vulnerability scanners. Here vulnerability scanners have the same limitations as other test automation tools. However, when used by experienced security testers, they can save time and improve the test coverage when used to automate the most strenuous parts of the security testing process. □

1 http://www.networkcomputing.com/rollingreviews/Web-Applications-Scanners/
2 http://ha.ckers.org/blog/20071014/web-application-scanning-depth-statistics/
3 http://anantasec.blogspot.com/2009/01/web-vulnerability-scanners-comparison.html
4 http://en.hakin9.org/attachments/consumers_test.pdf
5 http://www.webappsec.org/projects/wassec/
6 http://sites.google.com/site/wassec/final-draft
7 http://johnny.ihackstuff.com/ghdb/
8 http://en.wikipedia.org/wiki/Cross-site_scripting
9 http://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
10 http://en.wikipedia.org/wiki/Csrf
11 http://www.owasp.org/index.php/Cross-Site_Request_Forgery
12 http://en.wikipedia.org/wiki/SQL_injection
13 http://www.owasp.org/index.php/SQL_injection
14 http://sla.ckers.org/forum/read.php?2,15812
15 http://en.wikipedia.org/wiki/Unicode_and_HTML
16 http://www.acunetix.com/websitesecurity/rightwvs.htm

> About the author

Dave van Stein

is a senior test consultant at ps_testware. He has close to 8 years of experience in software and acceptance testing and started specializing in Web Application Security at the beginning of 2008. Over the years, Dave has gained experience with many open-source and commercial testing tools and has found a special interest in the more technical testing areas and virtualization techniques. Dave is active in the Dutch OWASP chapter, and he is both ISEB/ISTQB-certified and EC-Council 'Certified Ethical Hacker'

Subscribe at:
SecurityActs
www.securityacts.com

Your Ad here
sales@securityacts.com