

Analyzing the Effectiveness and Coverage of Web Application Security Scanners

By Larry Suto
Application Security Consultant
San Francisco
October, 2007

Abstract

This paper summarizes my study of web application scanners and attempt to quantify their effectiveness. This study utilizes a novel methodology I've developed to objectively test the three leading web application vulnerability assessment tools. So far as the author knows, this is the first publicly published study that statistically evaluates application coverage and vulnerability findings by these tools.

The study centered around testing the effectiveness of the top three web application scanners in the following 4 areas.

1. Links crawled
2. Coverage of the applications tested using Fortify Tracer
3. Number of verified vulnerability findings
4. Number of false positives

One of the most surprising result is the discrepancy in coverage and vulnerability findings between the three tools. Lesser known NTOSpider excelled in every category, and the vulnerability findings show AppScan missed 88% and WebInspect missed 95% of the legitimate vulnerabilities found by NTOSpider.

Introduction

Testing the capabilities of web application scanners is an ongoing challenge that can be approached in a number of ways; the challenge is to create an objective test that is precise and can be replicated. While previous studies have tested web vulnerability assessment tools, none has statistically tested coverage or vulnerability findings in a precise manner. In this paper I take an approach that allows the data to be quantifiable to distinguish effectiveness (in terms of finding vulnerabilities) between the scanning tools.

To do this I employed Fortify's Tracer product which inserts its hooks into actual production J2EE applications to enable measuring the "application security coverage" and then running each of the three top commercial security scanners against the application. This allowed for actual analysis of how much of an application's code base was actually executed during a scan which enabled me to look at the scanners in a new and quantifiable way.

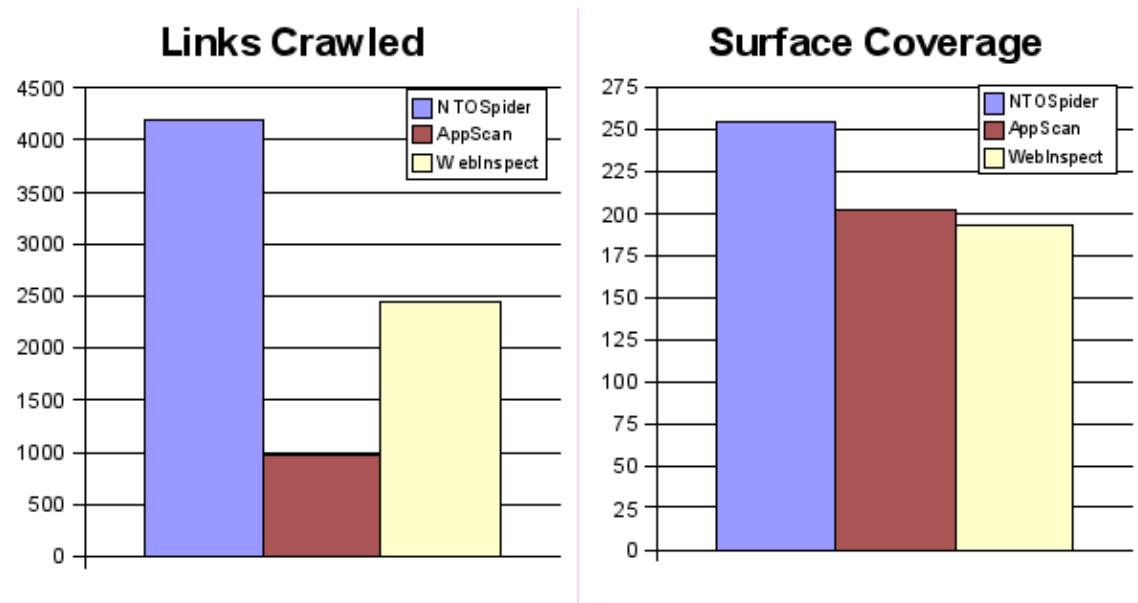
Summary

The full results of the testing are going to be analyzed in further detail later in the report, but I would like to start off with some of the conclusions first.

When looking at the results there are a number of ways to look at the data. There are difficulties around analysis of the overall percentages of the "possible" surface/sink coverage because it is hard to determine what subset of the full list is applicable to what a web scanner is expected to test

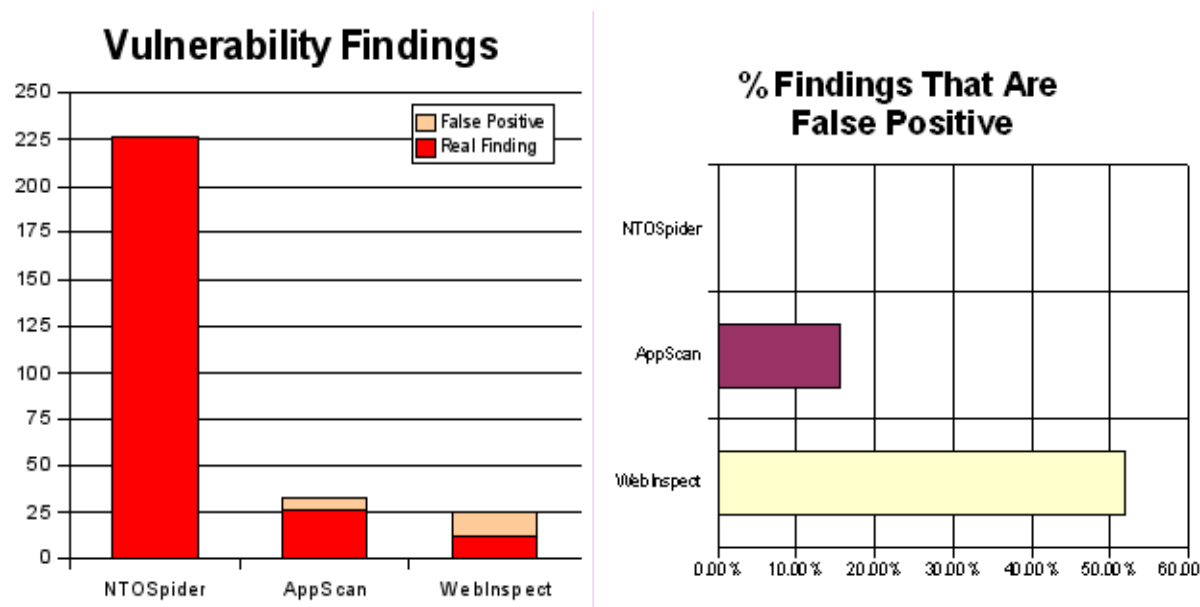
(excluding optional features not enabled, alternative database drivers, import/export data features, etc). For this reason, the numbers become unusable and I decided to remain focused on understanding the capabilities of the web application scanners against each other.

I started by comparing the results of the scanners against each other in the first two categories. Its interesting to see that the number of links crawled does not always indicate that more of the applications code base is being executed.



It is important that the scanner is able to crawl well, but wasting time on redundant inputs only adds to the crawled links, but does not increase coverage of the applications code base. In our tests, NT OBJECTives' NTOSpider product crawled the most links on average, and had the best coverage in all scans; SpiDynamic's WebInspect was able to crawl better than WatchFire's AppScan on one application, but AppScan did slightly better when I looked at the code base it executed. This means that WebInspect wasted time on redundant or otherwise unimportant links.

For this study I focused on verifying the findings found by the scanners for accuracy. Again the lesser known NTOSpider from NT OBJECTives had the most findings, lower false positives and most usable reports to aid in remediation.



Methods

Methodology

The basic idea of the Tracer tool is to place code into the application that will be able to analyze the web application scanner as it accesses security critical areas of the application. Code can be placed manually into the application source as it is built and turned on and off as testing takes place. This is more difficult and time consuming and requires access to the source code and thus the technique most often used is byte code injection which only requires access to the compiled byte code.

Software that uses bytecode in its operation (Java/.Net) can leverage techniques that come from the code coverage world such as instrumentation(byte code injection) in which code is inserted during build time or at run time using a custom class loader. Instrumentation adds the new code to the compiled code and thus the source is not necessary. Tracer employs byte code inserted statically into the compiled application byte code.

Testing

Each scanner was run in default mode and not tuned in any capacity to the application. The importance of this lies in how effective the default mode so that scalability of scanning is not limited by manual intervention and setup procedures which can be very time consuming. Second, in most cases it is simply unrealistic to spend much time with many applications.

Each scanner was provided a basic username and password similar to what a basic user would receive.

There were issues logging into some of the applications but calls to technical support quickly solved these problems.

The Tracer application was configured to monitor each web scanner as it executed its tests. After completing each test run, the results were saved and the monitoring tool was reset for the next scanner.

All vulnerability findings were hand vetted to determine if they were true findings or false positives.

Detailed Results

The following section reports the results of the tests. The analysis for this study was focused on 1) determining how well the scanners covered security sensitive sections of code within an application under test, 2) number of true vulnerability findings and 3) number of false positives. This study did not hand vet the applications to determine if there were any false negatives beyond those in the set discovered by any tool.

A set of security sensitive coverage categories were selected from the result set and the presented in the following tables:

Closed Source - Internal Corporate Application

Scanner	Links Crawled	Database API	Web API	Total API	Vuln Findings	False Positives
NTOSpider	91	20	35	55	0	0
AppScan	113	17	24	41	0	0
WebInspect	91	17	23	40	0	0

Roller - Open Source Blogging platform

Scanner	Links Crawled	Database API	Web API	Total API	Vuln Findings	False Positives
NTOSpider	736	2	121	123	2	0
AppScan	129	1	98	99	0	5
WebInspect	663	1	68	69	1	10

OpenCMS - Open Source Customer Management application

Scanner	Links Crawled	Database API	Web API	Total API	Vuln Findings	False Positives
NTOSpider	3,380	47	158	205	225	0
AppScan	742	36	132	168	27	0
WebInspect	1,687	45	140	185	11	3

Totals

Scanner	Links Crawled	Database API	Web API	Total API	Vuln Findings	False Positives
NTOSpider	4,207	69	314	383	227	0
AppScan	984	54	254	308	27	5
WebInspect	2,441	63	231	294	12	13

The three applications chosen have increasing level of complexity and size. The three scanners all did a reasonable job on the smaller, first application. The second application resulted in some false positives for Appscan and WebInspect.

The results for the last application are notable in that both AppScan and WebInspect severely underperformed NTOSpider in all three key areas of the test: 1) application coverage, 2) vulnerability findings and 3) avoidance of false positives. The fact that these results were most evident only in the most complex of these applications may indicate that for security groups to adequately test scanners, they need to use more complex applications. It is not surprising that smaller, less complex applications show less difference between the tools. One would expect fewer true findings and less complexity in crawling the applications.

In the aggregate, NTOSpider crawled 328% more links than AppScan and 72% more links than WebInspect; NTOSpider covered 24% more of the total APIs than AppScan and 30% more than WebInspect. NTOSpider found 227 total vulnerabilities versus 27 for AppScan and 12 for WebInspect. None of the findings by AppScan or WebInspect were missed by NTOSpider and

AppScan missed 88% and WebInspect missed 95% of the legitimate vulnerabilities found by NTOSpider. NTOSpider had a 0% false positive rate. Appscan had 5 false positives and a 16% false positive rate. WebInspect had 12 false positives and a 52% false positive rate.

The false positive findings were of interest because some appeared to be caused by custom 404 error handling routines in the web application, and some simply were based on faulty assumptions.

In addition the areas that coverage tool reported as missed were analyzed to determine if there were any security critical sections and also to try and to determine whether it would actually be possible for http based requests to access that portion of the application.

Conclusion

The most surprising result is the discrepancy in the number of vulnerability findings between the three tools. AppScan and WebInpsect are market share leaders in the space and their companies were both recently purchased by large, sophisticated technology companies (AppScan by IBM and WebInspect by HP). While security professionals testing small, highly secure, simple applications may achieve acceptable results from AppScan and WebInpsect, these results indicate that they may have some concern with relying on the results of these tools for larger applications. The relatively large number of false positives, particularly for WebInspect, is also a matter of some concern. False positives can be difficult for all but the most experienced security professional to identify. If they are not identified, they can cause difficulties by weakening the credibility of the security team with application developers. Additionally, vetting false positives by hand, even by experienced security professionals is a very time intensive process that will increase the cost of the program. While WebInspect has certain tools to reduce false positives, it would appear that this remedy is not necessary if using NTOSpider (and to a lesser extent AppScan). In any case, training the tool to reduce false positives will need to be done by experienced personnel and will increase program costs.

Biography

Larry Suto is an independent consultant which has consulted for companies such as Wells Fargo, Pepsico, Kaiser Permanente, Charles Schwab and Cisco during his time with Strategic Data Command Inc. based in Oakland, CA.

He specializes in enterprise security architecture, risk management, software quality analysis from a security perspective and RF security. Larry has been active in the industry for over ten years and has worked with many fortune 500 companies around the country. Recently his research has included introducing software testing and quality assurance techniques into the security engineering process in order to understand how the effectiveness of security tools and processes can be measured with more accurate and quantifiable metrics.

Larry can be reached at larry.suto@gmail.com