WEB APPLICATION SCANNERS

Evaluating Past the Base Case

GREG OSE
PATRICK TOOMEY

Presenter Intros

Overview

- An overview of web application scanners
- Why is it hard to evaluate scanner efficacy?
- Prior Work
- Our Analysis
- Recommendations
- Conclusion

Web App Scanners What, Who, Why

What

- What products are we considering web application scanners?
- What exactly do they do?

Who

- Who is using web application scanners?

Why

- Why are these products used
- What makes them attractive to their customers?

Scanner Evaluation Challenges

- Large number of products and vendors
- Large range in product price and feature set
- Testing requires a significant effort to reproduce real world environment and knowledge of how vulnerabilities manifest themselves in applications
- No means by which to easily measure one scanners vs another. Some test apps have been developed by vendors, but these are likely just geared as demos that are ensured to work with the product.

More Evaluation Challenges

- Unless the test app is custom tuned for evaluation purposes it is tough to say what is a false positive and/or false negative
- Many reviews are more feature specific than efficacy specific
- Those reviews that are efficacy specific often don't provide much sense of why a scanner found or didn't find the known vulnerabilities

Even More Evaluation Challenges

- These tools generate an inordinate amount of traffic, which makes it tough to evaluate
- To do a good job of evaluation would take a tremendous amount of time

Prior Work

- We mentioned it would take a ton of time to fairly assess the multitude of scanners that exist, sounds like the perfect way to use an NSF Grant!
- UCSB performed the largest evaluation of web applications scanners in July of 2010
  - Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA): “Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners”

Prior Work

- UCSB paper focused on entire lifecycle of web application scanner use (crawling, attacking, and analyzing the results)
- Analyzed a suite of vulnerability classes (largely focused around OWASP class of vulnerabilities)
- Analyzed 11 different commercial and open source scanners

Their Coverage

- XSS (reflected and stored)
- SQL Injection (first and second order)
- Code Injection (command injection and file inclusion)
- Access Control
- Session IDs
- Weak Passwords
- Directory Traversal
- Logic Flaws
- JavaScript Obstacles

Findings

- Half the vulnerabilities were not identified by any scanner
- Cost did not correlate with number of identified vulnerabilities
- Crawling is critical (particularly as the ubiquity of JavaScript marches forward)

Questions We Asked Ourselves

- Half the vulnerabilities were missed, mostly classes of vulnerabilities that present day scanners are wholly unsuited for. What if instead of focusing on what we know they are bad at, we restricted ourselves to what scanners are "good" at?
- For the things scanners should be good at, how far from the base-case do you need to get before the scanner is unable to reliably find a class of vulnerability?
- What is the take away from the above two questions for the average company that uses these scanners?

Typical List of Vulns

- Cross Site Scripting
- SQL Injection
- Insecure Cryptographic Storage
- Insufficient Transport Layer Protection
- Failure to Restrict URL Access
- CSRF
- Unvalidated Redirects
- Security Misconfiguration
- Broken Authentication/Session Management
- Insecure Direct Object Reference
- Information Leakage
- Logic Vulnerabilities
- Malicious File Execution

Difficulty of Automated Detection

Cross Site Scripting		Insecure Cryptographic Storage
Insufficient Transport Layer Protection	SQL Injection	Failure to Restrict URL Access (Forced Browsing)
Unvalidated Redirects	CSRF	
	Security Misconfiguration	
	Authentication/Session Management	
	Insecure Direct Object Reference	
		Logical Vulnerabilities
	Malicious File Execution	

Our Focus

- Stored XSS
- Blind SQL Injection
- Broken Authentication/Session Management
- Insecure Direct Object Reference (external control of file path)

Our Assumptions/Hypothesis

- Scanners will perform admirably under the base case for the set of vulnerabilities under evaluation
- The reason scanners typically do not identify vulnerabilities for which they should be well suited is due to “real world” variations

Our Analysis

- Create a set of trivial test cases for each of the vulnerabilities under evaluation
- Perform a scan for each test case independently, recording traffic for each scan
- Validate positive identification of the “base case” for each vulnerability class
- Modify each test case to incorporate “real-world” obstacles
- Repeat until scanning no longer was able to identify the vulnerability
- Analyze Results

Full Disclosure

- We restricted ourselves to the two commercial scanners that we had immediate access to (both scored in the top half of the UCSB evaluation)
- As a result, our coverage is not full, though we believe the insights gained are relevant regardless of the scanner under evaluation

Configuration

- During testing we received differing results across scanning runs
- We realized that some configuration options were set in a way that limited the number of tests performed per input
- Once configuration changed, all of the selected tests were actually run

Base Case Stored XSS

- Request:
  - http://site.com?var1=xssdata
- Response:
  - clickme
- Request:
  - http://site.com/viewpost?var1=16
- Response:
  - …xssdata…

Result

- Neither scanner Identified the vulnerability
- Why?
  - The dynamic ID returned from the POST was never used in subsequent fuzzing requests
- Takeaway:
  - The scanners would not identify stored XSS (or any test requiring state) unless, by chance, another page is already queued to be fuzzed that happens to use the XSS data stored to the database.
  - Content that is generated and requires specific identifiers to access will not be visited if it is not accessed during initial crawl.

Easier Base Case Stored XSS

- Request:
  - http://site.com?var1=xssdata
- Response:
  - click
- Request:
  - http://site.com/viewposts
- Response:
  - ….xssdata…

Result

- One scanner Identified the vulnerability (the less expensive one)
- Why?
  - The URL returned to view all the data stored in the database did not contain a dynamic parameter, and thus one of the scanners never visited the page after XSS data was injected.
- Takeaway:
  - Scanners try to optimize the number of pages visited to reduce scan times. This optimization can lead to reduced coverage and, as a side effect, an increased false negative rate.
  - Pages without dynamic parameters are not tested by the scanner, even if these pages contain dynamic content

We Can't Make This Easier Base Case Stored XSS

- Request:
  - http://site.com?var1=xssdata
- Response:
  - click
- Request:
  - http://site.com/viewposts?dummy=foo
- Response:
  - …xssdata…

Result

- Both scanners identified that XSS existed, but only one associated the identification of stored XSS to an input variable
- Why?
  - Your guess is as good as ours. The scanners used unique injections, which should allow them to trace the output back to an original input.
- Takeaway:
  - While both scanners eventually found the vulnerability, one of scanners required customizations to the configuration to increase some of the limits placed on testing requests.

Base Case Blind SQL Injection

- Request:
  - http://site.com?query=foo1
- Response:
  - Result Found
- Request:
  - http://site.com?query=foo1' and '1'='2
- Response:
  - Result Not Found
- Request:
  - http://site.com?query=foo1' and '1'='1
- Response:
  - Result Found

Result

- Neither scanner Identified the vulnerability
- Why?
  - At first we blamed one scanner's false negative on not using valid MySQL comment characters (so much for all those requests)
  - Further investigation showed that, both scanners sent a valid test of ' **and 'a'='a** in addition to ' **and 'a'='b**, but both failed to successfully use that knowledge to flag the finding
  - After debugging this issue, it appears that both scanners would not report blind SQL injection if the positive and negative response only differed by a small amount of data, a 3 byte difference in our case
- Takeaway:
  - To avoid flagging false positives, the scanners have a tolerance built into how they detect differences in response. This however may miss critical differences that can be used to identify blind SQL injection.

Verbose Base Case Blind SQL Injection

- Request:
  - http://site.com?query=foo1
  - http://site.com?query=foo1' and '1'='1
- Response:
  - AAAAAAAAAAAAAAAAAAAA…
Result Found
- Request:
  - http://site.com?query=foo1' and '1'='2
- Response:
  - Result Not Found

Result

- Both scanners now correctly identified the base case blind SQL injection.
- Takeaway
  - A difference of at least 10 bytes needs to be returned for both cases for the scanners to identify the vulnerability
  - In real world situation, it is not always the case that the output for a positive and negative response will dffer by this much

Dynamic Response Blind SQL Injection

- Request:
  - http://site.com?query=foo1
- Response:
  - Result Found…random_data…
- Request:
  - http://site.com?query=foo1' and '1'='2
- Response:
  - Result Not Found…random_data…
- Request:
  - http://site.com?query=foo1' and '1'='1
- Response:
  - Result Found…random_data…

Result

- One scanner Identified the vulnerability
- Why?
  - In an effort to reduce false positives/negatives scanners apply heuristics to allow for portions of the page to contain dynamic content while still finding/not finding the blind injection
- Takeaway:
  - Pages that contain dynamic content in responses may cause scanners to overlook test cases that depend on identifying differences in responses

Error Based SQL Injection

- POST:
  - http://site.com?query=foo1’
- Response:
  - ERROR ERROR ERROR ERROR ERROR
- POST:
  - http://site.com?query=foo1”
- Response:
  - OK

Result

- Both scanners identified the vulnerability
- Why?
  - The tested scanners test to see if an unclosed single tick result in an error
- Takeaway:
  - The tested scanners are good at correlating injections to error messages received by the scanner.
  - But what happens if the word ERROR doesn't appear in the response?

Modified Error Based Blind SQL Injection

- POST:
  - http://site.com?query=foo1'
- Response:
  - oh noes bad SQL!
- POST:
  - http://site.com?query=foo1''
- Response:
  - OK

Result

- One of the scanners failed to identify this SQL injection
- Why?
  - One of the scanners depends solely upon “error” messages to determine if invalid SQL syntax has was injected. For this scanner, a comparison of positive and negative responses is not performed.
- Takeaway:
  - The failing scanner relies on easily identifiable error message to identify this type of blind sql injection.

Timing-based SQL Injection

- POST:
  - http://site.com?query=foo1' and '1'='1
  - http://site.com?query=foo1' and '1'='2
- Response:
  - OK
- POST:
  - http://site.com?query=foo1"
- Response:
  - OK
- POST:
  - http://site.com?query=a' UNION SELECT SLEEP(5) --%20
- Response
  - 5 second delay then OK

Result

- Neither scanner identified the SQL injection
- Why?
  - The tested scanners do not send test cases for MySQL timing-based attacks. The test case for MSSQL was attempted
  - Even after configuring the scan to specifically target MySQL, no valid MySQL test cases were sent
- Takeaway:
  - The tested scanners advertise the ability to target specific environment, but never actually tune test cases for these environments

Base Case Session Handling

- POST:
  - http://site.com?user=user1&passwd=password
- Response(s):
  - set-cookie: session_id=1
  - set-cookie: session_id=2
  - set-cookie: session_id=3
  - set-cookie: session_id=n
  - set-cookie: session_id=n+1

Result

- Neither scanner identified the vulnerability
- Why?
  - Apparently no such analysis is being done on the entropy of the returned session identifiers. The UCSB team also noted the false negative in their report, though they had conjectured the failure was due to the scanners never attempting to log in as an administrator (where they placed their counter based session ID)
- Takeaway:
  - The application scanners under evaluation should not be used as a means of evaluating the quality of the session identifiers used in the application.

Easier Base Case Session Handling

- POST:
  - http://site.com?user=user1&passwd=password
- Response(s):
  - set-cookie: session_id=user1
  - set-cookie: session_id=user1
  - set-cookie: session_id=user1

Result

- Neither scanner identified the vulnerability
- Why?
  - Even though one of the scanner has a test case for “Use of Repeated Session Tokens”, no tests appear to have been run to test the trivial session tokens
- Takeaway:
  - The application scanners under evaluation should not be used as a means of evaluating the quality of the session identifiers used in the application.

Base Case External Control of File Name or Path

- Request:
  - http://site.com?fn=test_file.txt
- Response:
  - This is a test download file
- Request:
  - http://site.com?fn=test
- Response
  - Could not open /hackerhalted/download_files/test
- Request:
  - http://site.com?fn=../../../../../../../etc/passwd
- Response:
  - The contents of /etc/passwd

Result

- One scanner identified the vulnerability
- Why:
  - The unsuccessful scanner appended the directory traversal test cases to the original input. The directory traversal attempt was never attempted in place of the original input. This test case would only be valid in Windows, where non-existent directories can be traversed in to and out of.
- Takeaway:
  - While these test cases should be easily tested, flaws in implementation could lead to false negatives.

NULL-byte Truncated External Control of File Name or Path

- Request:
  - http://site.com?fn=test_file
- Response:
  - This is a test download file
- Request:
  - http://site.com?fn=test
- Response
  - Could not open /hackerhalted/download_files/test.txt
- Request:
  - http://site.com?fn=../../../../../../../../etc/passwd%00
- Response:
  - The contents of /etc/passwd

Result

- The previously successful scanner identified the vulnerability
- Why:
  - NULL-byte termination is a common to eliminate any appended file extensions in exploiting vulnerabilities.
- Takeaway:
  - Again, base-case external control of filename vulnerabilities are identified within automated testing.

“Filtered” External Control of File Name or Path

- Request:
  - http://site.com?fn=test_file.txt
- Response:
  - This is a test download file
- Request:
  - http://site.com?fn=../../../../../../../../etc/passwd
- Response:
  - Could not open /hackerhalted/download_files/etc/passwd
- Request:
  - http://site.com?fn=....//....//....//....//....//....//....//....//etc/passwd
- Response:
  - Content of /etc/passwd

Result

- Neither scanner identified the vulnerability
- Why:
  - No test cases were performed to attempt to identify common mistakes made when attempting to filter for directory traversal characters.
- Takeaway:
  - The evaluated application scanners do not test for variations of the base case test files to reflect common improper filtering attempts. This is something that manual testing, in combination with the revealed file path, would be easily enumerable by the tester.

Multiple Input External Control of File Name or Path

- Request:
  - http://site.com?dn=test_dir&fn=test_file.txt
- Response:
  - This is a test download file
- Request:
  - http://site.com?dn=../../test1&fn=../../test2
- Response:
  - Could not open /hackerhalted/download_files/../../test1/test2
- Request:
  - http://site.com?dn=../../../../../../../etc&fn=passwd
- Response:
  - Content of /etc/passwd

Result

- Neither scanner identified the vulnerability
- Why:
  - The scanners do not successfully test multiple parameters in unison to identify vulnerabilities that may result in the fuzzing of more than one input.
- Takeaway:
  - Vulnerabilities that require multiple parameters to be utilized together will not be identified.
  - To perform testing of this nature would exponentially increase the number of requests required to complete the tests.

Conclusions

- Configuration matters
- State changes between crawl and test impact results
- Pages with non-dynamic parameters likely won't be tested
- A number of environmental specific test cases may not be tried
- We found tests largely ignore common filtering evasion techniques
- We found tests do not try to fuzz multiple parameters concurrently