Syllogistic Web Application Testing

INTRODUCTION TO WEB APPLICATION TESTING - LESSONS LEARNED - MOVING FORWARD

At this talk we will provide insights we've learned from performing web application testing, writing web application testing tools, and using the OSSTMM methodology for for web application testing.

Presented By:

Outline

Speakers

Speakers:

- Robert E. Lee
  - CEO, Dyad Labs, Inc.
  - Director of Projects and Resources, ISECOM
  - OPST & OPSA Certified Trainer
- Jack C. Louis
  - Chief Security Researcher, Dyad Labs, Inc.
  - Systems Programmer
  - OPST & OPSA Certified Trainer

Special Thanks

Special Thanks to Haroon Meer of Sense Post

Syllogistic Web Application Testing

Syllogism

- To conclude by reasoning
- The conclusion necessarily follows from the premises; so that, if these are true, the conclusion must be true, and the argument amounts to demonstration

Syllogistic Web Application Testing

Requires tester to understand what they are trying to accomplish

- Follow a Methodology
- Create a Hypothesis
- Create Test Scenarios
  - Introduce Stimulus
  - Measure Response
  - Compare Results to Hypothesis

Man vs Machine

- People Can Think
  - Analysis
  - Decision Making
- Computers Can Process
  - Tedious
  - Repetitive
  - Simple If/Else - Nothing more

Rules of Engagement

- Customers pay for testing
  - Work with them, not against
- Doctors office analogy
  - Don’t play games if you are the customer/patient

ISECOM Quick Facts

- Institute for Security and Open Methodologies
  - Originally called the Ideahamster Organization (Est. 2001)
  - Non Profit Organization Registered in Spain and U.S.A.
  - Open Source Community Registered OSI
- Open Source Security Testing Methodology Manual
  - Released in Jan. 2001
  - Over 1.7 million downloads

Background to the OSSTMM

Open Source Security Testing Methodology Manual

- It is a guide for a thorough security test from the outside to the inside.
- It includes security testing parameters for 6 viewpoints known as sections:
  - Physical Security
  - Internet Technology Security
  - Information Security
  - Wireless Security
  - Communications Security
  - Process Security

Re-Engineering

- Manually walk through program logic
  - Contrast authenticated vs unauthenticated access
  - Contrast access from various privilege levels if applicable
- Determine the protocol specification of the server/client application
- Decompose or deconstruct the application component (flash, vb-script, Java, etc) , if applicable

Authentication

- Find authentication points in the application
- Attempt to find valid login credentials with password/username brute forcing, if possible
- Attempt to bypass authentication system with replay of valid authentication tokens
- Determine the application logic to maintain the authentication sessions
  - number of (consecutive) failure login attempts allowed (in what period of time)
  - login time-out, etc.

Session Management

- Determine the session management information
  - number of concurrent sessions allowed for one user account
  - IP-based session state
  - identity-based session state
  - cookie usage
  - session ID in URL encoding string
  - session ID in hidden HTML field variables, etc.
- Reverse engineer the session ID sequence and format

Session Management - Cont

- Measure the session tokens randomness
  - Entropy
  - Compression Reduction
  - Chi square distribution
  - Arithmetic mean value of data bytes
  - Monte Carlo value for Pi
  - Serial correlation coefficient

Input Manipulation

- Use a wide variety of fuzzing strings (special characters, varied lengths, varied encodings etc) for every input found
  - Client Header Fuzzing
  - Get Request Fuzzing
  - Post Fuzzing, Hidden Field Fuzzing
  - Cookie Fuzzing
  - Reverse IP DNS Fuzzing
- Look for SQL Injection opportunities
- Look for "Cross-Site Scripting" opportunities
- Examine unauthorized directory/file access with path/directory traversal in the input strings of the applications

Output Examination

- Valuable information stored in the cookies
- Valuable information from the client application cache
- Valuable information stored in the serialized objects
- Valuable information stored in the temporary files and objects
- Valuable information in log files - look for "bad things"
- Talk to the web application coding team
- When in doubt, go to the source (if possible)

Things I'm not going to talk about

This stuff has already been talked about too much:

- .bak files
- default scripts
- default manual pages
- file/directory fuzzing/finding
- WS_FTP.LOG files

Error Detection

- It's hard due to many layers
  - Web Server
  - Subcomponents of Web Service (PHP/asp, etc)
  - Application Tier
  - Database Tier
  - Network
  - OS Level
  - Etc

Error Detection

- Hard to automate detection of error conditions
  - Reset packet or an internal server error
  - An SQL error message
  - Some injected content containing HTML that the client renders
  - Partial page content
  - etc, etc, etc.

Error Detection

- To make things worse generally you'll need to find out the ways that the specific platform will do in error conditions
- Perhaps the user configured errors to be 'sorry this page is not available now', instead of the application blowing its guts all over the screen

Error Detection

- Some solutions:
  - Have access to the log files of the application servers or other related log files for review after the fuzzing/testing
  - Ask the admin to turn on verbose error reporting while the test is going on
  - Work with the development team to save time, because a lot of times they will know the specific impact of a particular problem way before you will
  - Remember time is the enemy, and testers have less time and fewer resources than attackers do

Error Detection

- Any attempt of developers/admins to obfuscate errors during your test will help you to miss or not understand critical security issues
- This will leave your admin feeling clever while being insecure in the end

Examples of different Errors

Examples of different sorts of errors that will happen at different layers:

- Back end Errors
  - stat() with a huge file, an overflow inside of an SQL query, SQL injection, etc
- Application Errors
  - direct code injection, authentication bypass, improperly sanitized user input, etc

Examples of different Errors - Cont

Examples of different sorts of errors that will happen at different layers:

- Protocol Errors
  - showing an ISE [500], HTTP post read timeout with back trace (too long content-length parameter) aka the REL Fat finger Special
  - Note: A lot of things simply look for this but it doesn't work a lot of the time
- Web Server Errors cause overflow within code of web server itself causing the error to show up as a TCP reset or something.

Spike Selection

- Choose a base set of spikes relevant to the specific architecture and then find out the standard decoding methods (base64, URL encoding, etc) and reenter all the spikes with that encoding
- Remember, network programs are liberal in what they accept and conservative in what they send, so take advantage of this automatic translation feature to allow you to inject things that are otherwise not allowed

Now lets fuzz

- Generic fuzzing is easy at this point, you have HTTP headers, GET variables and POST variables (and perhaps reverse DNS names that contain spikes too, for example server environment variables)
  - Set your container for the application, Restricting crawling to where it lives.
  - Find places where you need to login, basically different entry points to the application, repeat fuzz for each entry point
  - Crawl web application
  - Determine all points to fuzz and systematically go though application fuzzing each variable with each spike in turn.

Now lets fuzz - Cont

Don't just blindly crawl it:

- Make sure you don't keep crawling pages with silly parameters changing
  - Example: a calender section where the days and years change, but the parameter to fuzz on the back end is all the same
- Just crawl those parameters once, and get some good values for fuzzing later on (like use 2005 for the year and thats it)

Some fuzzing tips

- Tool shouldn't automatically flag things, its better to just record everything and review afterward based upon information gathered about the platforms error messages
- Logins cause many issues (or sessions) because certain fuzzing will trigger conditions where re-login is required, many times this may be application specific, but there are some generic methods of doing this.

Some fuzzing tips - Cont

- Javascript and flash cause problems, but its not un-solvable, if a web browser can do it, so can an application fuzzer
  - Some people think this isn't true, but someone wrote Netscape, and someone wrote a flash plugin
  - It's actually less work to re-implement these things without graphics in mind.
- Fuzzer should use plugins just like browsers do, Javascript isn't *that* bad because we are (generally) not required to render visual things, so we can place blank stubs for Javascript that resizes windows and changes the title bar, etc.

Some non-obvious things we learned

Common mis-conceptions:

- If a web development platform doesn't have issues with buffer overflows, then to really be safe, you cannot pass data back to anything that does have issues (like the kernel or libc or a back end database written in C++ or C, etc)
- Good way to break 'safe' web applications is to use very large variable lengths and wait for the kernel to error, you'll at least get information leakage bugs for later use.

Cruiser

Cruiser Overview:

- uses libcurl + libxml2 + file(1) + (libexif + swfrw + libjs from mozilla)
- that means:
  - the HTML parser is forgiving and flexible (libxml2 rocks)
  - we have mime types from the server, but we peek at the data too and record file info we can extract like file(1)
  - libcurl knows about a lot of protocols
- optionally
  - you can use exif to extract image meta information
  - (not in the upcoming release but the one after it) it has a REAL Javascript parser
  - it can do basic SWF grokking to at least extract links from flash (when support is finished it will walk flash like HTML)

Cruiser - Cont

- Written in C, sports a command line interface that doesn't require a Display
- Has a gdb like interface for interactive testing (with breakpoints and backtraces and tab completion!)
- Uses a PHP+PostgreSQL web front end to quickly search collected data (yeah grep works too, but its not relational)
- Its not complete but usable, it will be a while before it is *done* because of the complexity of the job at hand.

Where do I get it?

- Cruiser will be part of the OSACE suite on Source Forge soon
- Alternatively, it will be available at: http://www.osace.org/cruiser
- Cruiser specific code will be released under the GPL license

Hacking Exposed - Live

Lolololololololz!1oneon1o!none

What did I just see?

- Real Security Testing is tedious and meticulous manual work
  - It is not sufficient to simply "ask questions" or "run tools"
- Automation is OK so long as you don’t turn your brain off in the process
  - Don’t trade your brain for a pretty tool
  - Artificial Intelligence is Artificial
  - Seek out tools that make you more efficient that still allow you to think
- Cruiser might be a fun new addition to your Web Application testing tool suite.

The End

Thank you for your time.

- For more information, see:
  - http://www.sensepost.com
  - http://www.isecom.org
  - http://www.osstmm.org
  - http://www.osace.org
  - http://www.dyadlabs.com
- Or just write us:
  - Jack at dyadsecurity dot com
  - Robert at dyadsecurity dot com