Introduction to Cyber Security Week 5: Vulnerabilities, Static and Dynamic Analysis

Ming Chow (mchow@cs.tufts.edu)

Twitter: @0xmchow

Learning Objectives

- By the end of this week, you will be able to:
  - Be exposed to and understand the difference between CVE and CWE
  - Use static analysis software to identify vulnerabilities
  - Understand the difference between static and dynamic analysis
  - Write a technical risk analysis

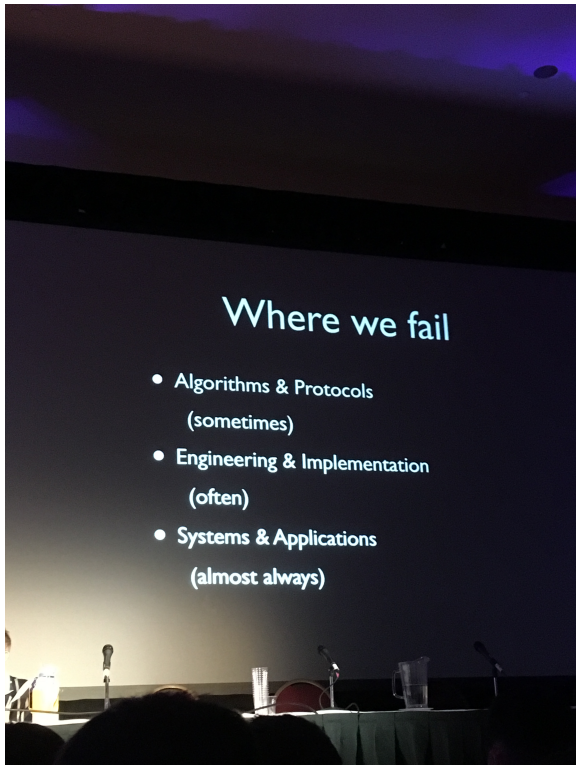
Why Talk About Vulnerabilities, Risk Analysis, Static and Dynamic Analysis Now?

- Terminology and vocabulary
- The Capture The Flag game is now over; time to look at what happened, what was wrong with the application, how and what to fix
- The issue of vulnerability disclosure is an ongoing debate, one of the really hard problems in Cyber Security
- Understand why software development is very difficult; all software have bugs but some bugs are worse than others
- The last two topics (Cryptography and Web Security) have a lot to do with vulnerabilities

Motivation: How Hard is Software Development?

- Here's a problem for you: *on paper*, write a binary search program.
  - Recall binary search: find the position of a target value within a sorted list by "comparing the target value to the middle element of the array; if they are unequal, the half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful or the remaining half is empty." (Wikipedia)
- A humbling exercise, pitfalls galore including off-by-one errors: https://stackoverflow.com/questions/504335/what-are-the-pitfalls-in-implementing-binary-search
- Thank Jon Bentley via Matt Blaze for this https://twitter.com/mattblaze/status/771086675258802176

From Matt Blaze and Sandy Clark's talk
"Crypto War II: Updates from the Trenches"
at The Eleventh HOPE Conference in NYC,
July 2016



Vocabulary: Bug vs Flaw

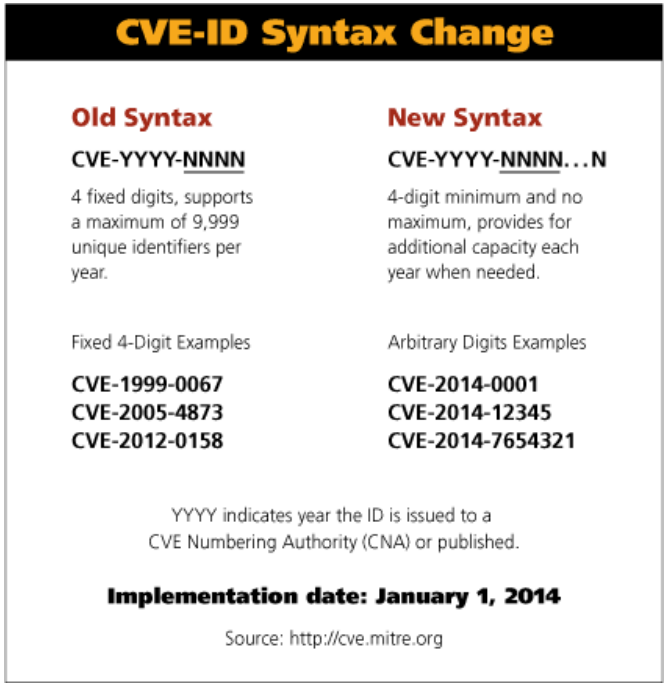
- **Bug** - An error that exists in the implementation-level (i.e. only exist in source code); very correctable
- **Flaw** - An error at a much deeper level, particularly in the design, and likely in the code level; can be very difficult and costly to correct

Vocabulary: Vulnerability

- “A weakness in the computational logic (e.g., code) found in software and some hardware components (e.g., firmware) that, when exploited, results in a negative impact to confidentiality, integrity, OR availability.”
- Source: https://cve.mitre.org/about/terminology.html
- Furthermore: *“Examples of vulnerabilities include:*
  - *phf (remote command execution as user "nobody")*
  - *rpc.ttdbserverd (remote command execution as root)*
  - *world-writeable password file (modification of system-critical data)*
  - *default password (remote command execution or other access)*
  - *denial of service problems that allow an attacker to cause a Blue Screen of Death*
  - *smurf (denial of service by flooding a network)”*

What is CVE?

- Common Vulnerabilities and Exposures (CVE)
- Created in 1999 by MITRE, Steve Christey Coley (@SushiDude) and David Mann
- https://cve.mitre.org/
- A *dictionary (not a database)* of common names (i.e., CVE Identifiers) for publicly known cybersecurity vulnerabilities
- Free for public download and use
- CVE ID Syntax: *CVE prefix + Year + Arbitrary Digits*
- Does NOT provide proof of concept (PoC) or exploit!



Then There is CWE. What is CWE?

- Common Weakness Enumeration
- Also maintained by MITRE
- https://cwe.mitre.org/
- *"A formal list of software weakness types created to:*
  - *Serve as a common language for describing software security weaknesses in architecture, design, or code.*
  - *Serve as a standard measuring stick for software security tools targeting these weaknesses.*
  - *Provide a common baseline standard for weakness identification, mitigation, and prevention efforts."*
- Source: https://cwe.mitre.org/about/index.html

What is CWE? Continued

- **Some Common Types of Software Weaknesses:**
  - Buffer Overflows, Format Strings, Etc.
  - Structure and Validity Problems
  - Common Special Element Manipulations
  - Channel and Path Errors
  - Handler Errors
  - User Interface Errors
  - Pathname Traversal and Equivalence Errors
  - Authentication Errors
  - Resource Management Errors
  - Insufficient Verification of Data
  - Code Evaluation and Injection
  - Randomness and Predictability

What's the Difference Between CVE and CWE?

- Arguably the best explanation via Daniel Miessler:
  - "CWE: has to do with the vulnerability—not the instance within a product or system"
  - "CVE: has to do with the specific instance within a product or system—not the underlying flaw."
  - Source: https://danielmiessler.com/blog/difference-cve-cwe/
- Example:
  - CVE-2015-2213 is a SQL injection vulnerability in WordPress
  - CWE-89: Improper Sanitization of Special Elements used in an SQL Command (is the weakness (or flaw) in the code of WordPress that caused CVE-2015-2213.)
  - Source: https://www.veracode.com/blog/2016/08/language-appsec

National Vulnerability Database

- https://nvd.nist.gov/home.cfm
- Maintained by NIST: National Institute of Standards and Technology
- Uses CVE
- Database; contains references to advisories, solutions, and tools
- Example, regarding CVE-2015-2213 (from previous slide): https://nvd.nist.gov/vuln/detail/CVE-2015-2213

Open Sourced Vulnerability Database (OSVDB)

- http://osvdb.org/
- People: attrition.org, H.D. Moore, Rain Forest Puppy, Chris Sullo
- DEAD, looking for someone to pick it back up
- Open source
- “Provided accurate, detailed, current, and unbiased technical information on security vulnerabilities. The project promoted greater and more open collaboration between companies and individuals.” (Wikipedia)

The Exploit Database

- https://www.exploit-db.com/
- Maintained by Offensive Security
- A CVE compliant archive of exploits and vulnerable software
- "A repository for *exploits and proof-of-concepts rather than advisories*"
  - Source: https://www.exploit-db.com/about/
- Downloadable
  - Tool: `searchsploit` - command line search tool for Exploit-DB

Scanning for Vulnerabilities

- Tools:
  - Nikto
  - Nessus
  - OpenVAS
  - Metasploit
  - w3af
  - Many others

Tool: Nikto

- Written by Chris Sullo
- Open Source
- Web server scanner “designed to find various default and insecure files, configurations and programs on any type of web server”
- Documentation: https://cirt.net/nikto2-docs/
- Source code: https://github.com/sullo/nikto
- Example: `nikto --host <IP ADDRESS>`

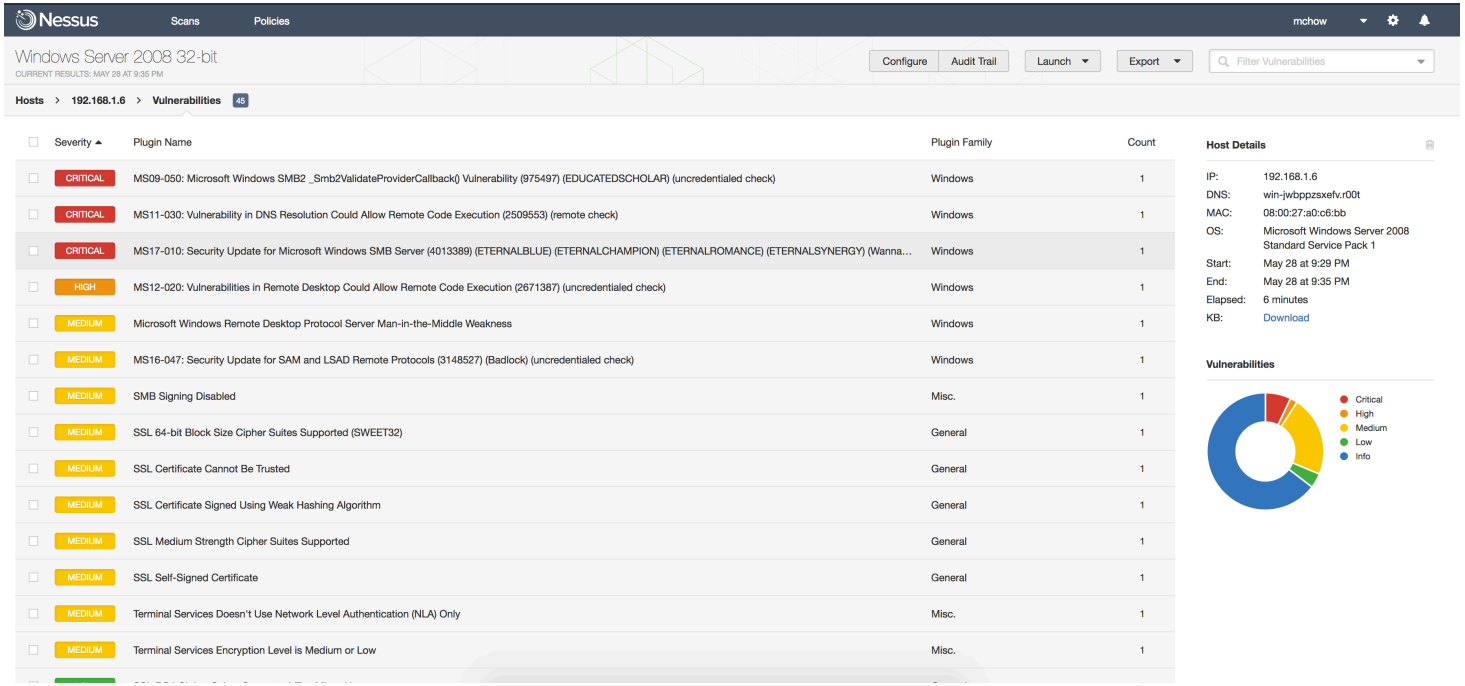
Tool: Nikto (example continued)

```
$ nikto --host=192.168.1.66 --root=/mutillidae
- Nikto v2.1.5
---------------------------------------------------------------------------
+ Target IP:          192.168.1.66
+ Target Hostname:    192.168.1.66
+ Target Port:        80
+ Target Path:        /mutillidae
+ Start Time:         2017-06-05 18:13:27 (GMT-4)
---------------------------------------------------------------------------
+ Server: nginx/1.6.2
+ Cookie PHPSESSID created without the httponly flag
+ Cookie showhints created without the httponly flag
+ The anti-clickjacking X-Frame-Options header is not present.
+ Uncommon header 'logged-in-user' found, with contents:
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Server leaks inodes via ETags, header found with file /mutillidae/robots.txt, fields: 0x586fe41c 0
xbe
+ "robots.txt" contains 8 entries which should be manually viewed.
+ /mutillidae/index.php?page=../../../../../../../../../../etc/passwd: The PHP-Nuke Rocket add-in is
 vulnerable to file traversal, allowing an attacker to view any file on the host. (probably Rocket,
but could be any index.php)
+ OSVDB-3233: /mutillidae/phpinfo.php: Contains PHP configuration information
+ OSVDB-12184: /mutillidae/index.php?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potential
ly sensitive information via certain HTTP requests that contain specific QUERY strings.
+ OSVDB-3093: /mutillidae/.htaccess: Contains authorization information
+ OSVDB-5292: /mutillidae/?_CONFIG[files][functions_page]=http://cirt.net/rfiinc.txt?: RFI from RSna
ke's list (http://ha.ckers.org/weird/rfi-locations.dat) or from http://osvdb.org/
+ OSVDB-5292: /mutillidae/?npage=-1&content_dir=http://cirt.net/rfiinc.txt?%00&cmd=ls: RFI from RSna
ke's list (http://ha.ckers.org/weird/rfi-locations.dat) or from http://osvdb.org/
+ OSVDB-5292: /mutillidae/?npage=1&content_dir=http://cirt.net/rfiinc.txt?%00&cmd=ls: RFI from RSnak
```

Tool: Nessus

- Commercial
- Was open source at one point; closed source in 2005
- “The world’s most widely deployed vulnerability scanner”
- “There are 86731 plugins, covering 38201 unique CVE IDs and 25042 unique Bugtraq IDs.” https://www.tenable.com/plugins/index.php?view=all
- https://www.tenable.com/products/nessus-vulnerability-scanner
- Nessus Home (free): https://www.tenable.com/products/nessus-home
- OpenVAS (Open Vulnerability Assessment System) is a free and open source fork of Nessus

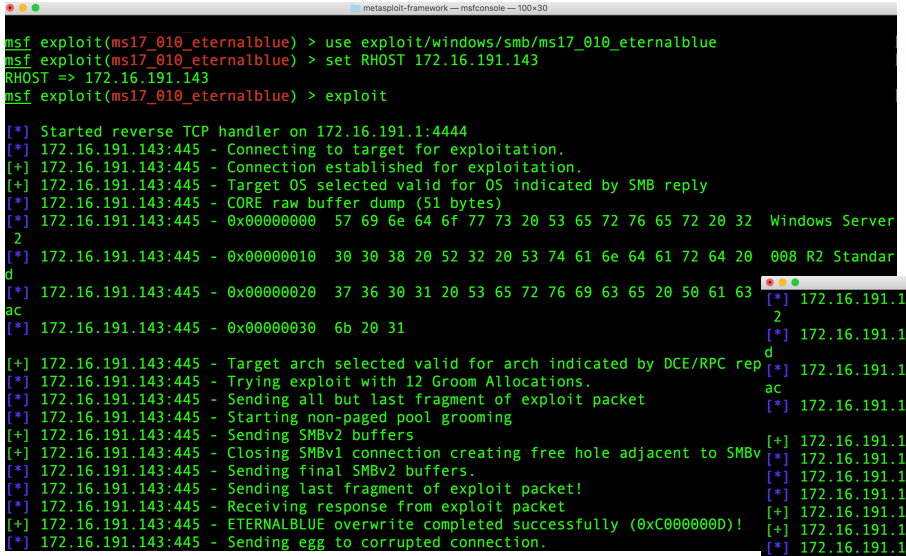
Tool: Nessus (continued)



Tool: Metasploit

- https://www.metasploit.com/
- Source code: https://github.com/rapid7/metasploit-framework
- Written by H.D. Moore
- Acquired by Rapid7 in 2009
- Open-source platform for developing, testing, and using exploit code
- Currently has over 1600 exploits, 400 payloads

Tool: Metasploit (continued)

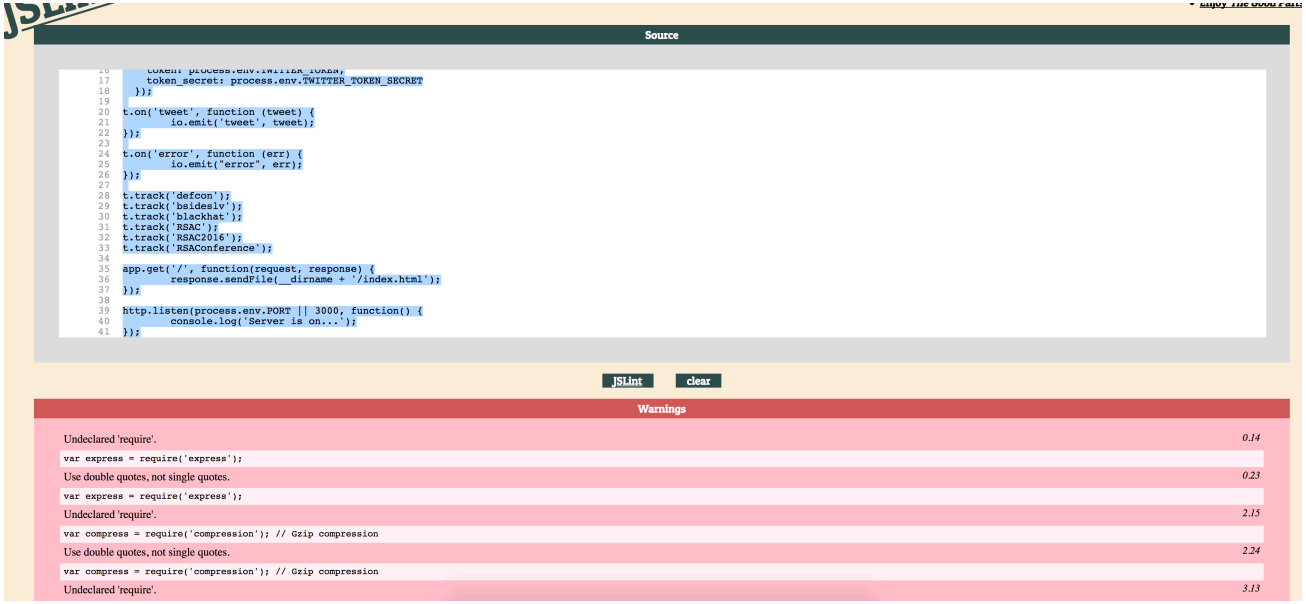


Static Analysis

- Also known as static code analysis
- No execution of program
- Rule based
- *Full code coverage*
- Will catch bugs in source code such as using insecure or unsafe functions
- Binary static analysis: black box, no code
- Code: white box , given source code
- Examples: grep, lint, Coverity (commercial), Fortify (commercial), Veracode (commercial)
- Reference: https://www.veracode.com/products/static-analysis-sast/static-code-analysis

Tool: JSLint (Lint for JavaScript)

- http://www.jslint.com/



A Glance at Static Analysis Techniques

1. Data flow analysis
   - Collect runtime info about data while in a static state
   - Basic block (the code), control flow, control path
2. Control graph
   - Node => block
   - Edges => jumps / paths
3. Taint Analysis (also Deterministic Finite Automaton)
   - Identify variables that have been tainted
   - Used vulnerable functions known as sink
4. Lexical analysis
   - code => tokens (e.g., /* gets */)

Strengths and Weaknesses of Static Analysis

- Strengths:
  - Find vulnerabilities with high confidence
- Weaknesses:
  - Many false positives or false negatives can be generated
  - Can't find configuration issues
  - Can you prove findings are actual vulnerabilities?

Dynamic Analysis

- System execution; run-time
- Trial and error
- Detect dependencies
- Deal with real runtime variables
- Based on automated tests, user interactions
- No guarantee of full coverage of source
- Example: valgrind – for memory debugging, memory leak detection, and profiling. http://valgrind.org/

To Ponder

- Question: If you do a scan or a penetration test of a system and no vulnerabilities are reported, is that a good thing?
- Source of picture: Gary McGraw