



Automating security checks

Security and Privacy
VSE Prag, 7 - 11.6.2010

Institute for Information Processing and
Microprocessor Technology (FIM)
Johannes Kepler University Linz, Austria

E-Mail: sonntag@fim.uni-linz.ac.at
<http://www.fim.uni-linz.ac.at/staff/sonntag.htm>



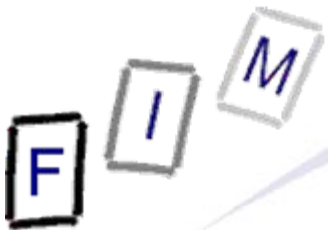
Why automatization?

- Ensuring security is not that hard for a single system
 - You know it in detail
 - When something is discovered, it is implemented and tested
- But: Many sites with many configuration options?
 - Do you know them all?
 - » Are they identical everywhere (versions!)?
 - Do you have time to change everything accordingly?
 - » Or do you depend on automatic updates/roll-out?
 - Are you sure you did not miss one option somewhere?
 - » Testing the same thing several times is tedious
- Solution: Automatic testing whether a problem exists
 - Professions write tests → You just apply them
 - » No need to know exactly how the attack works!
 - Regular re-testing is possible
 - Ad-hoc & patchy testing → Systematic & comprehensive



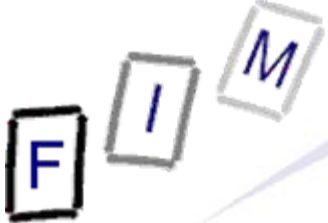
Overlap with monitoring

- Some overlap with system monitoring exists
 - Failures are just a “different kind” of attack
 - Some problems may occur accidentally or intentionally
 - » Example: Blacklisting of mail servers
 - Monitoring may uncover exploitation of a problem
 - » Will not find how the attacker hacked the system, but that, e.g. through increased load, increased outgoing traffic, ...
- But there are some important differences:
 - Monitoring knows in advance what to look for, security requires frequent updates for newly discovered problems
 - Monitoring takes place more frequently
- Similar software/integration possible, but not the same!



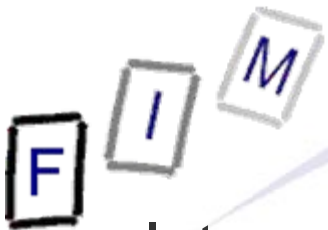
What can be automated?

- Code tests: Analysis of source code
 - For known errors or potentially dangerous patterns
 - Or just trying: E.g. fuzzing (random input)
- Web application tests
 - Very important, because they are a regular source of problems and can be exploited from everyone at a distance
 - » Elevation of privilege → Only your employees!
 - Examples: DNS hijacking, blacklisting, defacement, malware injection, suspicious account activity, specific exploits
- Properties of tests:
 - Probabilistic: Some tests give no definite answer; e.g. exploits that only work rarely (depending on memory layout, ...)
 - Destructive: Some tests will crash the software/system
 - Method vs. exploit: Checking for general method of attack (e.g. SQL injection) or testing a specific problem (typ. bug)?



Source code analysis

- Often external programs run on the source
 - Better: Integration in development environment
 - » Run continually, i.e. after every change/before compilation
- Checking for code problems
 - Can do a lot of analysis impossible later (compilation!)
 - Quality varies: Always a problem ↔ Rarely wrong
 - » Still: **Every** single issue must be investigated in detail!
- Typically static analysis, but need not be
- Examples:
 - Using unsafe methods (“sprintf” instead of “snprintf”)
 - Access to shared variable without locking from threads
 - Accessing non-reserved memory; not freed memory
 - Uninitialized variables, data tracing, duplicated code, ...



Development environments: Eclipse & Java

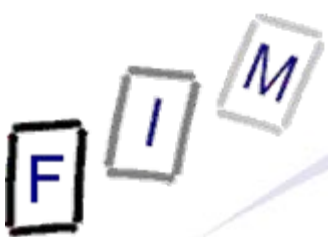
- Integrated under Java Compiler Errors/Warnings
 - Long list including other aspects
 - » E.g. code style → understanding problems
- Checked whenever a Java file is saved
- Examples:
 - Assignment problems: `x=x; if (x=y);`
 - Switch case fallthrough: `case ?: x; case ?: ...`
 - Null pointer access
 - Dead code: `if (false) ...`
 - Redundant/unnecessary code: unused variables
 - Hidden fields/variables
 - Overriding/no overriding methods
- Most are not directly security relevant, but hint at bugs
 - And bugs sometimes lead to security problems
- Similarly: Validation of HTML/XML/JSP/... files



- DNS Hijacking: Modification of DNS server/responses
 - Redirecting requests to other IP addresses
 - Requires checking various DNS servers all over the world
 - » Not a guarantee, however!
- Domain Hijacking (theft): Transfer of the domain name to a different owner; typ. also to a different server
 - Verification of the registrar information/WhoIS
- Defacement: Modification of the website by a third party
 - Typically the result of a hack
 - Difficult to distinguish automatically from authorized modifications and for dynamic pages (e.g. blogs)
- Certificates: HTTPS cert. valid, identical, not insecure
 - E.g. replaced certificate (→ hack)



- Possible for both websites and E-Mail
 - May be based on domain name or IP address
- E-Mail: Spam, phishing
 - Sources: Spam Haus, SURBL
- Web: Spam, phishing, virus, exploits, popups, ...
 - E.g. Norton safe Web, Google Safe browsing, Site Advisor
 - Marked as inappropriate for children (→ minor protection!)
- Possible reasons:
 - Someone hacked your site/placed malware on it
 - Someone sent spam with you as sender/over your mailserver
 - Incorrect message sent to owner of list
- Can be difficult to get off the list!

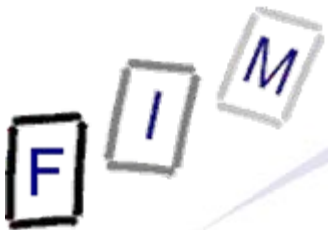


- Adding JavaScript to the webpage or code to the source
 - Intention: Infecting the computer of the browser
 - Will typically **not** be a (technical!) problem for your server
 - » **But will probably be a legal problem!**
- Requires a bug or lacking security on your site
- Example: Hidden iframe (size: 1x1 pixel, hidden)
 - Often created through (nested) obfuscated scripts
 - Then used for drive-by downloads
- Can be very difficult to detect, as the code can be obfuscated, randomly modified etc.
 - Typical solution: Compare with known-good page/source
 - Alternative: Check for suspicious activity/links/frames
 - Alternative: Use real browser and monitor actions

Suspicious account activity

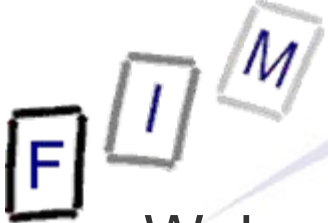


- Checks whether an account has been hijacked
 - So typically user-oriented, but also for servers
 - » Systematic problem allowing hijacking, not trojan on client
 - » Typical problem: Cross Site Scripting (XSS)
 - Steal session ID → change password → own account
- Other elements may be checked as well: Used for sending Spam, phishing, illegal activity, credit card fraud etc.
 - This is typically very specific for the individual site and therefore not available in general!
- Typical signs for account hijacking:
 - Log ins from different IPs/IPs in different countries
 - Log-ins to multiple accounts from the same IP
- Cannot be distinguished from outside; requires software within or on the server
- Basic vulnerabilities can be discovered in other ways



General: Specific exploits

- This covers all kinds of vulnerabilities
 - Web server, operating system, installed software, etc.
- Can be run from inside or outside; where attackers might be
 - Reason: Inside protection is often much more lenient and when someone managed to get in, there should still be no obvious security problems
- Signatures are implemented as small scripts
 - Each new attack/weakness/bug → New script
 - » Requires continuous updating!
- Note: Will be used by attackers as well!
- Example: Nessus (see practical day)
- More exploit oriented: Metasploit
 - Regularly used by attackers
 - Main element is exploitation, less finding a security problem

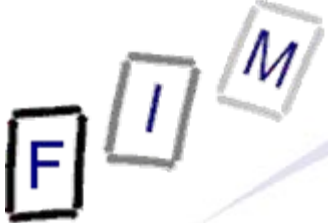


Example: Skipfish

- Web application security scanner
 - Will scan a whole site for various security problems
- Very simple usage
- Scans for various risk levels:
 - High: SQL injection, command injection, file upload, ...
 - » Brute force: Huge logs, enormous time!
 - Medium: Directory traversal, stored/reflected XSS, script/css injection, mixed content, MIME- and charset mismatches, incorrect caching directives, etc.
 - Low: Directory listing, stored/reflected redirection, content embedding, mixed content, credentials in URLs, SSL certificates, forms without XSRF protection, ...
- Allows partial checking (checks take quite long)
 - X % of all links followed/problems checked
 - » Randomly determined → Regular scanning → Probably checked everything over some time!



- Note: Skipfish has only a very limited database of known vulnerabilities
 - Based on three-step differential probes
 - » Uses wordlists to look for extensions and for filling in forms
- Skipfish is provided as source code
 - For a Linux-like environment (Mac, Cygwin, ...)
 - Just run “make” to compile it
- Select a dictionary to use
 - Note: The bigger the dictionary, the longer the scan takes!
- Start it on command line with output directory and URL
 - Additional parameters allow restricting the depth, percentage of links followed, specify authentication cookies (to get around logins), connection rate limiting, ...
- Example: `./skipfish -o output_dir http://www.example.com/`



Skipfish: Output interpretation

- Output is produced as an annotated sitemap
 - First line can expand
- Below: Problems found in decreasing importance with brief explanation
 - Note: Many things not necessarily a problem!
 - » E.g. PUT: If file upload is intended, this is OK (here it is not 😊!)
- Note: Took 88 hours, but is not even remotely complete!

Skipfish - scan results browser

Scanner version: 1.33b Scan date: Mon May 3 07:47:41 2010
Random seed: 0x83340d23 Total time: 88 hr 45 min 51 sec 492 ms
[Problems with this scan? Click here for advice.](#)

skipfish

Crawl results - click to expand:

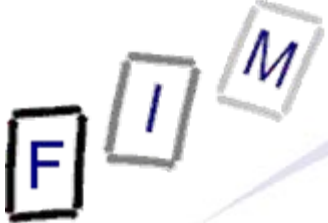
+ <http://www.██████████/> 6 45 40 78 122 441
Code: 200, length: 16108, declared: text/html, detected: application/xhtml+xml, charset: iso-8859-1 [[show trace +](#)]

Document type overview - click to expand:

- application/javascript (7)
- application/xhtml+xml (18)
- image/gif (10)
- image/jpeg (1)
- image/png (29)
- text/css (6)
- text/html (9)
- text/plain (6)
- text/xml (3)

Issue type overview - click to expand:

- PUT request accepted** (5)
 1. <http://www.██████████/clickheat/index.php/PUT-sfi9876> [[show trace +](#)]
 2. <http://www.██████████/clickheat/index.php/images/PUT-sfi9876> [[show trace +](#)]
 3. <http://www.██████████/clickheat/index.php/images/flags/PUT-sfi9876> [[show trace +](#)]
 4. <http://www.██████████/clickheat/index.php/sfi9876/PUT-sfi9876> [[show trace +](#)]
 5. <http://www.██████████/clickheat/index.php/styles/PUT-sfi9876> [[show trace +](#)]
- SQL query or similar syntax in parameters** (1)
- Interesting file** (1)
- Incorrect or missing charset (higher risk)** (31)



- Reliability of automated security checks is very mixed
 - Specific exploit code tested → Perfect (attack did work)
 - General programming style → Might sometimes be a problem
- Typical scans always produce a large number of warnings
 - Your SSL certificate is not an officially recognized one, users can upload files, character set mismatches (alone unimportant, but together with user-contributed content this suddenly becomes dangerous!)
 - Investigate in detail the first time you can
 - Later on: Check for modifications only!
 - » Something new, something “enlarged” (more files) etc.
 - » Therefore they work best for relatively “static” webpages
 - Meaning that structure and programming remains the same, not necessarily the actual content shown on the pages!



Conclusions

- Very useful, but requires typically a lot of work for configuring
 - Including the first run: Investigate and decide what are false positives or can be ignored
- Only useful if really fully automated
 - Can be ignored completely unless something happens
- More security checks become integrated into development
 - Later on it becomes expensive
 - Big danger: Too many → Disable/auto-ignore them
 - » E.g. Eclipse: Only disabling by type, but must not by instance
 - “Here it is intentional/not a problem, but warn me about all others”

F I M

Questions?

Thank you for your attention!



- Java: FindBugs
<http://findbugs.sourceforge.net/index.html>
- C/C++: Valgrind
<http://valgrind.org/>
- Web: Skipfish
<http://code.google.com/p/skipfish/>
- General: Metasploit
<http://www.metasploit.com/>