



Automating security checks

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>



Agenda

- Why automatization?
- What can be automated?
- Example: Skipfish
- How reliable are these tools?
- Practical examples of searching for vulnerabilities:
 - Information collection with NMap
 - Password cracking (John the Ripper, Ophcrack)
 - Exploit scanning with Nessus



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
 - Professionals 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, huge 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!



Overlap with hacking

- Tools are available to search for vulnerabilities
 - These can be used for identifying the fact, to fix them (good)
 - Or for later exploiting them (bad)
 - It depends on the intention and whose system is scanned
- Note: Various tools exist, which do not only search for vulnerabilities, but also exploit them
 - Injecting code, opening shells etc.
 - These are legally even more “dangerous”!
- Some tools cannot be assigned a “good” or “bad” class
 - E.g. password cracking: The SW does exactly the same, and only the interpretation of the result/actions differs
- Here special care about the legality of the actions is needed
 - Clear (ideally: written) permission by the owner of the system



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 one
 - » Still: **Every** single issue must be investigated in detail!
- Typically static analysis, but need not be
 - Adding code for test runs, which identifies runtime problems
- Examples:
 - Using unsafe methods (“sprintf” instead of “snprintf”)
 - Access to shared variable from multip. threads without locking
 - Accessing non-reserved memory; memory not freed
 - 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 fall through: `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



Web: Various problems

- 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 certificate 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: SpamHaus, 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!



Web: Malware injection

- 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



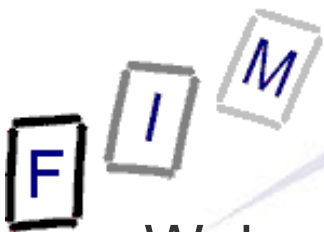
Suspicious account activity

- 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 later)
- 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!



Skipfish: How to scan

- 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!

The screenshot shows the Skipfish scan results browser interface. At the top, the title bar reads "Skipfish - scan results browser". Below the title bar, the Skipfish logo is displayed on the left. To the right of the logo, the scanner version (1.33b) and random seed (0x83340d23) are shown, along with the scan date (Mon May 3 07:47:41 2010) and total time (88 hr 45 min 51 sec 492 ms). A link for "Problems with this scan? Click here for advice." is also present.

The main content area is divided into three sections:

- Crawl results - click to expand:** This section shows a list of crawled URLs. The first URL is <http://www.██████████/>, which is marked with a green plus icon and a red circle containing the number 6. Below the URL, the code (200), length (16108), declared type (text/html), detected type (application/xhtml+xml), and charset (iso-8859-1) are listed, along with a link to "show trace +".
- Document type overview - click to expand:** This section provides a summary of the document types found during the scan. The list includes: 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), and text/xml (3).
- Issue type overview - click to expand:** This section lists the types of issues found. The first issue is "PUT request accepted (5)", which is marked with a red circle containing the number 5. Below this, five specific URLs are listed, each with a link to "show trace +". The other issues listed are "SQL query or similar syntax in parameters (1)", "Interesting file (1)", and "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 may suddenly becomes dangerous!)
 - Investigate in detail the first time
 - 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!



- NMap (Network MAPper) is a network scanner
 - It tries to find all computers in a specific network and checks, what ports are open, what OS they are running, whether there is a firewall, etc.
- It does not look for specific vulnerabilities!
 - But it gives recommendations; e.g. services to disable
 - Some scans + vuln. systems → Lock-up/crash!
- Used as a tool for inventory generation in a network
 - Are there any computers which should not be there?
 - Can also be used to gather information for a later attack
 - » Which OS/software and which version is running
- Stages: 1 = Host discovery, 2 = Port scan, 3 = Service/version detection, 4 = OS detection, 5 = Scripting
 - Scripting may also include vulnerability/malware detection!



- Usage: Trivial!
 - Start program and enter IP address
 - Select profile for scanning
 - » Special options only available in the command line version or when constructing a new profile!
- More complex options:
 - Stealth scans
 - » Trying to not show up on various statistics



Sample result: NMap local subnet scan

Zenmap

Scan Tools Profile Help

New Scan Command Wizard Save Scan Open Scan Report a bug Help

Operating System Detection on 140.78.100.31 X Regular Scan on 140.78.100.244 X Regular Scan on 140.78.100.128/25 X

Target: 140.78.100.128/25 Profile: Regular Scan Scan

Command: nmap -v 140.78.100.128/25

Hosts Services Ports / Hosts Nmap Output Host Details Scan Details

OS Host

r1-intern.fim.uni-linz.ac.at
hp2626-1a.fim.uni-linz.ac.at
hp2824-1a.fim.uni-linz.ac.at
hp2824-2a.fim.uni-linz.ac.at
levelone1.fim.uni-linz.ac.at
hplj4100dtn.fim.uni-linz.ac.at
hplj4100dtn-2.fim.uni-linz.ac.at
jrm_w2k.fim.uni-linz.ac.at
son_wxp.ads-fim.fim.uni-linz.ac.at
koeck.fim.uni-linz.ac.at
140.78.100.182
susanne_xp.ads-fim.fim.uni-linz.ac.at
hoer_xp64.fim.uni-linz.ac.at
alex_w2k.fim.uni-linz.ac.at
bartpe-16534 140.78.100.182
inge_stap04pc.ads-fim.fim.uni-linz.ac.at

ports)
Host [koeck.fim.uni-linz.ac.at](#) (140.78.100.172) appears to be up ... good.
Interesting ports on [koeck.fim.uni-linz.ac.at](#) (140.78.100.172):
Not shown: 1710 filtered ports
PORT STATE SERVICE
139/tcp open netbios-ssn
445/tcp open microsoft-ds
5001/tcp open complex-link
8009/tcp open ajp13
8080/tcp open http-proxy
MAC Address: 00:04:75:76:22:22 (3 Com)

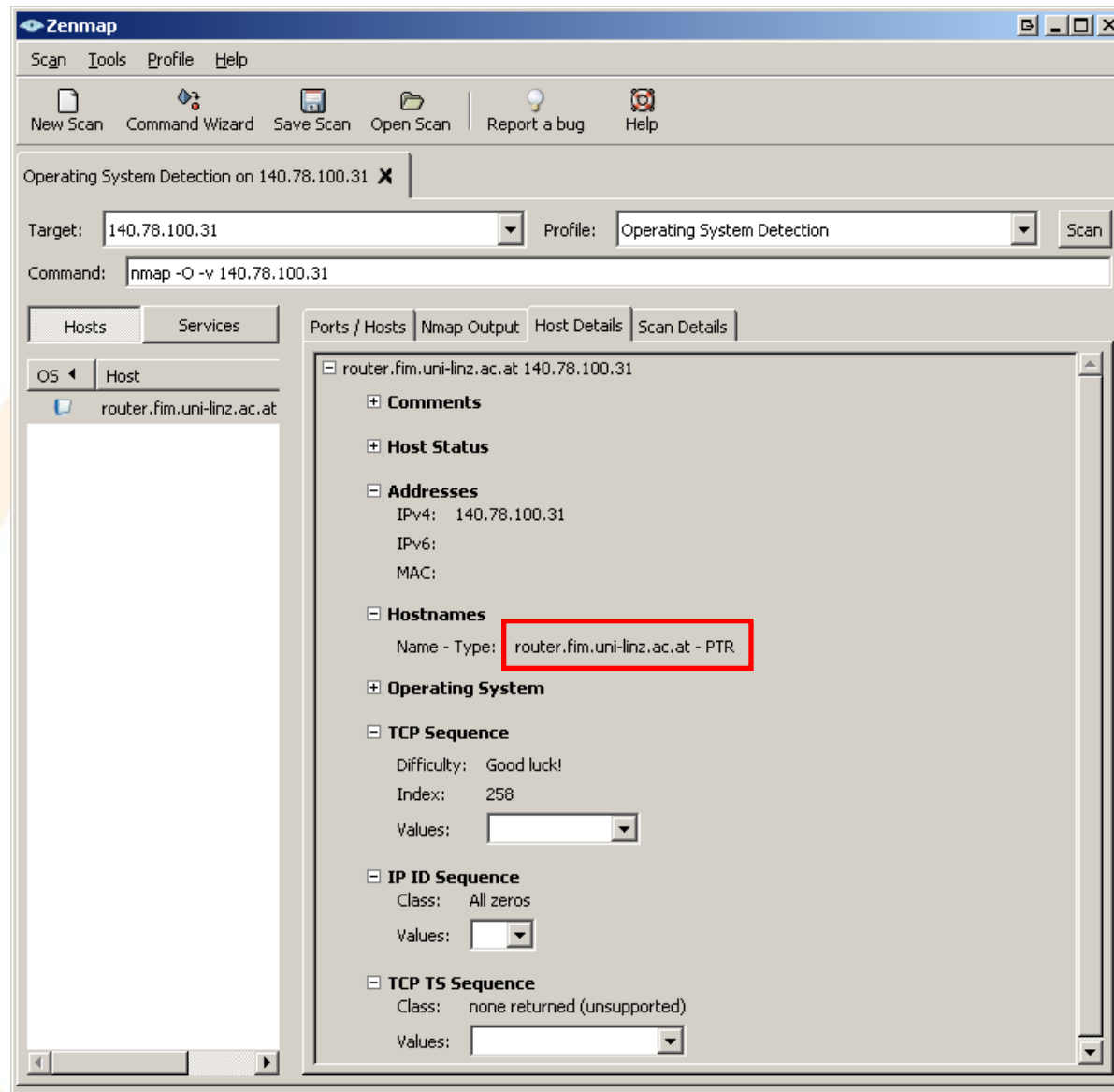
Host 140.78.100.182 appears to be up ... good.
Interesting ports on 140.78.100.182:
Not shown: 1712 filtered ports
PORT STATE SERVICE
135/tcp open msrpc
139/tcp open netbios-ssn
445/tcp open microsoft-ds
MAC Address: 00:14:22:3F:3B:40 (Dell)

Host [susanne_xp.ads-fim.fim.uni-linz.ac.at](#) (140.78.100.199) appears to be up ... good.
Interesting ports on [susanne_xp.ads-fim.fim.uni-linz.ac.at](#) (140.78.100.199):
Not shown: 1712 filtered ports
PORT STATE SERVICE
139/tcp open netbios-ssn
443/tcp open https

☒ Enable Nmap output highlight Preferences Refresh

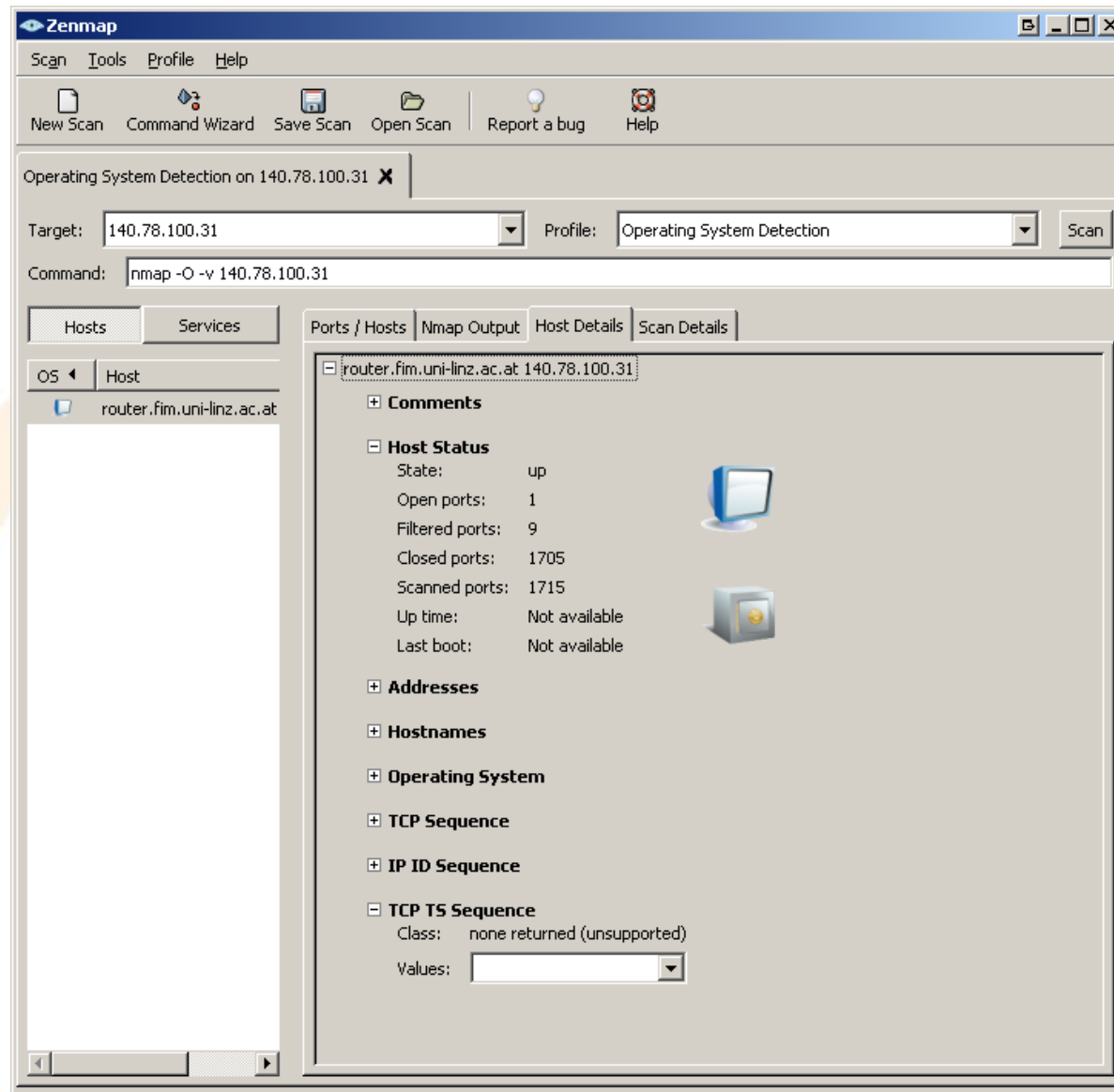


Sample result: NMap OS detection





Sample result: NMap OS detection





Sample result: NMap OS detection

The screenshot shows the Zenmap application window. The main panel displays the results of an NMap OS detection scan on the target 140.78.100.31. The 'Host Status' section shows the host is up. The 'Operating System' section shows the detected OS is 'Cisco C3500XL switch (IOS 12.0(5))' with a 91% accuracy. The 'OS Class' section shows a table of possible OSes, with 'router Cisco IOS 12.X' highlighted. The 'TCP Sequence' section shows the sequence number is 140.78.100.31.

Target: 140.78.100.31 Profile: Operating System Detection Scan

Command: nmap -O -v 140.78.100.31

Hosts Services

OS Host

router.fim.uni-linz.ac.at

Host Status

Addresses

Hostnames

Operating System

Name: Cisco C3500XL switch (IOS 12.0(5))

Accuracy: 91%

Ports used

Port-Protocol-State: 514 - tcp - open

Port-Protocol-State: 1 - tcp - closed

OS Class

Type	Vendor	OS Family	OS Generation	Accuracy
switch	Cisco	IOS	12.X	96%
switch	Cisco	IOS	12.2	93%
switch	Cisco	embedded		85%
router	Cisco	IOS	12.X	85%

TCP Sequence

IP ID Sequence

TCP TS Sequence



- Password cracking tool
 - Uses word lists as well as brute-force
 - » Word lists can be "multiplied" by mangling rules (reverse, ...)– Note: Long lists take longer, but provide better chances!
 - » Brute force: Define character set and set password length limit
 - Can also be used as password-strength checking module
 - "Reconstructs" the password from its hash
 - » Therefore requires access to the password file!
 - Can be interrupted and restarted (may take a long time!)
- Supported are the following password hash types
 - crypt(3) hash types: traditional & double-length DES-based, BSDI extended DES-based, FreeBSD MD5-based (also used on Linux, Cisco IOS), OpenBSD Blowfish-based (also used on some Linux distr.), Kerberos/AFS, Windows NT/2000/XP LM DES-based
 - » More with additional patches!



- Password cracking tool for Windows
 - LAN Manager/NT LAN Manager hashes (i.e. Win passwords)
 - » LM / NTLM hashes (not stored in cleartext, but as hash only)
 - » Windows Vista has the (easier) LM hashes disabled by default
 - Older versions still store the weak LM for backwards compatibility
 - Can import the hashes from various formats or read it directly
- Based on Rainbow tables and brute force
 - Some are freely available, others cost money
 - » You could theoretically create them yourself, but this is an extremely time- and resource-intensive activity!
 - Free tables: About 99.9 % coverage for alphanumeric passwords of up to 14 characters (LM), 99% for NTLM
 - » All printable chars/symbols/space (NT/Vista); German →á US\$ 99

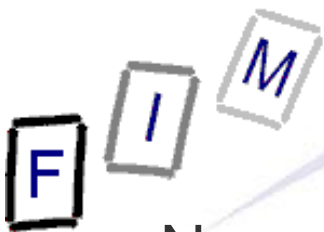


Rainbow tables

- Reducing time by investing memory
 - "Pre-computed passwords"
- Simplest form: Generate all passwords + their hashes and store them for later lookup (immediate cracking!)
 - Drawback: Gigantic table!
- Rainbow tables: Compute all passwords, but store only a small part of them → After finding the hash, some time is required to obtain the actual password
 - Time is reduced by the square of the available memory
- Countermeasure: Use "salting"
 - A random value is generated, prepended to the password, and stored
 - Rainbow table would have to be enlarged for the salt
 - » 4 char salt + 14 char password → 18 char rainbow table!



- Windows password hashes have several problems
 - LM are effectively 2 passwords of 7-characters
 - LM passwords are converted to uppercase
 - LM and NTLM do not employ any "salting"
 - » This is why rainbow tables are feasible here!
- How to disable at least the especially weak LM hashes:
 - » Attention: Will not allow connecting from Windows ME/98/... computers any more!
 - » Disabled by default on Windows Vista
 - Set the registry key
HKLM\SYSTEM\CurrentControlSet\Lsa\NoLMHash to 1



- Nessus is a scanner for vulnerabilities
 - Based on signatures → Finds only known problems!
 - » Currently about 41500 plugins
 - No installation on FAT disks → Too many files in a single directory!
- Updating the signatures: Possible/Automatic
- First step: Identify OS → Almost all vuln. depend on this
 - Registry, SNMP, ICMP, MSRPC, NTP
- Second step: Check which vuln. might apply and test them
 - Not by actually exploiting them, only whether it would work!
- From where to run the scan?
 - Outside: Probably already safe, best to be sure
 - Inside (Critical machines): Defence in depth
 - DMZ: One computer was hacked → Others still secure?
- Commercial use/additional functionality → You have to pay!
 - US\$ 1200 per scanner per year



- Nessus is separated into a daemon and a client
 - Scanning is done by the daemon(s); the client is just an UI
 - Can do more intensive scanning if provided credentials for logging on to a computer
- Vulnerabilities are scripted in NASL
 - Nessus Attack Scripting Language (see next page)
 - » You can write your own too!
 - Detection is not perfect: False positives may occur
- Attention: Some scans can crash the target!
 - Take care before enabling "all" scans!
 - Option "Safe checks" disables anything dangerous and checks through banners only; no actual trying
- Found a vulnerability? Fix it!
 - Prioritize the problems detected
 - Bugtraq ID or CVE number for obtaining further information



Nessus: NASL example (phpcms_xss.nasl)

```
if(description)
{
  script_id(15850);
  script_version("$Revision: 1.5 $");
  script_cve_id("CVE-2004-1202");
  script_bugtraq_id(11765);

  script_name(english:"phpCMS XSS");

  desc["english"] = "
The remote host runs phpCMS, a content management system
written in PHP.

This version is vulnerable to cross-site scripting due to a lack of
sanitization of user-supplied data in parser.php script.
Successful exploitation of this issue may allow an attacker to execute
malicious script code on a vulnerable server.

Solution: Upgrade to version 1.2.1pl1 or newer
Risk factor : Medium";

  script_description(english:desc["english"]);
  script_summary(english:"Checks phpCMS XSS");
  script_category(ACT_GATHER_INFO);
  script_copyright(english:"This script is Copyright (C) 2004 David Maciejak");
  script_family(english:"CGI abuses : XSS");
  script_require_ports("Services/www", 80);
  script_dependencie("http_version.nasl", "cross_site_scripting.nasl");
  exit(0);
}
```

```
include("http_func.inc");
include("http_keepalive.inc");

port = get_http_port(default:80);
if ( ! get_port_state(port))exit(0);
if ( ! can_host_php(port:port) ) exit(0);

if ( get_kb_item("www/" + port + "/generic_xss") ) exit(0);

buf = http_get(item:"/parser/parser.php?file=<script>foo</script>",
  port:port);
r = http_keepalive_send_recv(port:port, data:buf, bodyonly:1);
if( r == NULL )exit(0);

if(egrep(pattern:"<script>foo</script>", string:r))
{
  security_warning(port);
  exit(0);
}
```




Nessus: Sample results

Nessus : C:/Documents and Settings/SONNTAG.ADS-FIM/My Documents/Tenable/Nessus Client

File Help

TENABLE NESSUS 3



Scan Report

Report: 08/05/29 12:10:37 PM - Complete [Delete] [Export...]

140.78.100.166

- general/icmp
- general/tcp
- general/udp
- ntp (123/udp)
- netbios-ssn (...)
- microsoft-ds (...)
- telexis-vtu (2...)
- epmap (135/t...)
- netbios-ns (1...)

140.78.100.166

Scan time :

Start time : Thu May 29 12:10:55 2008
End time : Thu May 29 12:25:12 2008

Number of vulnerabilities :

Open ports :	6
Low :	34
Medium :	6
High :	7

Information about the remote host :

Operating system : Microsoft Windows XP Professional (German)
NetBIOS name : SON_ACER8
DNS name : son_acer8.fim.uni-linz.ac.at.

[Disconnect]



Nessus: Sample results

The screenshot shows the Nessus 3 client window. The title bar indicates the path: C:/Documents and Settings/SONNTAG.ADS-FIM/My Documents/Tenable/Nessus Client. The interface has a menu bar with 'File' and 'Help'. Below the title is the 'TENABLE NESSUS 3' logo and a yellow eye icon with the word 'Nessus'. There are two tabs: 'Scan' and 'Report'. The 'Report' tab is active, showing a report for '08/05/29 12:10:37 PM - Complete'. On the left is a tree view of scan results for IP 140.78.100.166, with 'general/icmp' selected. The main pane displays details for the 'icmp timestamp request' plugin. It includes sections for Synopsis, Description, Solution, Risk factor, and Plugin output. The 'Plugin output' section lists non-standard timestamps, little endian format, and a 32-second clock difference, along with CVE-1999-0524 and Nessus ID 10114. A 'Disconnect' button is at the bottom left.

Nessus : C:/Documents and Settings/SONNTAG.ADS-FIM/My Documents/Tenable/Nessus Client

File Help

TENABLE
NESSUS 3

Scan Report

Report: 08/05/29 12:10:37 PM - Complete Delete Export...

140.78.100.166

- general/icmp
- general/tcp
- general/udp
- ntp (123/udp)
- netbios-ssn (...)
- microsoft-ds (...)
- telexis-vtu (2...)
- epmap (135/t...)
- netbios-ns (1...)

icmp timestamp request

Synopsis :

It is possible to determine the exact time set on the remote host.

Description :

The remote host answers to an ICMP timestamp request. This allows an attacker to know the date which is set on your machine.

This may help him to defeat all your time based authentication protocols.

Solution :

Filter out the ICMP timestamp requests (13), and the outgoing ICMP timestamp replies (14).

Risk factor :

None

Plugin output :

This host returns non-standard timestamps (high bit is set)
The ICMP timestamps might be in little endian format (not in network format)
The difference between the local and remote clocks is 32 seconds

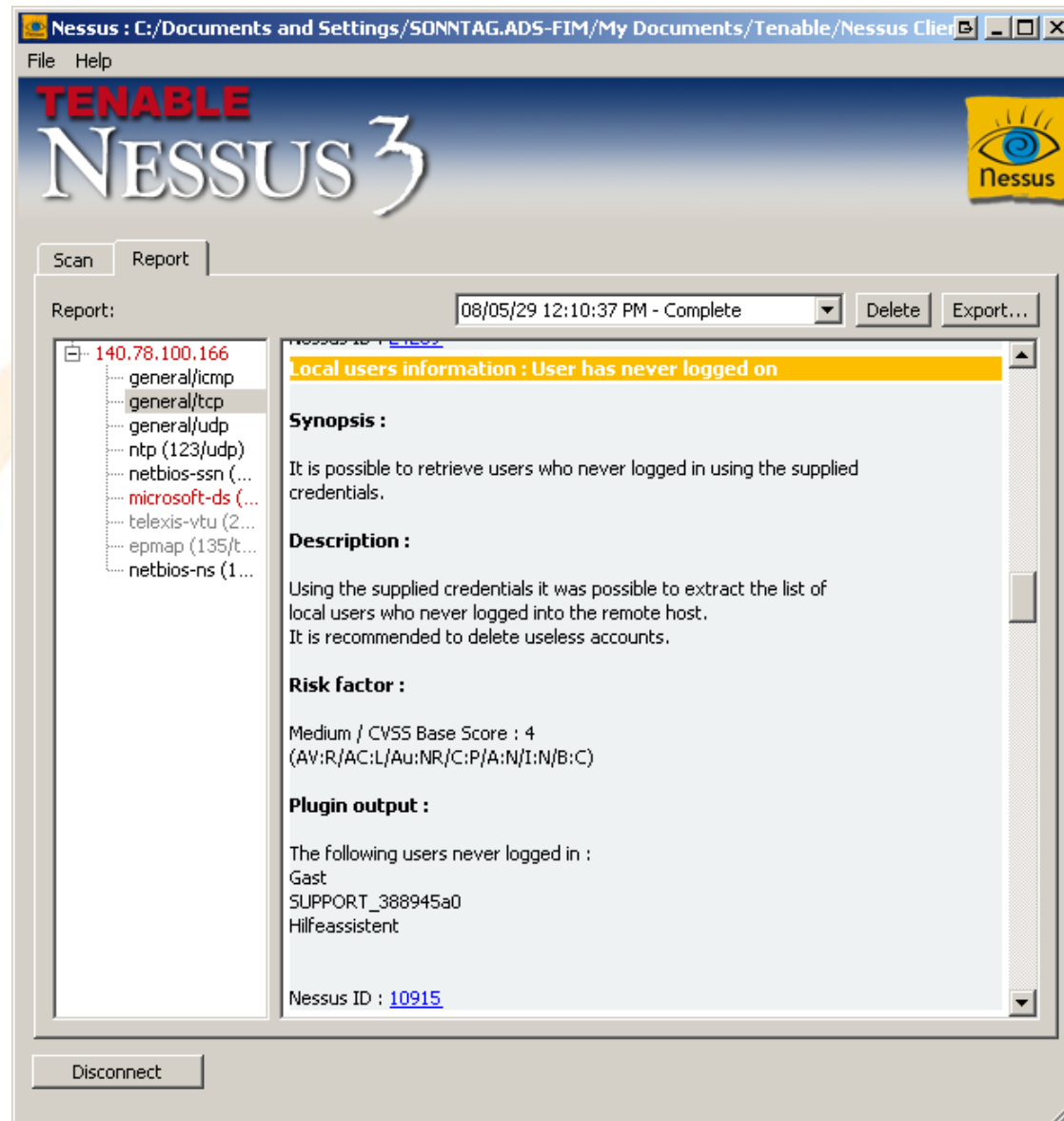
CVE : CVE-1999-0524

Nessus ID : [10114](#)

Disconnect



Nessus: Sample results





Nessus: Sample results

140.78.100.166
general/icmp
general/tcp
general/udp
ntp (123/u...
netbios-ss...
microsoft-...
telexis-vtu...
epmap (13...
netbios-ns...

Flash Player APSB07-12

Synopsis :

The remote Windows host contains a browser plugin that is affected by multiple issues.

Description :

According to its version number, the instance of Flash Player on the remote Windows host could allow for arbitrary code execution by means of a malicious SWF file.

In addition, it may also fail to sufficiently validate the HTTP Referer header, which may aid in cross-site request forgery attacks. This issue does not, though, affect Flash Player 9.

See also :

<http://www.adobe.com/support/security/bulletins/apsb07-12.html>

Solution :

Upgrade to Flash Player version 9.0.47.0 / 8.0.35.0 / 7.0.70.0 or later.

Risk factor :

High / CVSS Base Score : 9.3
(CVSS2#AV:N/AC:M/Au:N/C:C/I:C/A:C)

Plugin output :

Nessus has identified the following vulnerable instance(s) of Flash Player installed on the remote host :

- ActiveX control (for Internet Explorer) :
C:\WINDOWS\system32\macromed\flash\flash.ocx, 7.0.14.0

CVE : CVE-2007-3456, CVE-2007-3457
BID : 24856

Nessus ID : [25694](#)

CVSSv2 (Base metrics only!):

- Access Vector: Network
- Access Complexity: Medium
- Authentication: None
- Confidentiality: Complete
- Integrity: Complete
- Availability: Complete

Result: Base score 9.3

Impact Subscore: 10

Exploitability Subscore: 8.6

CVE-2007-3456:

Integer overflow in Adobe Flash Player 9.0.45.0 and earlier might allow remote attackers to execute arbitrary code via a large length value for a (1) Long string or (2) XML variable type in a crafted (a) FLV or (b) SWF file, related to an "input validation error," including a signed comparison of values that are assumed to be non-negative.



Conclusions

- Automatic checking is 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
 - Ideally the software can compare it against a “baseline” and show only the changes
- 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”

If you are not using this software, the attackers will!

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/>
- Ophcrack:
<http://ophcrack.sourceforge.net/>
- Nessus:
<http://www.nessus.org/>
- General: Metasploit
<http://www.metasploit.com/>