Mag. iur. Dr. techn. Michael Sonntag

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)

Web: Blacklisting

- 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

Web: 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!

Reliability

- 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!

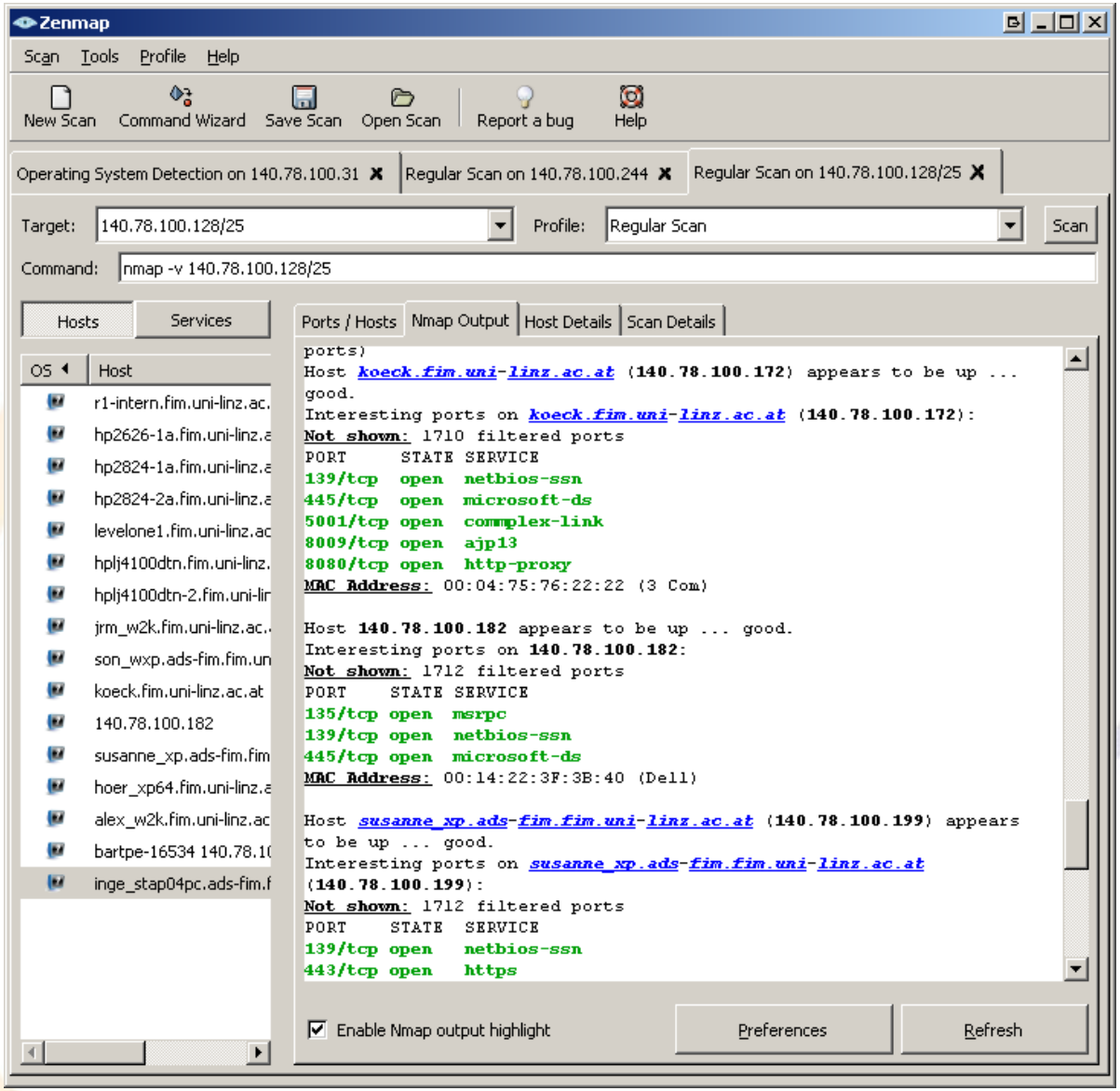
NMap

- 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!

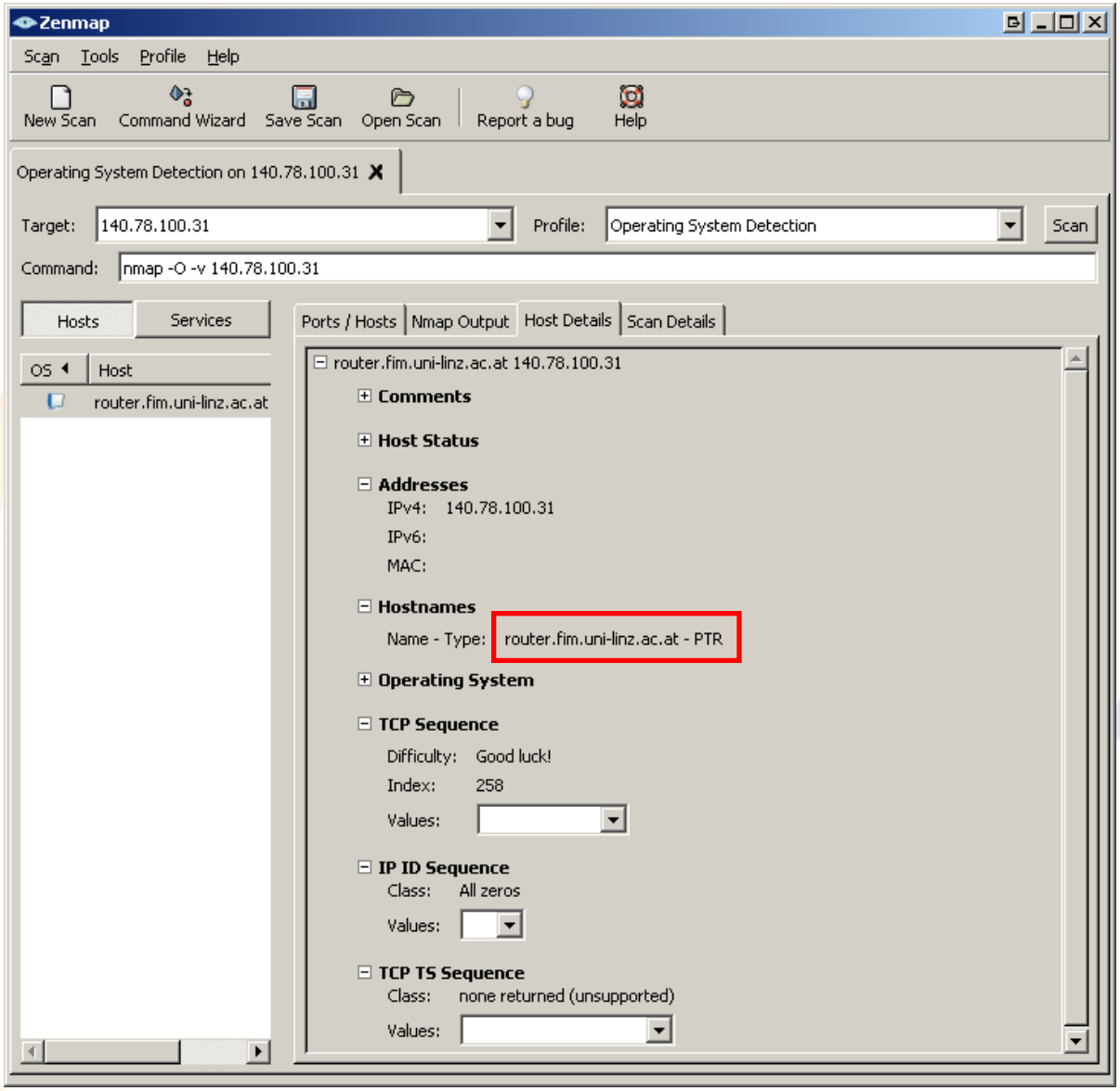
NMap

- 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

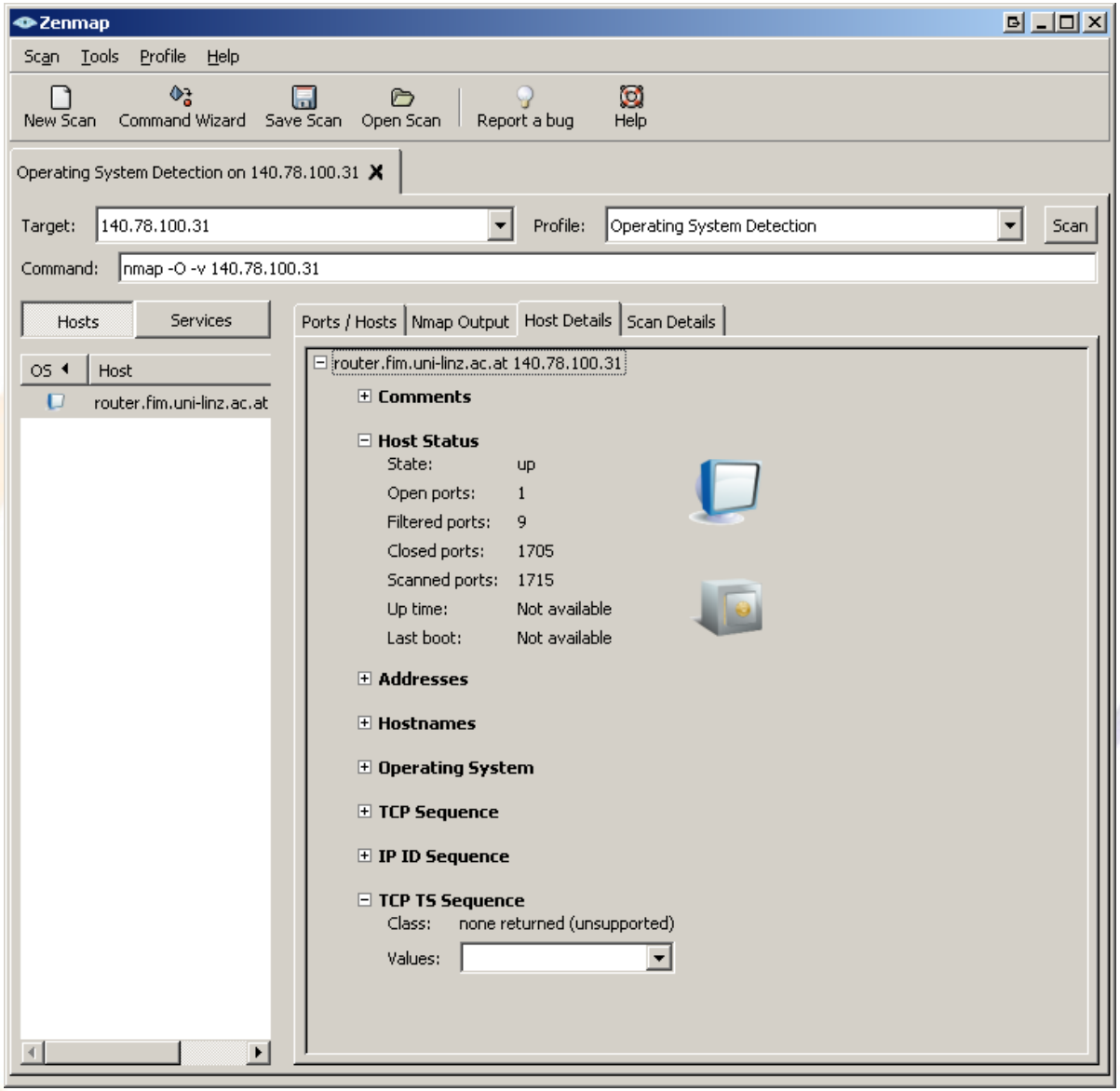
Sample result: NMap local subnet scan



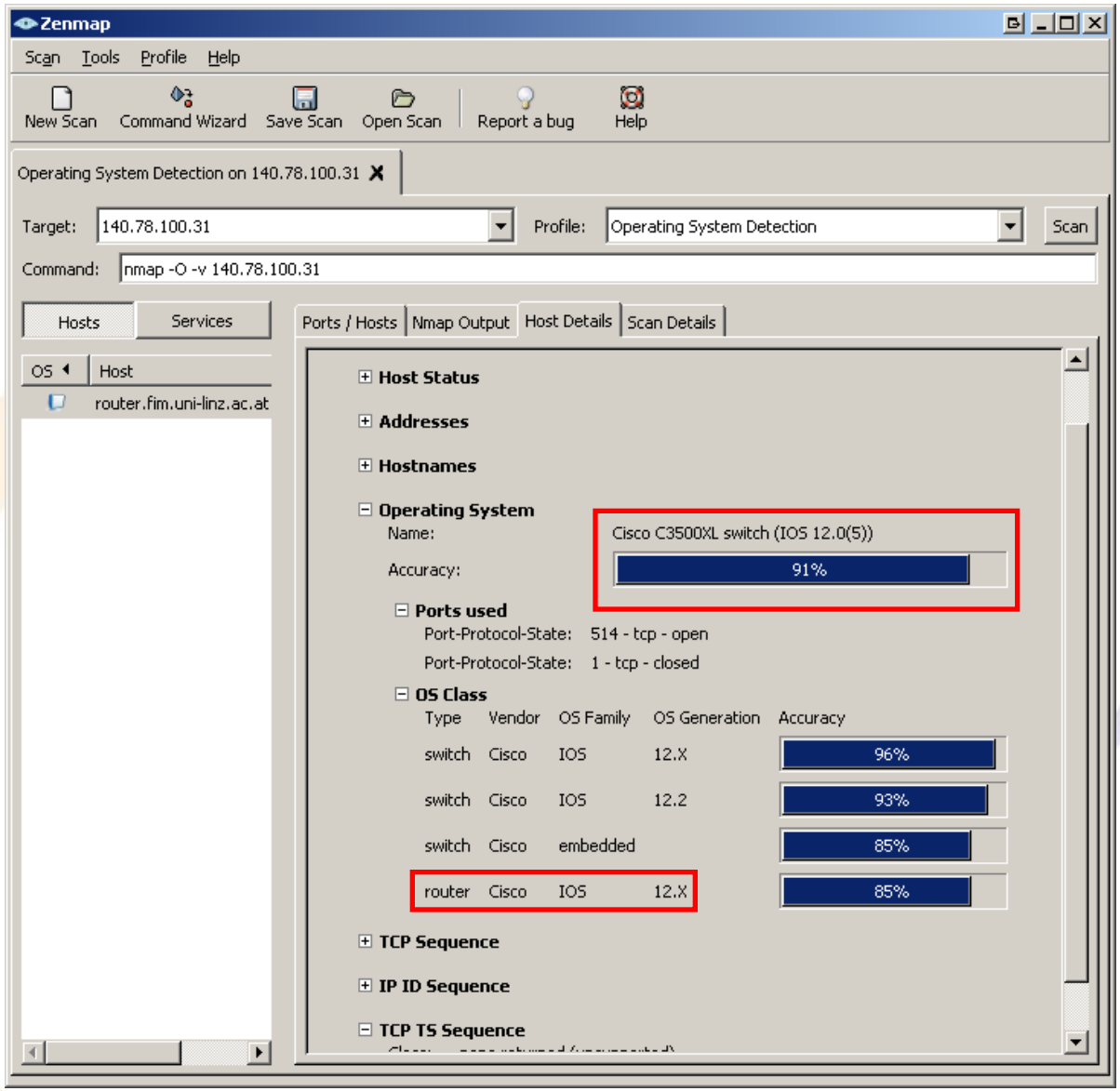
Sample result: NMap OS detection



Sample result: NMap OS detection



Sample result: NMap OS detection



John the Ripper

- 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!

Ophcrack

- 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!

Ophcrack: LM hashes

- 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

- 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

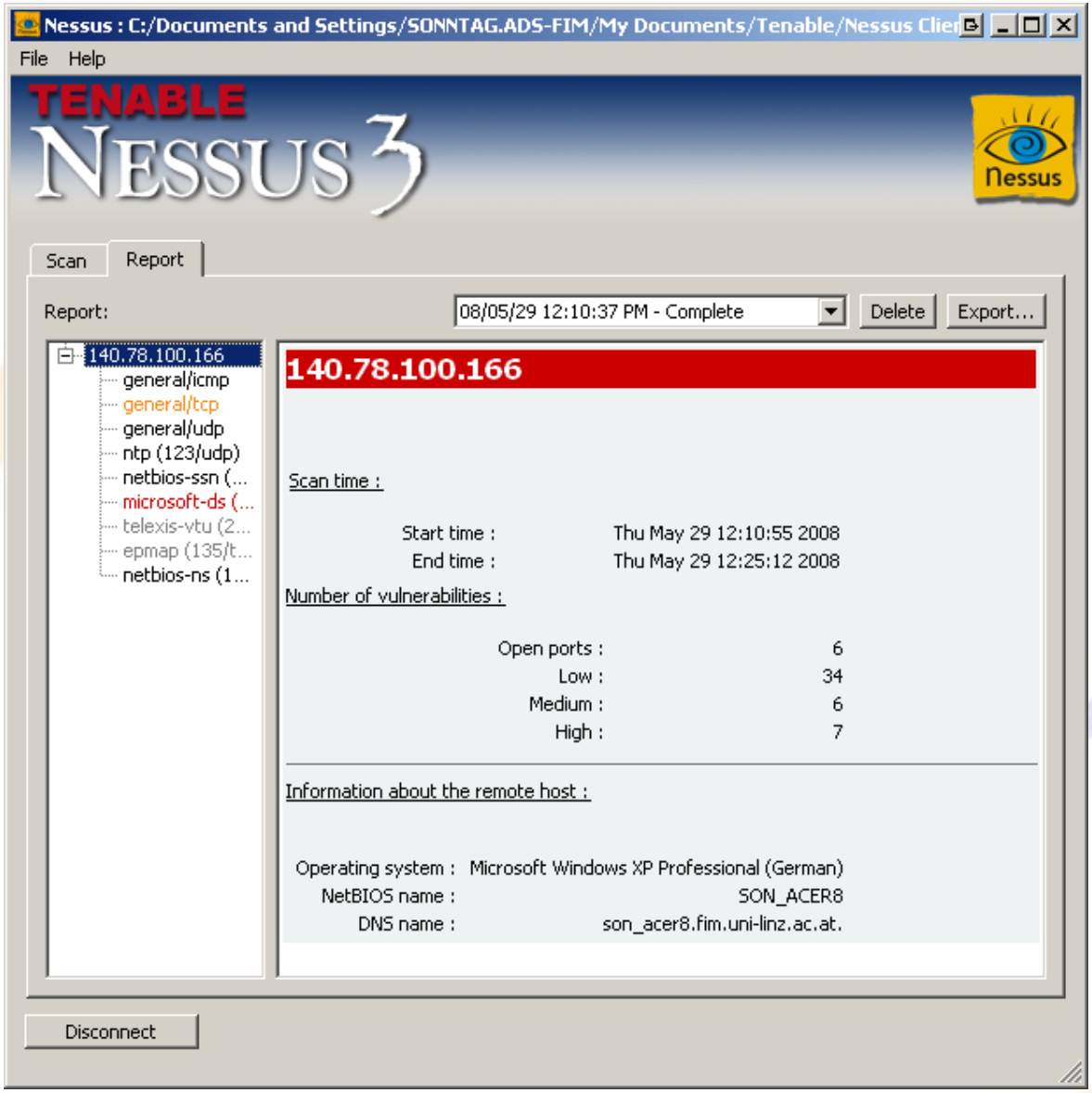
Nessus

- 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 my 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

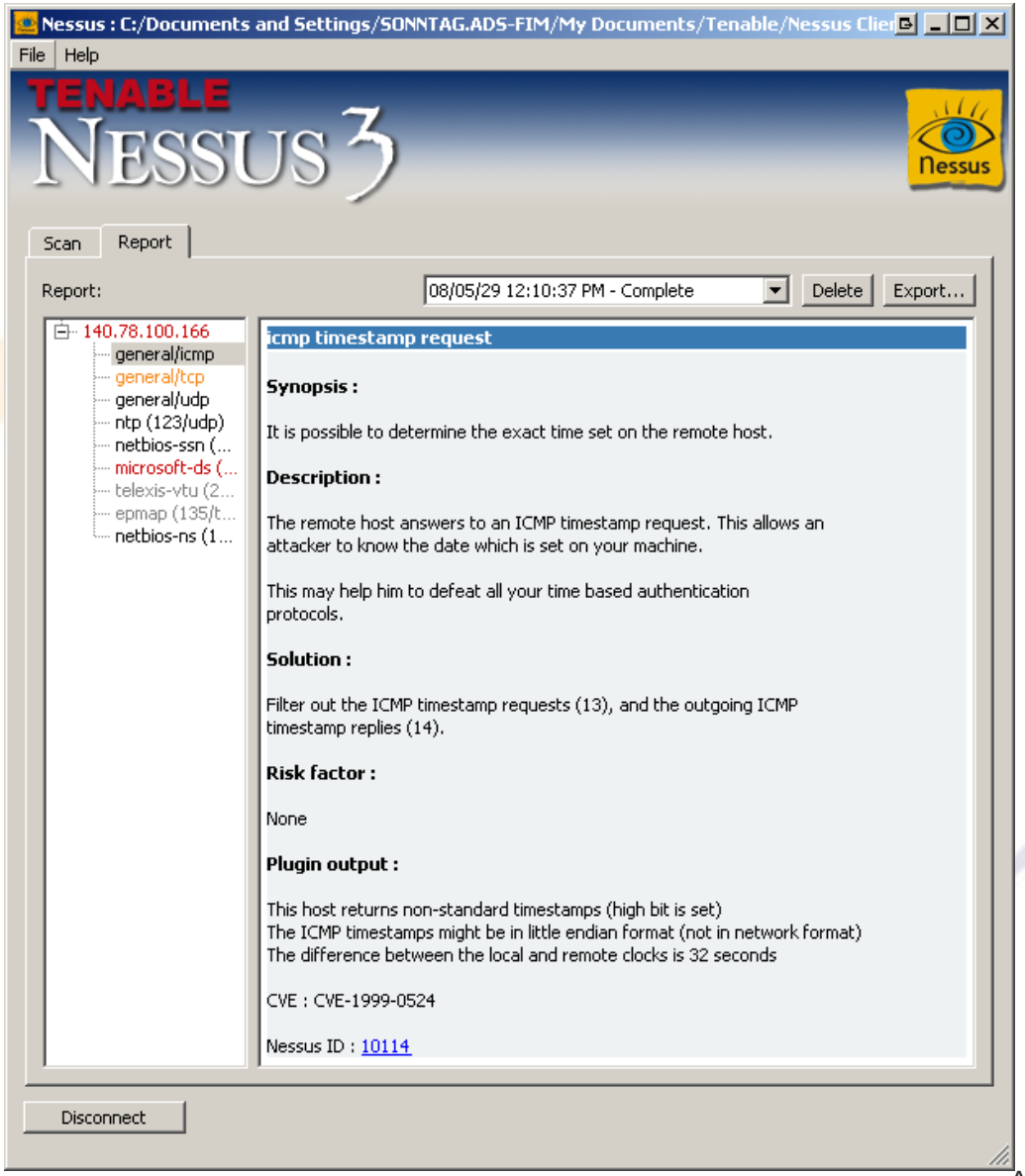
Nessus: NASL example (phpcms_xss.nasl)

```
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);
}
```

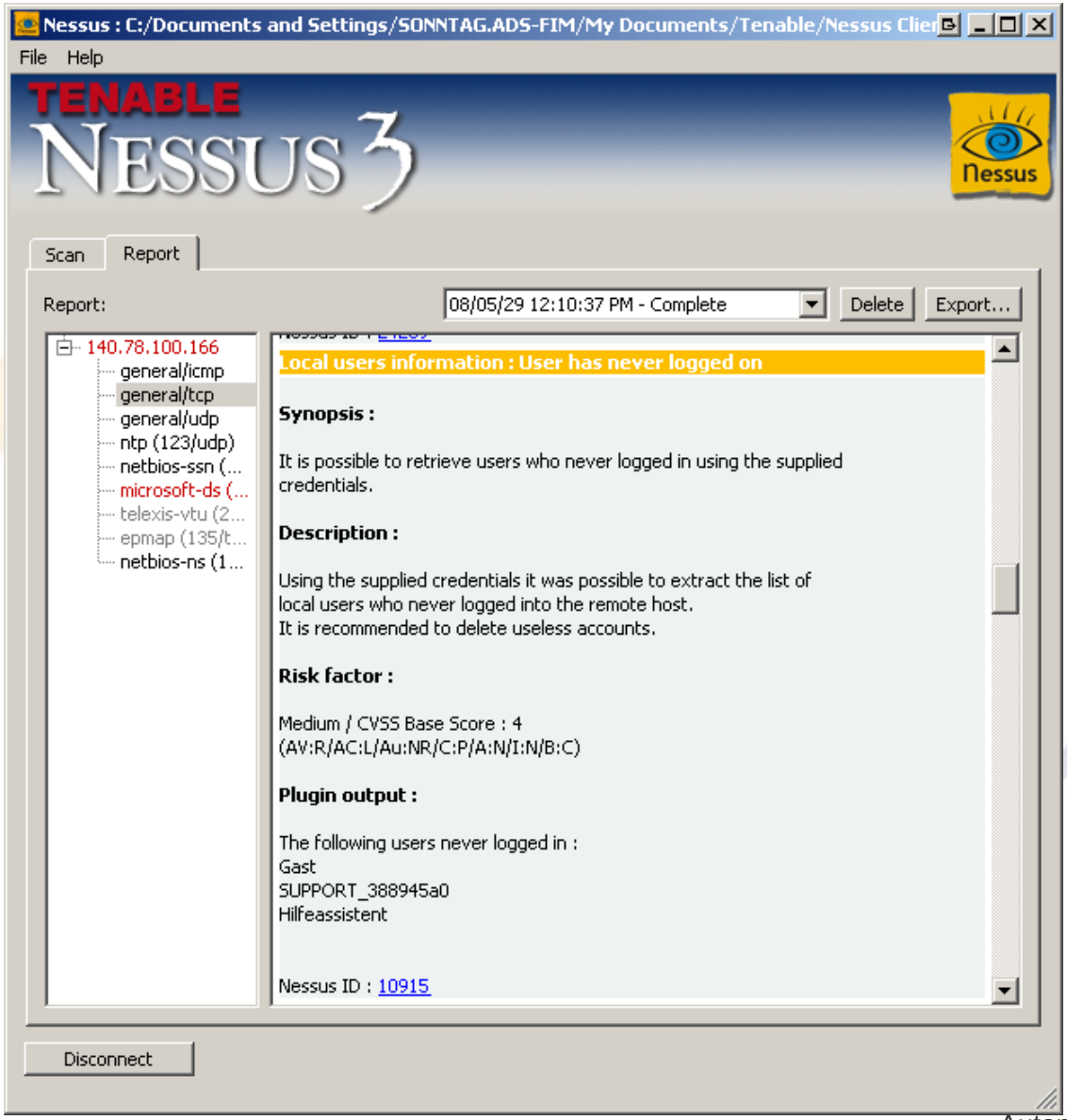
Nessus: Sample results



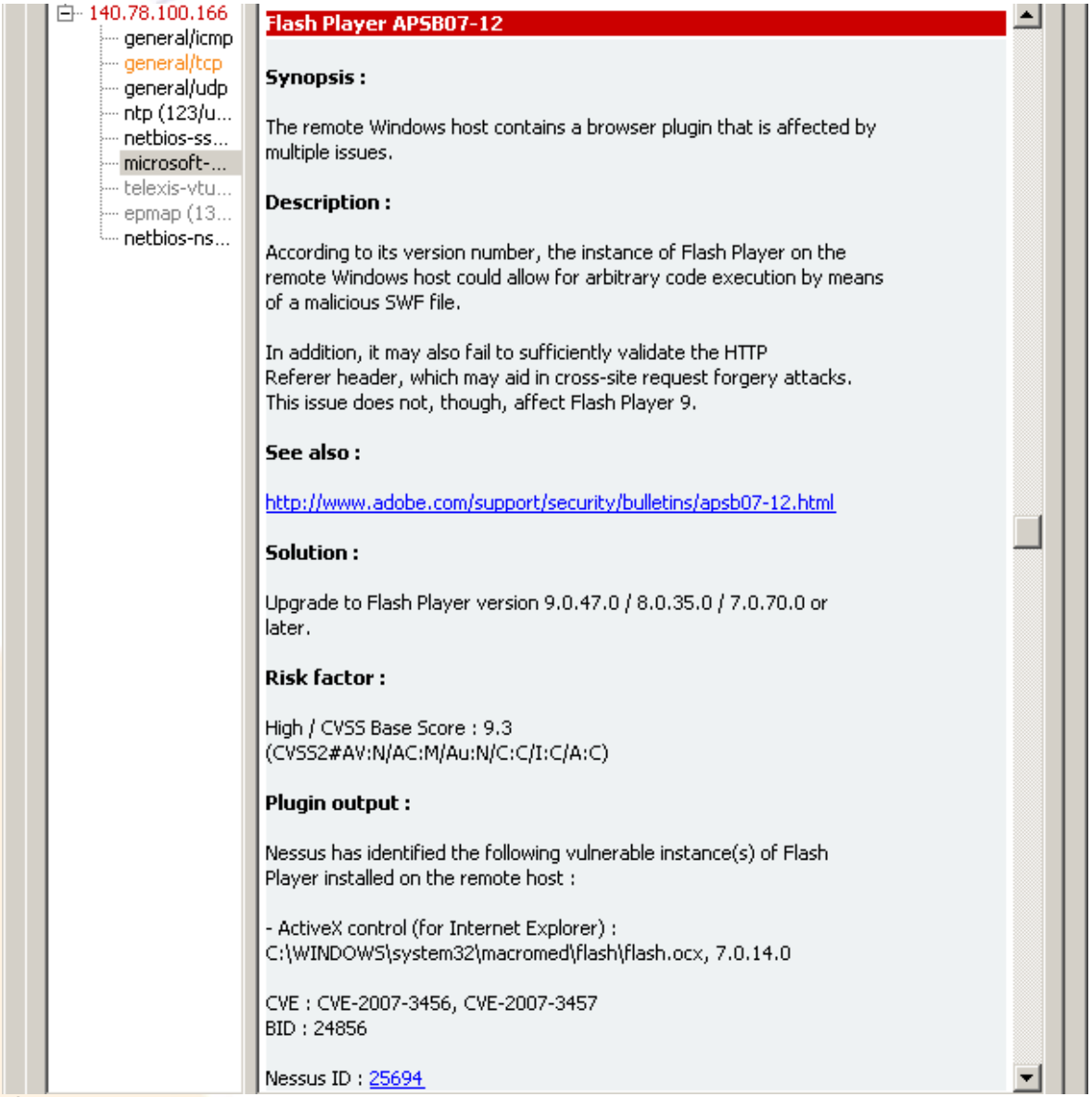
Nessus: Sample results



Nessus: Sample results



Nessus: Sample results



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!

Questions?

Thank you for your attention!

Literature

- Java: FindBugs
  http://findbugs.sourceforge.net/index.html
- C/C++: Valrgind
  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/