



# Protection and security of information systems

## Security check

prof. Ph.D. Krešimir Fertalj

University of Zagreb

Faculty of Electrical Engineering and Computing

Protected by license <http://creativecommons.org/licenses/by-nc-sa/2.5/hr/>



# Creative Commons

---



## □ you are free to:

- **share**—reproduce, distribute and communicate the work to the public
- **remix**—rework the work

## □ under the following conditions:

- **attribution.** You must acknowledge and attribute the authorship of the work in a manner specified by the author or licensor (but not in a manner that suggests that you or your use of their work has their direct endorsement).
- **non-commercial.** You may not use this work for commercial purposes.
- **shares under the same conditions.** If you modify, transform, or create using this work, you may distribute the adaptation only under a license that is the same or similar to this one.

In the case of further use or distribution, you must make clear to others the license terms of this work. The best way to do this is to link to this website.

Any of the above conditions may be waived with the permission of the copyright holder.

Nothing in this license infringes or limits the author's moral rights.

The text of the license was taken from <http://creativecommons.org/>.

---

# **Security check**

Security Testing

---

# Software correctness check (general)

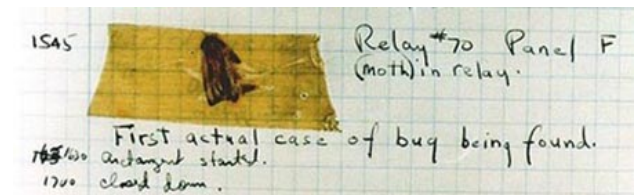
---

- Program testing, program verification, program testing
  - detection of errors or defects within the program
  - the success of the test is proportional to the number of errors found
- According to the purpose of testing
  - Verification - verification of correctness (good implementation)
  - Validation - confirmation of validity (real, acceptable product)
- According to the check object
  - Structural (white-box testing) - source code
  - Functional (black-box) - compilation
- According to the verification method
  - static analysis - no startup, source code or .NET MSIL, Java Bytecode
  - dynamic analysis - by running, which does not exclude the source code

# Key terms

---

- **["normal" ] Test**-verifies that some aspect of the software is correct
- **Security test**-tries to prove that some part is not working properly
- **Error**(error) - an omission by the programmer, eg due to misunderstanding
  - leads to one or more failures
  - we distinguish in relation to "error" which means an unwanted state, i.e. a malfunction
- **Failure**(fault), defect, informally bug - defective part of the code
  - eg wrongly assuming that an array is indexed by 1 instead of 0 causes an array element access failure
- **A standstill** in operation (failure) - condition caused by one or more malfunctions
  - eg system shutdown due to "buffer overrun" error
- **Correction**(Fix) - state of repair

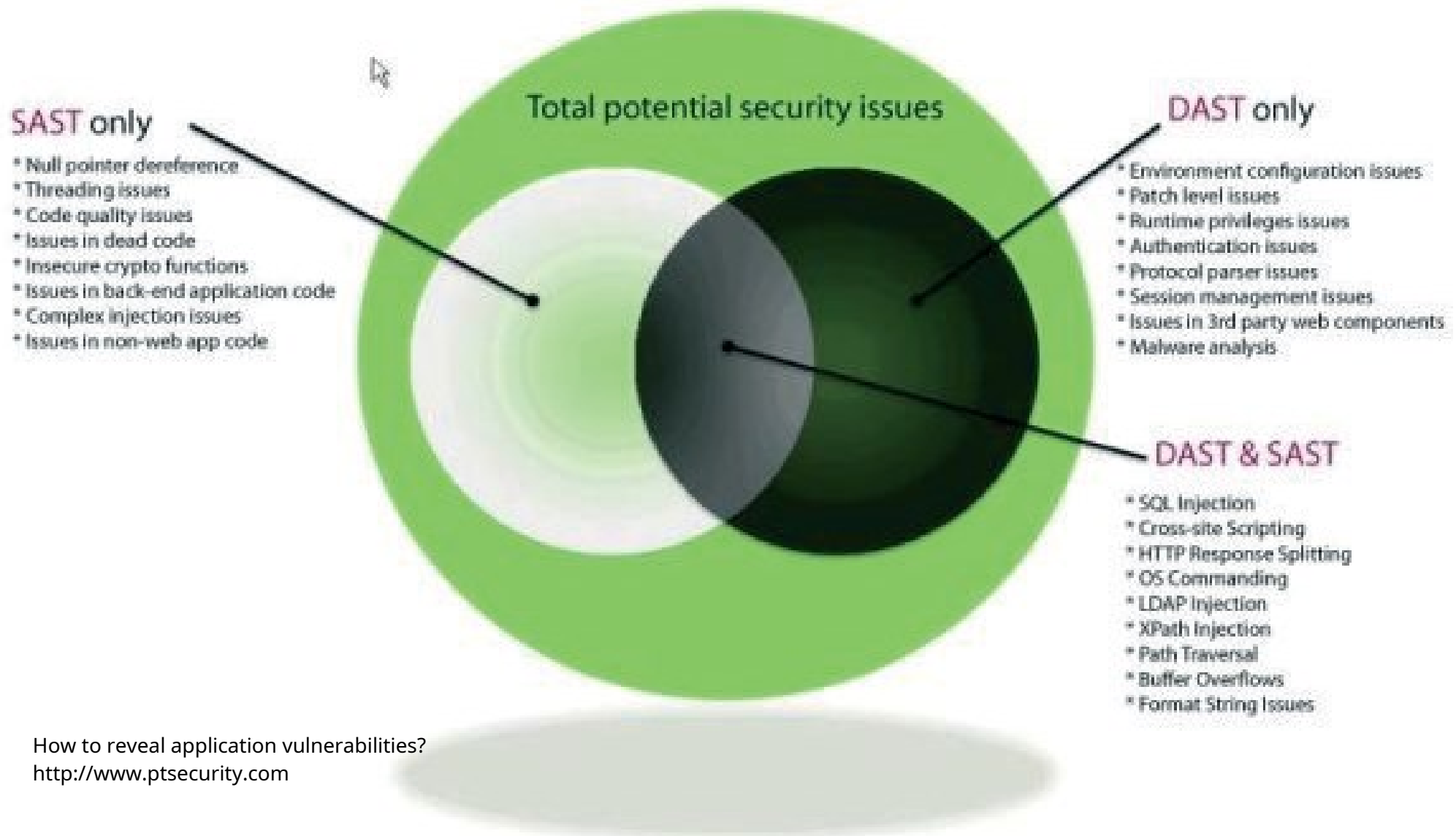


# Application security verification procedures

---

- Supervision ("Technical Reviews", "Code Reviews", "Inspections", ...)
  - testing tends to cause deadlock, monitoring looks for malfunction
- Static Application Security Testing (**SAST**) # white-box static
  - which does not require execution
  - access to the source code on the client and on the server
- Dynamic Application Security Testing (**DAST**) # black-box dynamic
  - requires execution
  - does not have access to the source code and operating environment on the server
- Interactive Application Security Testing (**IAST**) # white-box dynamic
  - dynamic with access to the source code and operating environment on the server
- Source code analysis - static or dynamic with access to the entire code

# How to reveal application vulnerabilities?



---

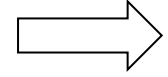
**Supervision**

Security Review, Code Reviews, Inspection



# Revision, review (peer review)

---



## -Variants

- inspection
- Team review
- Walkthrough

## -Use it

- finding defects earlier in the life cycle - up to 80% before testing
- finding defects with less effort than testing
  - IBM - review 3.5 h/defect, test 15-25 h/defect
- finding different defects than by testing - design and requirements problems
- teaching developers - not to repeat the same mistakes

# Inspection

---

## -Formal process

### -Thorough coverage of separate roles

- Moderator - leads the meeting, monitors problems

- The reader - paraphrases (retells) the code, not the author

- Recorder - records defects

- Author - provides code context, explains, fixes after review

### -Checklists for specific objectives

### -Data collection for error tracking

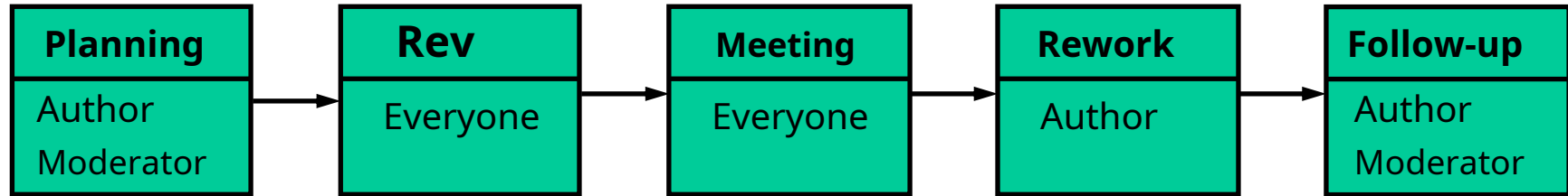
- Determining the need for subsequent inspections

## -Extensive documentation of effectiveness

- 16-20 defects/kLOC inspections vs. 3 defects/kLOC passes

# Inspection process

---



## -Planning

- the author initiates, the moderator equips, the meeting prepares the inspection package

## -Preparation

- reviewers review, use checklists and analytical tools, mark defects

## -Meeting

- the reader recounts, the reviewers comment and question, the recorder records

- the team concludes the code evaluation

## -Processing

- the author fixes

## -Control (follow-up)

- the moderator verifies the correctness of the changes, the author reports the code (check-in)

# Variants

---

## -Team review

- Team inspection ("light" inspection)
- Persons: moderator, reviewers (who are not code authors)
- Modules or smaller sets of classes
- 1-2 hours, < 1 kLOC

## -Walkthrough

- The author leads the meeting and explains the code
- A less formal process
- Undefined process
- No checklists or metrics

### -Pair programming

- Guide (driver) and observer (observer, navigator)
- Change of roles and partners

### -*Peer deskcheck*

- Only one reviewer with the author
- informal review
- may include checklists and other procedures

### -*Pass around*(circular addition?)

- multiple, simultaneous*Peer deskcheck*
- multiple reviewers (same code)

---

# Static check

Static Analysis

# Static analysis

---

## -SAST (Quick and Dirty)

- Code analysis without execution
- It covers everything but testing
- Using the code analyzer
- It can be part of a code review

## -Limitations: false detection and false non-recognition

- False Positives**-non-existent bugs, powerlessness with complex code or external
- False Negatives**-failure to recognize bugs, complexity of code, weakness of rules

# Types of static analysis

---

- Type checking - part of the programming language
  - ex. `int i = "abc";`
- Style checking - good practices
  - Rules - spaces/spaces, nomenclature, comments, ...
- Program understanding - inferring meaning
  - All method usage, global variable declaration finding, ...
- Property checking - ensuring that there is no bad behavior
  - For example memory leak : `if (malloc() == NULL)`
- Program verification (Program verification) - ensuring correct behavior
  - For example `free(mem);`
- Bug finding - detection of possible errors
  - Bug patterns



# Mechanisms of static analysis

```
if (state != buf)
  state = buf; {
  strcpy(buf, buf);
  system("other");
}
```



Build  
Model



Perform  
Analysis



Present  
Results

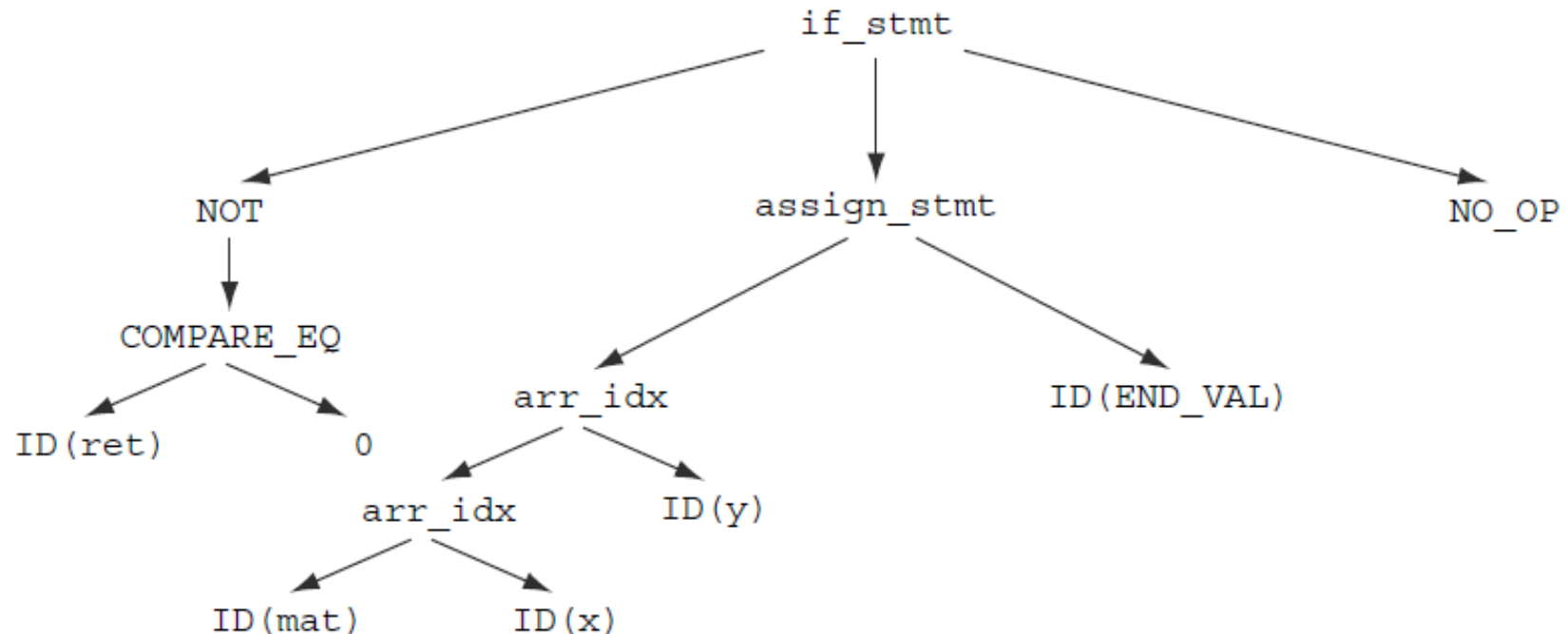


Security  
Knowledge

-Parser, Model Builder, Analysis Engine

-A parser for every programming language

-generates an Abstract Syntax Tree (AST)



### -Lexical analysis and parsing

-Example, program section

```
if (ret) // probably true
    mat[x][y] = END_VAL;
```

-The corresponding sequence of symbols (tokens)

```
IF LPAREN ID(ret) RPAREN ID(mat) LBRACKET ID(x) RBRACKET LBRACKET
ID(y) RBRACKET EQUAL ID(END_VAL) SEMI
```

-Extracted by syntax rules, parsed according to grammar productions

---

if	{ return IF; }
(	{ return LPAREN; }
)	{ return RPAREN; }
[	{ return LBRACKET; }
]	{ return LBRACKET; }
=	{ return EQUAL; }
;	{ return SEMI; }
/[ \t\n]+/	{ /* ignore whitespace */ }
/\//\./ */	{ /* ignore comments */ }
/[a-zA-Z][a-zA-Z0-9]*"/	{ return ID; }

---

---

```
stmt := if_stmt | assign_stmt
if_stmt := IF LPAREN expr RPAREN stmt
expr := lval
assign_stmt := lval EQUAL expr SEMI
lval = ID | arr_access
arr_access := ID arr_index+
arr_idx := LBRACKET expr RBRACKET
```

---

### -Data Flow Analysis

- Collecting information about the data stream when the program is executed while it is stopped

- Semantic analysis - based on AST and symbol table

- Basic block - a sequence of statements that does not stop or branch except at the end

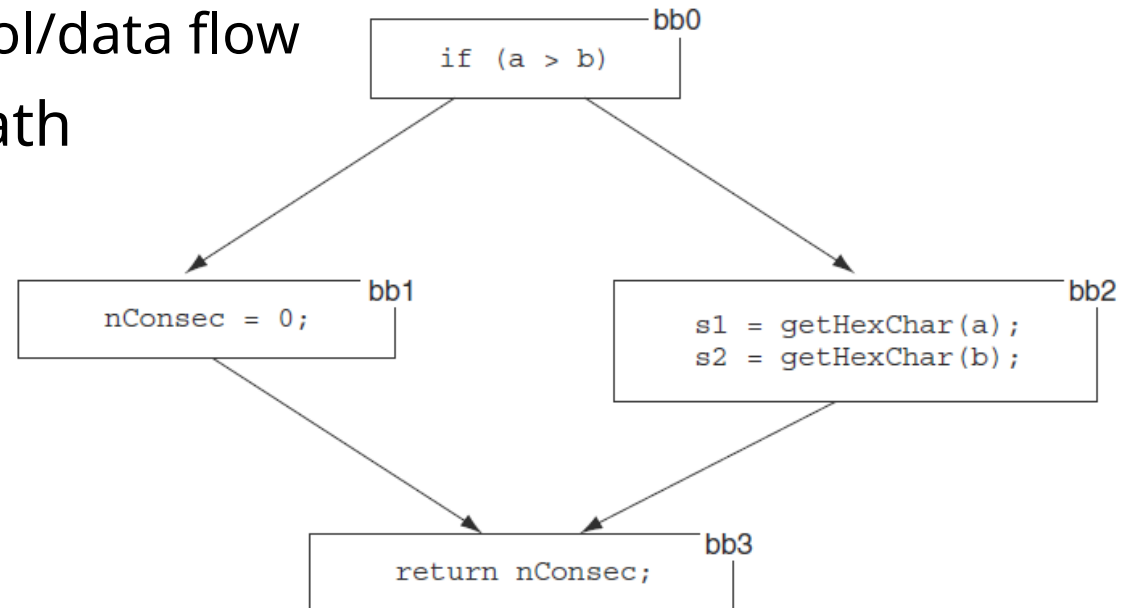
- Control Flow Analysis – control/data flow

- Control Flow Path - data path

### -Flow control chart

- Control Flow Graph (CFG)

- Example: graph with 4 basic blocks



### -Taint Analysis

- Identification of variables *soiled* by user input
- Monitoring their propagation according to possibly vulnerable functions (*son*)
- If they are not disinfected before the drain - vulnerability

### -The rules of blob propagation

#### -Source rules - data entry

-Orders *read()*, *getenv()*, *getpass()*, *gets()*.

#### -Sink rules - locations that should not receive dirty data

-Example, *JavaStatement.executeQuery()*, *Cstrcpy()*

#### -Pass-through rules

-If it is *string* soiled and *trim(string)* will be dirty

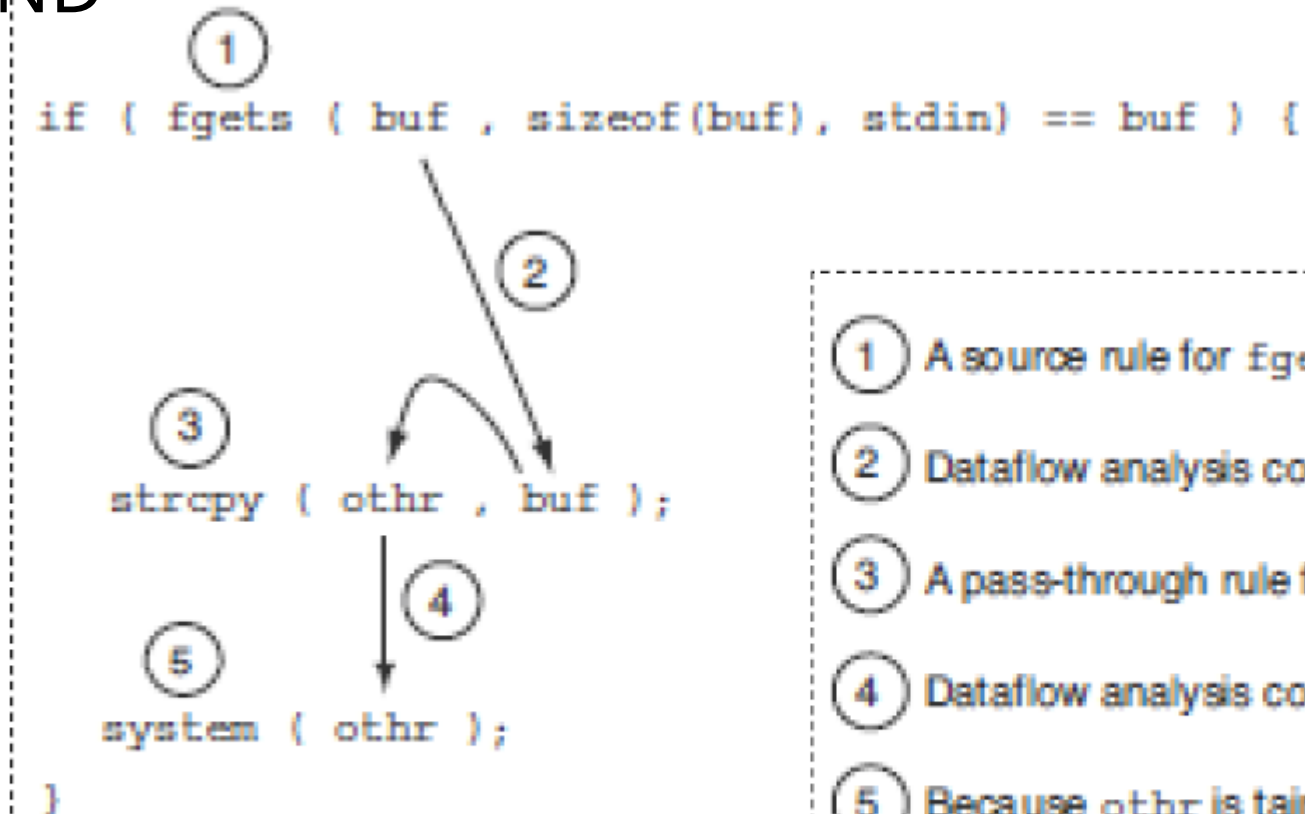
#### -Cleanse rules - input validation

#### -Entry-point rule - similar to the source, e.g. *main(...)*

---

## Example of static analysis

-AND



- ① A source rule for `fgets()` taints `buf` `othr`
- ② Dataflow analysis connects uses of `buf`
- ③ A pass-through rule for `strcpy` taints
- ④ Dataflow analysis connects uses of `othr`
- ⑤ Because `othr` is tainted, a sink rule for `system()` reports a command injection vulnerability

## Advantages and disadvantages of static analysis

---

### -Advantages

- Full code coverage (code coverage) - in theory
- The potential to confirm the absence of entire classes of bugs
- It catches *bugs* different compared to dynamic analysis

### -Weaknesses

- High false detection rate
- Difficult test shaping
- Complexity of construction (tool) - "parser for every language"
  - not enough when using additional frameworks or libraries
- Not having the entire source code in practice (OS, shared libraries, DLLs, ...)

# Tools for static analysis

---

- [https://www.owasp.org/index.php/Source\\_Code\\_Analysis\\_Tools](https://www.owasp.org/index.php/Source_Code_Analysis_Tools)

- StyleCop <https://github.com/StyleCop/StyleCop> - C#
- CodeSmarter <http://www.axtools.com/> - C#, C++, VB.NET
- NDepend <http://www.ndepend.com/> - C#, jDepend for Java
- VS Code Analysis (FxCop, Roslyn analyzers) - C#, C/C++, ...
- PMD - Java, C, C++, C#, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL, ...
- Fortify Source Code Analyzer - 25 languages
- Checkstyle - Java
- Klocwork K7 Suite - Java
- FindBugs, Find Security Bugs - Java
- Coverity Prevent - C#, clang, gcc

# Example: StyleCop

Server Explorer | Toolbox | SQL Server Object Explorer

UserModel.cs

```
1 namespace Web.Models
2 {
3     public class UserModel
4     {
5         public UserModel(){}
6         public UserModel(int ID , int
7             this.ID=ID;
8             this.CinemaID=CinemaID;
9             this.Username=Username;
10            this.Password=Password;
11            this.Name=Name;
12            this.Surname=Surname;
13            this.Role=Role;
14        }
15        public int ID {get;set;}
16        public int CinemaID {get;set;}
17        public string Username {get;s
18        public string Password {get;s
19        public string Name {get;set;}
20        public string Surname {get;se
21        public string Role {get;set;}
22    }
23 }
24 }
```

Error List

Entire Solution 0 Errors 55 Warnings

Code	Description
SA1009	CSharp.Spacing : Invalid spacing arou
SA1012	CSharp.Spacing : Invalid spacing arou
SA1013	CSharp.Spacing : Invalid spacing arou
SA1001	CSharp.Spacing : Invalid spacing arou

StyleCop 5.0 (5.0.6419.0) Project Settings - C:\Users\DodoAcer\Desktop\R...

Rules | Settings Files | Options | Spelling | Company Information | Hungarian | Build Integration

Choose which rules to run on this project. Find Go

Enabled rules

- ☒ C#
- ☐ Documentation Rules
- ☐ Layout Rules
- ☐ Maintainability Rules
- ☐ Naming Rules
- ☐ Ordering Rules
- ☐ Readability Rules
- ☒ Spacing Rules
  - ☒ SA1000: KeywordsMustBeSpacedC
  - ☒ SA1001: CommasMustBeSpacedCo
  - ☒ SA1002: SemicolonsMustBeSpacec
  - ☒ SA1003: SymbolsMustBeSpacedCo
  - ☒ SA1004: DocumentationLinesMustE
  - ☒ SA1005: SingleLineCommentsMustE

Detailed settings

- ☒ Analyze designer files
- ☐ Analyze generated files

Indicates whether to include designer files (\*.Designer.cs).

OK Cancel Apply



# Example: IBM Rational Appscan Source Edition for Security

The screenshot displays the IBM Rational Appscan Source Edition for Security interface. The main window is divided into several panes:

- Findings (19):** A list of findings categorized by severity (High, Vulnerability, ErrorHandling.RevealDetail, Injection). The 'Vulnerability (5)' category is selected.
- Trace:** A table showing the flow of data from the source to the sink. The table has columns: Trace, API, Source, and Sink. The source is `javax.servlet.http.HttpServletRequest.getAttribute` and the sink is `javax.servlet.jsp.JspWriter.print`.
- Trace Details:** A detailed view of the trace, showing the flow of data from the source to the sink. The source is `javax.servlet.http.HttpServletRequest.getAttribute` and the sink is `javax.servlet.jsp.JspWriter.print`.
- Findings Detail:** A pane showing details about the selected finding, including Context, Classification, Vulnerability Type, Severity, and Bundle.
- Remediation Assistance:** A pane providing guidance on how to remediate the finding, including a section on Mitigation.
- Source Code:** A pane showing the source code of the application, with the finding highlighted. The code is in `feedbacksuccess.jsp` and shows a vulnerability in the `String` handling.

The **Mitigation** section in the Remediation Assistance pane provides the following text:

To defend against these problems, the application should apply appropriate [validation](#) and [encoding](#) on the string. The validation mechanism should ensure that the string does not contain malicious data and code. HTML entity encoding should be applied to data that is not intended to be interpreted as scripts. For more details on validation and encoding, please refer to [Validation Required](#) and [Validation Encoding Required](#).

The **Example** section shows the following JSP code snippet:

```
welcome.jsp
<html> welcome
<%= request.getParameter("name")%>
</html>
```

# Example: NDepend

The screenshot displays the Microsoft Visual Studio interface with the NDepend extension. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Architecture, Test, NDepend, ReSharper, Analyze, Window, and Help. The NDepend menu is open, showing options like Dashboard, Rules, Issues, Graph, Matrix, Diff, Trend, Metrics, Coverage, Search, Project, Analyze, Report, Tools, Options..., Windows, and Help. The 'View Explorer Panel' option is selected, which has opened a sub-menu with 'View Explorer Panel', 'View Editor Panel', 'New ...', and a list of quality gates and rules. The 'Queries and Rules Explorer' on the left shows a tree view of project rules, including Quality Gates, Hot Spots, Code Smells, Code Smells Regression, Object Oriented Design, Design, Architecture, API Breaking Changes, Code Coverage, Dead Code, Visibility, Immutability, Naming Conventions, Source Files Organization, .NET Framework Usage, Defining JustMyCode, Trend Metrics, Code Diff Summary, Statistics, and Samples of Custom rules. The 'Rules extracted from Source Code' is also listed at the bottom.

RIS - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Architecture Test NDepend ReSharper Analyze Window Help

Debug Any CPU

Queries and Rules Explorer Dashboard EmployeeProjectWork.cs

Create Group Rule File Delete

Project Rules (318 queries)

- Quality Gates (14 queries)
- Hot Spots (7 queries)
- Code Smells (9 queries)
- Code Smells Regression (9 queries)
- Object Oriented Design (14 queries)
- Design (15 queries)
- Architecture (10 queries)
- API Breaking Changes (9 queries)
- Code Coverage (13 queries)
- Dead Code (4 queries)
- Visibility (11 queries)
- Immutability (13 queries)
- Naming Conventions (19 queries)
- Source Files Organization (6 queries)
- .NET Framework Usage (31 queries)
- Defining JustMyCode (7 queries)
- Trend Metrics (73 queries)
- Code Diff Summary (25 queries)
- Statistics (13 queries)
- Samples of Custom rules (16 queries)

Rules extracted from Source Code (0 query)

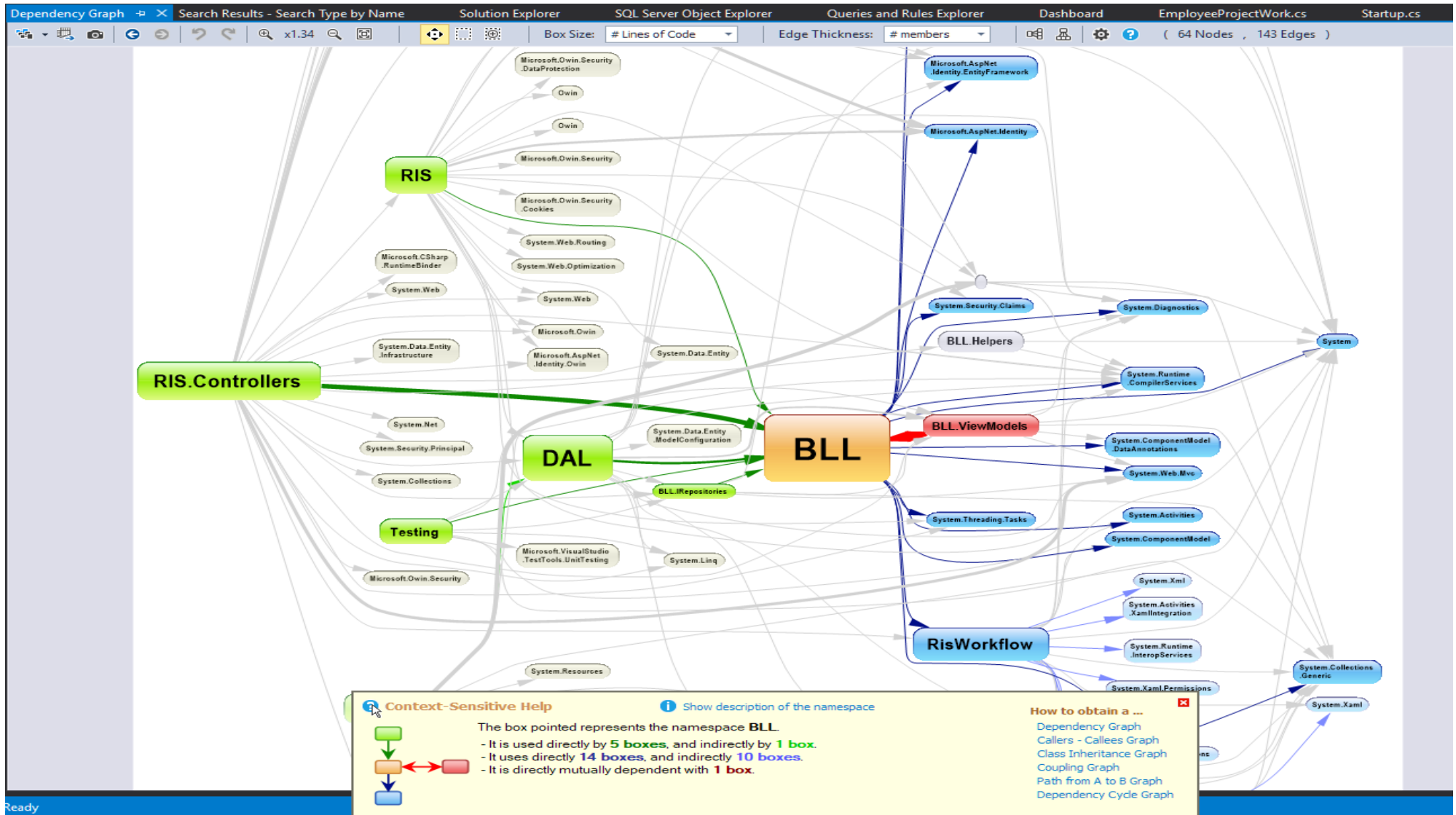
NDepend menu options:

- Dashboard
- Rules
- Issues
- Graph
- Matrix
- Diff
- Trend
- Metrics
- Coverage
- Search
- Project
- Analyze
- Report
- Tools
- Options...
- Windows
- Help

View Explorer Panel sub-menu options:

- View Explorer Panel
- View Editor Panel
- New ...
- 2 Quality Gates Fail
- 0 Quality Gate Warn
- 9 Quality Gates Pass
- 4 Critical Rules Violated
- 29 Rules Violated
- 112 Rules Ok
- More Selections
- Code Smells
- Object Oriented Design
- Design
- Architecture
- Dead Code
- Visibility
- Immutability
- Naming Conventions
- Source Files Organization
- .NET Framework Usage
- Query Options

# Example: application component dependency



---

# Dynamic check

Dynamic Analysis

Fuzzing

Penetration Testing

---

# Fuzzing - "combing" (eng. fuzz = hair)

---

## -DAST (The Good, the Bad and the Ugly)

- fault injection into the application (fuzzing, fuzz testing)
- sending incorrect, unexpected or random data to the program input
- similar to regression, only with bad data
- "combing" applications, protocols, files

## -Advantages:

- simplicity, platform, language independence

## -Disadvantages:

- application to a narrow set of vulnerabilities, eg. *Buffer overflows, Integer overflows,...*
- complex application to technologies (Web 2.0, JSON, Flash, HTML 5.0, Jscript)
- relatively long duration (permutations of incorrect data samples)

## Procedures

---

### -Glupo = Dumb (mutational) fuzzing

- enough less knowledge about the goal and the tools
- pseudorandom anomalies of correct data
- repercussions
  - more analysis needed
  - redundancy of findings

### -Smart (generational) fuzzing

- data generated based on the model
- requires in-depth knowledge of the target and specialized tools
- purposeful anomalies by knowledge of formats, standards, ... (PDF, RFC)
- repercussions
  - less need for analysis
  - less duplication of findings

#### **Standard HTTP GET request**

GET /index.html HTTP/1.1

#### **Anomalous requests**

AAAAAA...AAAA /index.html  
HTTP/1.1

GET //index.html HTTP/1.1

GET %n%n%n%n%n%n.html HTTP/1.1

GET /AAAAAAAAAAAAA.html HTTP/1.1

GET /index.html  
HTTTTTTTTTTTTTTP/1.1

GET /index.html  
HTTP/1.1.1.1.1.1.1.1

# Tools for dynamic analysis

---

-[https://www.owasp.org/index.php/Category:Vulnerability\\_Scanning\\_Tools](https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools)

-CERT Basic Fuzzing Framework (BFF) and Failure Observation Engine (FOE)

-Open source<https://github.com/CERTCC/certifuzz>

-Peach Fuzzer -*automated security testing platform*

-WebScarab - analysis of applications that use HTTP/HTTPS,*intercepting proxy*

-Burp - web applications, buffer overflow, CSS, SQL injection, ...

-Fuddly -*fuzzing and data manipulation framework (for GNU/Linux)*

-Hongfuzz -*general fuzzer*

-*Profilers, ...*

# Penetration testing (Pen Test), ethical hacking

---

- assessment of system or network security by simulating a malicious attack
- person, team, preferably external consultants
- written permission of the owner (implementation of illegal activities)

## -Purpose

- Confirmation of the functionality of security controls
- Timely detection of security flaws
- Prevention of security incidents
- Investment justification
- Compliance with regulatory requirements



# Approach to penetration testing

---

- Types of verification according to the availability of information
  - without available information (eng.*black-box test*) - as a real attack
  - with all information (eng.*white-box test*) - the worst case, when the attacker knows everything, or the simulation of an attack by an internal attacker
  - with partially available information (eng.*gray-box test*) - hybrid
- Test starting point criterion
  - External - from a remote location (Internet) to publicly available systems
  - Internal - from the intranet, simulation of an incident of unauthorized access to the internal network infrastructure
- Other criteria
  - Range, stealth, techniques, aggressiveness

# Performing a penetration test

---

- **Research**(Eng.*reconnaissance*), scouting
  - the examiner tries to collect as much information as possible.
  - passive - publicly available information (eg data from social networks, Google)
  - active - research tools (e.g.*nslookup*), to determine certain parameters
- **Scanning**(Eng.*scanning*)
  - the tester scans open ports (*port scanning*) using tools (e.g. Nmap)
  - goal - enumeration of services, versions of enumerated services and OS (*OS and service fingerprinting*).
  - vulnerability scan (*vulnerability scanning*), automated tools (e.g. OpenVAS)
- **Gaining access**(Eng.*obtaining access*)
  - exploitation of vulnerabilities, either manually or with a tool (e.g. Metasploit),
  - depending on the agreement with the owner, some vulnerabilities will not be exploited (eg server crash)
- **Access retention**(Eng.*maintaining access*)
  - examiner installs malicious*backdoorandrootkit*programs for further access to the system
  - this and the next phase are not usually carried out in practice, but represent a scenario of a real attack
- **Erasing traces**(Eng.*erasing records*)
  - the examiner attempts to delete log entries that would indicate their unauthorized access

Tools	Basic functionality	Phase
Harvester	Researching publicly available information using search engines, social networks, etc.	Research
Nmap	Network research and port scanning.	Scanning
QualysGuard (Network)	Network scan for known vulnerabilities.	Scanning
QualysGuard (WAS)	Vulnerability scanning of web applications.	Scanning
ASPAuditor	Identification of vulnerable and poorly configured ASP.NET servers.	Scanning
Nobody	Scanning web servers for various vulnerabilities, such as dangerous ones files etc.	Scanning
REC	A multifunctional tool for penetration testing of web applications.	Scanning
Sqlmap	SQL injection vulnerability detection and exploitation.	Conducting an attack
Metasploit	A multifunctional platform for exploiting vulnerabilities.	Conducting an attack
HTTP DoS	Denial of service at the application layer.	Conducting an attack
Hydra	Remote password cracking.	Conducting an attack
Wireshark	Recording and analysis of network traffic.	Conducting an attack

## Example: Nmap and QualysGuard

```
root@bt:~# nmap -sS -sV 22.22.22.22 Host is up
(0.0027s latency).
```

Not shown: 992 filtered ports PORT

	STATE	SERVICE	VERSION
21/tcp	open	ftp	Microsoft ftp
25/tcp	open	smtp?	
80/tcp	open	http	Microsoft IIS httpd 6.0
110/tcp	open	pop3	Microsoft Windows 2003 POP3 Service1.0 Microsoft
443/tcp	open	ssl/http	IIS httpd 6.0
3389/tcp	open	microsoft-rdp	Microsoft Terminal Service

### Summary of Vulnerabilities

Vulnerabilities Total	24	Security Risk (Avg)	 5.0
-----------------------	----	---------------------	---

#### by Severity

Severity	Confirmed	Potential
5	1	0
4	0	0
3	0	0
2	1	0
1	0	0
Total	2	0



5

Microsoft Windows Remote Desktop Protocol Remote Code Execution

QID: 90783

Category: Windows

CVE ID: [CVE-2012-0002](#), [CVE-2012-0152](#)

Vendor Reference: [MS12-020](#)

Bugtraq ID: -

Service Modified: 03/29/2012

User Modified: -

#### 5 Biggest Categories

Category	Confirmed	Potential
Information gathering	0	0
TCP/IP	0	0
Windows	2	0
Web server	0	0
CGI	0	0
Total	2	0

## Example: Hydra, Wireshark

### -Dictionary attack with the Hydra tool

[STATUS] 446.66 tries/min, 187152 tries in 06:59h, 0 todo in 00:01h [STATUS] attack finished for 22.22.22.22 (waiting for children to finish) 1 of 1 target successfully completed, 0 valid passwords found

Hydra (<http://www.thc.org/thc-hydra>) finished at 2012-06-10 17:20:30

### -Authentication data during the attack captured by the Wireshark tool

23992	531.375	192.168.235.128	██████████	FTP	72	Request: PASS zagreb0702!
23993	531.375	██████████	192.168.235.128	TCP	60	ftp > 48027 [ACK] Seq=83068 Ack=3674
23994	531.481	██████████	192.168.235.128	FTP	87	Response: 331 Password required for
23995	531.481	192.168.235.128	██████████	FTP	72	Request: PASS zagreb0703!
23996	531.481	██████████	192.168.235.128	TCP	60	ftp > 48028 [ACK] Seq=85051 Ack=3762

File Transfer Protocol (FTP)

PASS zagreb0703!\r\n

Request command: PASS

Request arg: zagreb0703!

## Example: Metasploit

```
msf > use auxiliary/dos/windows/rdp/ms12_020_maxchannelids msf
```

```
auxiliary(ms12_020_maxchannelids) > show options Module
```

options (auxiliary/dos/windows/rdp/ms12\_020\_maxchannelids): Current

To us	Setting	Required	Description
-----	-----	-----	-----
RHOST		yes	The target address
REPORT	3389	yes	The target port

```
msf auxiliary(ms12_020_maxchannelids) > set RHOST 11.11.11.11 RHOST => 11.11.11.11
```

```
msf auxiliary(ms12_020_maxchannelids) > exploit
```

```
[*] 11.11.11.11:3389 - Sending MS12-020 Microsoft Remote Desktop Use-After-Free DoS
```

```
[*] 11.11.11.11:3389 - 210 bytes sent [*]
```

```
11.11.11.11:3389 - Checking RDP with [+]
```

```
11.11.11.11:3389 seems down
```

```
A problem has been detected and windows has been shut down to prevent damage to your computer.
```

```
If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:
```

```
Check to be sure you have adequate disk space. If a driver is identified in the Stop message, disable the driver or check with the manufacturer for driver updates. Try changing video adapters.
```

```
Check with your hardware vendor for any BIOS updates. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.
```

```
Technical information:
```

```
*** STOP: 0x0000008E (0xC0000005, 0x8DA6F987, 0x931778F0, 0x00000000)
```

```
*** termdd.sys - Address 8DA6F987 base at 8DA6E000, DateStamp 4ce7a116
```

```
Collecting data for crash dump ...  
initializing disk for crash dump ...
```

# Tools for penetration testing and intrusion detection

---

## -Pentest

-[https://www.owasp.org/index.php/Category:Penetration\\_Testing\\_Tools](https://www.owasp.org/index.php/Category:Penetration_Testing_Tools)

- Aircrack-ng - WIFI scanner - open source
- Burp Suite - web vulnerability scanner
- Cain & Abel - "password recovery tool" - packet sniffer, password cracker, ...
- Ettercap - suite for man in the middle attacks - open source
- John The Ripper - password cracker
- Nessus - vulnerability scanner - free trial
- Kismet - network detector, packet sniffer, IDS - freeware
- Zed Attack Proxy (**REC**) - web application security scanner - Apache 2 License

## -ID/PS

-<https://www.comparitech.com/net-admin/network-intrusion-detection-tools/>

- Snort**, OSSEC, ..., Solarwinds Log and Event Manager, ...

# Example: Snort

Services / Snort / Alerts

Snort Interfaces

Global Settings

Updates

Alerts

Blocked

Pass Lists

Suppress

IP Lists

SID Mgmt

Log Mgmt

Sync

Clear all interface log files

Alert Log View Settings

Interface to Inspect

WAN

Choose interface..

☐ Auto-refresh view

1000

Alert lines to display.

Save

Alert Log Actions

Download

Clear

Alert Log View Filter

Last 1000 Alert Log Entries

Date	Pri	Proto	Class	Source IP	SPort	Destination IP	DPort	SID	Description
2017-07-23 20:49:52	1	UDP	A Network Trojan was Detected	66.240.205.34	1066		16464	1:31136	MALWARE-CNC Win.Trojan.ZeroAccess inbound connection
2017-07-22 06:15:49	2	UDP	Potentially Bad Traffic	163.172.17.76	54465		5060	140:26	(spp_sip) Method is unknown
2017-07-21 09:26:30	2	UDP	Potentially Bad Traffic	163.172.22.169	52428		5060	140:26	(spp_sip) Method is unknown
2017-07-21 01:03:28	2	UDP	Potentially Bad Traffic	163.172.17.76	46834		5060	140:26	(spp_sip) Method is unknown
2017-07-20 20:36:37	2	UDP	Potentially Bad Traffic	163.172.22.169	54788		5060	140:26	(spp_sip) Method is unknown
2017-07-20 08:31:30	2	UDP	Potentially Bad Traffic	163.172.17.76	59571		5060	140:26	(spp_sip) Method is unknown



## References

---

- [OWASP Category: Vulnerability Scanning Tools](#)
- [OWASP Code Review Guide](#)
- [OWASP Testing Guide](#)

### -More tools...

- <http://sectools.org/>
- [https://www.owasp.org/index.php/Appendix\\_A:\\_Testing\\_Tools](https://www.owasp.org/index.php/Appendix_A:_Testing_Tools)
- [Software Security Assessment Tools Review, 2009](#)