Security Testing

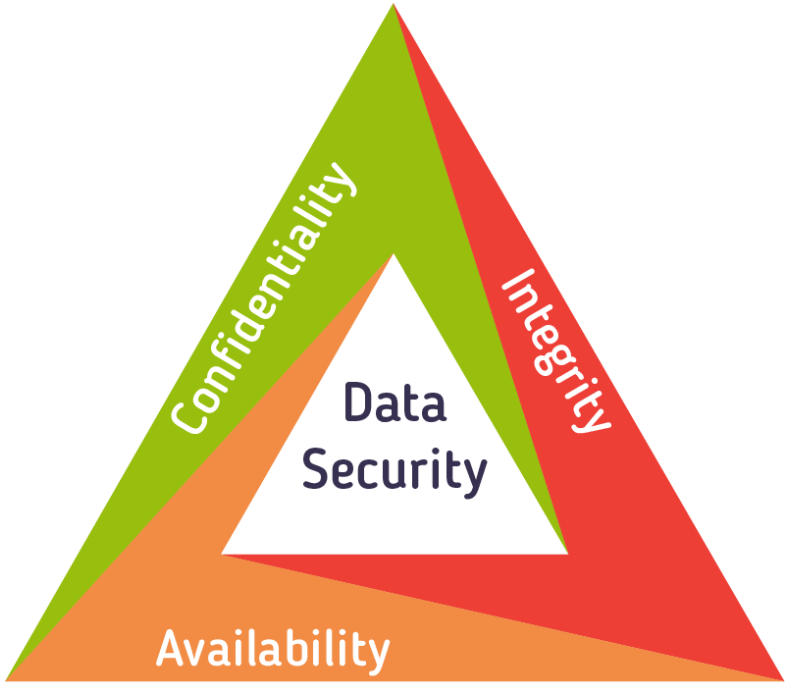
Azqa Nadeem
PhD candidate @ Cyber Analytics Lab
Department of Intelligent Systems
Delft University of Technology
azqa.nadeem@tudelft.nl

Software testing
vs.
Security testing

Security testing

“Intended to reveal weaknesses in the security principles of a software”

Assumption of an “active adversary” with an intent to inflict harm



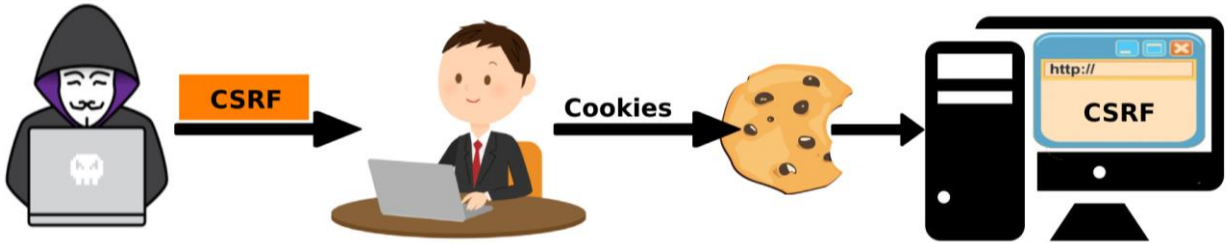
https://devopedia.org/information-security-principles

“Bug” as a security vulnerability

Heartbleed vulnerability

https://nl.wikipedia.org/wiki/Heartbleed

“Feature” as a security vulnerability



Cross-site Request Forgery (CSRF)

https://securitysouls.com/a-tale-of-cross-site-request-forgery-csrf/

Security vulnerabilities

- May be a software bug

- May be non-functional

- May exploit non-buggy code
- May not be directly present in *your* codebase

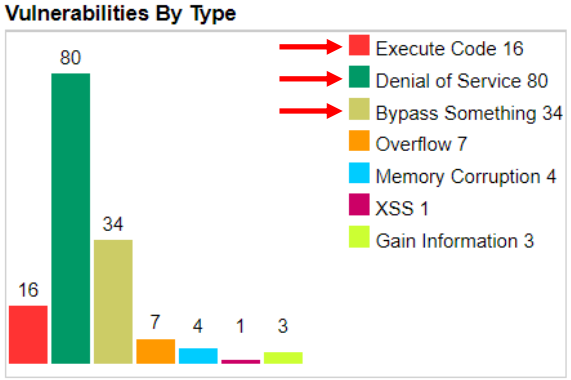
- May hit time-tested code!

https://devopedia.org/information-security-principles

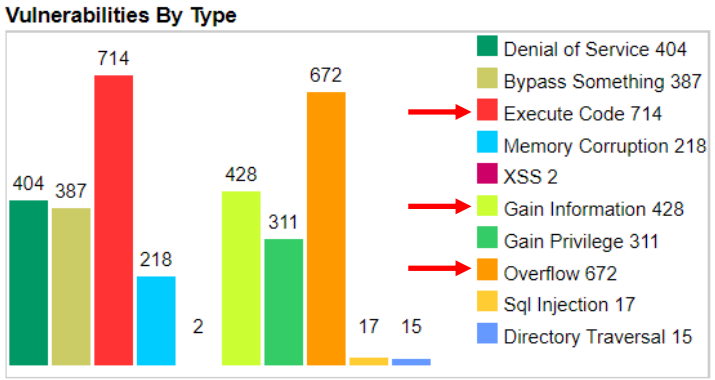
… Java is safe, right?

Understanding Java vulnerabilities

- Java Virtual Machine → Native C code
  - Alternate ways to achieve memory corruption



JRE



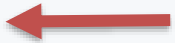
Android

https://www.cvedetails.com/product/19117/Oracle-JRE.html?vendor_id=93
https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224

Some Examples

What's wrong?

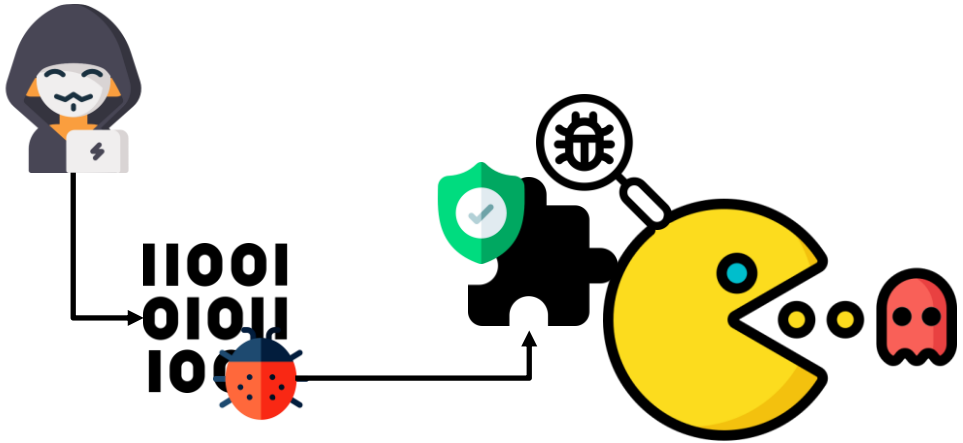
```
Socket socket = null;
BufferedReader readerBuffered = null;
InputStreamReader readerInputStream = null;

/*Read data using an outbound tcp connection */
socket = new Socket("host.example.org", 39544);

/* Read input from socket */
readerInputStream = new InputStreamReader(socket.getInputStream(), "UTF-8");
readerBuffered = new BufferedReader(readerInputStream);

/* Read data using an outbound tcp connection */
String data = readerBuffered.readLine();

Class<?> tempClass = Class.forName(data);
Object tempClassObject = tempClass.newInstance();

IO.writeLine(tempClassObject.toString());

// Use tempClass in some way
```



TU Delft

Code Injection vulnerability [1/2]

- Execute code in unauthorized applications
- Update Attack *(via Dynamic Class Loading)*



Code Injection vulnerability [1/2]

- Execute code in unauthorized applications
- Update Attack *(via Dynamic Class Loading)*
- Tricky to fix
  - Disallow untrusted plugins
  - Disallow remote calls to untrusted servers
  - Limit access rights

Remote Code Execution [2/2]

- Execute arbitrary code on a remote device
- Achieved via
  - Out-of-bounds writes
  - Injection attacks
  - Deserialization attacks

https://www.cvedetails.com/cve/CVE-2021-44228/

Remote Code Execution [2/2]


```
final Logger logger = LogManager.getLogger(...);

// input = "Hello world"
logger.error("Search query: {}", input)
```

https://www.cvedetails.com/cve/CVE-2021-44228/
https://youtu.be/uyq8yxWO1ls

Remote Code Execution [2/2]

- Execute arbitrary code on a remote device
- Achieved via
  - Injection attacks
  - Deserialization attacks
  - Out-of-bounds writes
- Fixes (Log4J)
  - Set `trustURLCodebase` flags to `False`
  - Update to latest version
  - Patch class directly

TU Delft

https://www.cvedetails.com/cve/CVE-2021-44228/

Oracle April 2018 CPU: Most Java flaws can be remotely exploited

By News | April 18, 2018 | Alerts

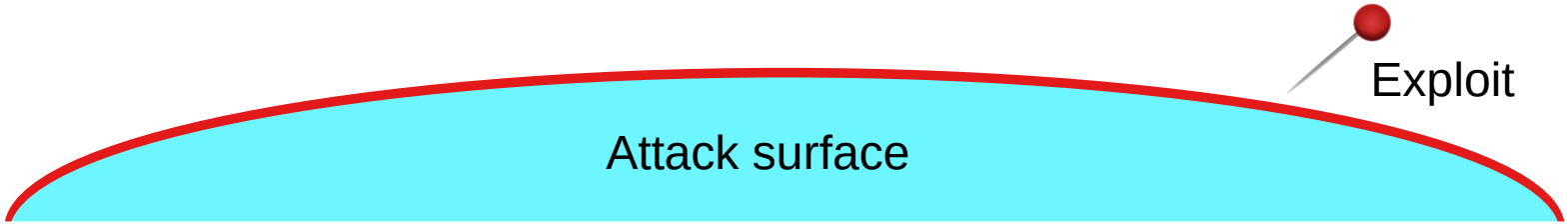
Half of the Java patches relate to Deserialization Flaws.

Customer Alert 20180418

Oracle Critical Patch Update April 2018 Released

https://www.waratek.com/alert-oracle-guidance-cpu-april-2018/

The state of security



Security testing goals:

- Limit exposure
- Increase exploitation difficulty

Who's job is to test for security?

When to test for security?

Secure implementation,
White-box testing

Code reviews, Black-box testing
& Vulnerability scanning

"With great power comes great responsibility"

- Peter Parker principle

Threat modelling
& Abuse cases

Security requirements,
Risk assessment

https://www.dignitasdigital.com/blog/easy-way-to-understand-sdlc/
https://www.javatpoint.com/security-testing

Quality Assessment Criteria

- Soundness
  - No missed vulnerability (0 FNs)
  - No alarm → no vulnerability exists
- Completeness
  - No false alarms (0 FPs)
  - Raises an alarm → vulnerability found

	No alarm	Alarm!
Safe	TN	FP
Unsafe!	FN	TP

Quality Assessment Criteria

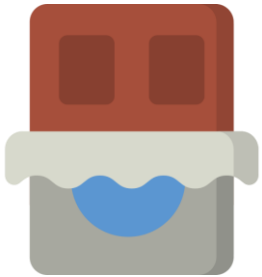
- Soundness
  - No missed vulnerability (0 FNs)
  - No alarm → no vulnerability exists
- Completeness
  - No false alarms (0 FPs)
  - Raises an alarm → vulnerability found

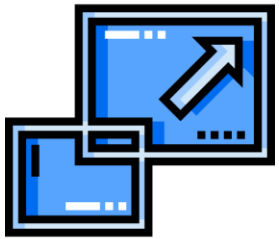
- Ideally: ↑Soundness + ↑Completeness
- Reality: Compromise on FPs or FNs

TU Delft

Usable Security Testing Tools

- ↓ FPs vs. ↓ FNs
- ↑ Interpretability
- ↑ Generalizability





Facets of Security Testing

- Static vs. dynamic testing?
- White-box vs. Black-box?

	White-box	Black-box
Static Application Security Testing	Code checking, Pattern matching, ASTs, CFGs, DFDs	
Dynamic Application Security Testing	Tainting, Dynamic validation, Symbolic execution	Penetration testing, Reverse engineering, Behavioral analysis, Fuzzing

Static Application Security Testing

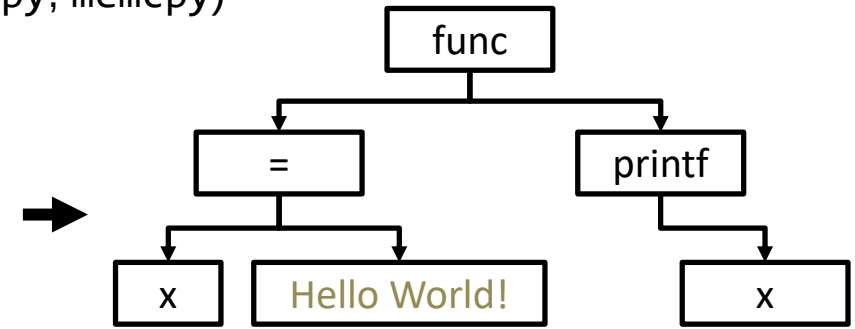
1. Code analysis
2. Structural analysis

Code analysis

- Looks for pre-defined patterns in a codebase

- Regular expressions
  - Misconfigurations (`port 22`)
  - Bad imports (`System.io.*`)
  - Call to dangerous functions (`strcpy`, `memcpy`)
- Abstract Syntax Trees
  - Format string attack

```
x = "Hello World!";
printf(x);
```



Error: Missing param!

Structural analysis

- Looks at control and data flows of a codebase

- Control Flow Graphs
  - Access rights violations (Privilege escalation)
  - Code-reuse detection (malware variant detection)

- Data Flow Analysis
  - (Simple) code injection (Sanitization issues)
  - Use-after-free vulnerability (Memory corruption)

Sun, Xin, et al. "Detecting code reuse in android applications using component-based control flow graph." IFIP international information security conference. Springer, Berlin, Heidelberg, 2014.

Static Application Security Testing

1. Code analysis
   - Regular expressions
   - Abstract Syntax Trees

2. Structural analysis
   - Control flow graphs
   - Data flow graphs

Signature-based!
Denial of Service?? Crashes??

Dynamic Application Security Testing

1. Taint analysis
2. Reverse engineering
3. Fuzzing
4. Penetration testing

Taint analysis

- Tracks data in memory and its propagation in an application

- Dynamic version of Data Flow Analysis
- Contact tracing for data

- Code instrumentation (white-box!)

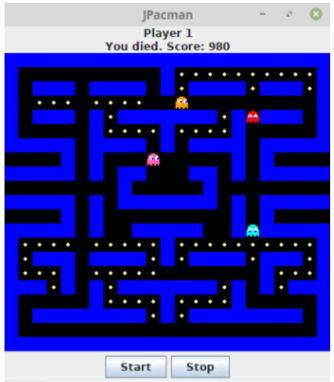
- Panorama: Looks for hooks into OS functions

Yin, Heng, et al. "Panorama: capturing system-wide information flow for malware detection and analysis." Proceedings of the 14th ACM conference on Computer and communications security. 2007

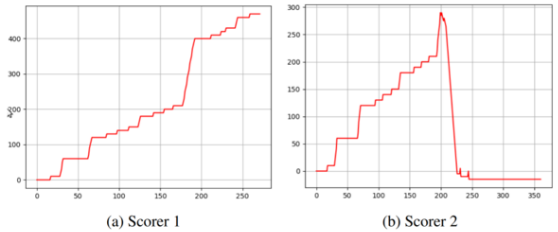
Reverse engineering

- Attempts to reveal the internal structure of an application

- Black-box → white-box
- Behavioral analysis via input/output mapping
  - Modeling application behavior
  - Forensic log analysis



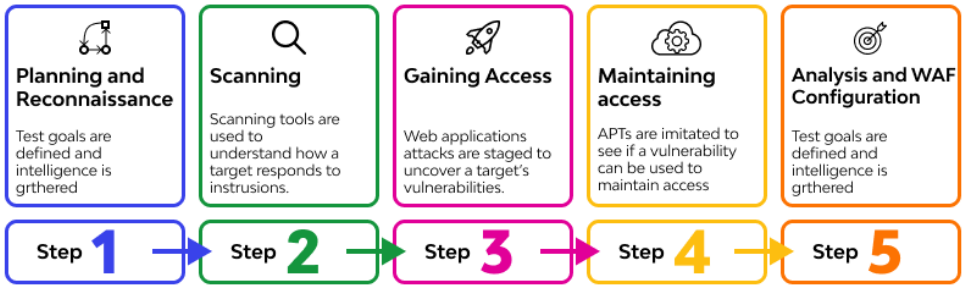
Malicious Point Calculator



Penetration testing

- Attempts to breach the application security like an adversary would
  - *a.k.a Ethical hacking*

- Most popular testing technique!
- Generally black-box
- Penetration Testing Execution Standard



https://www.wallarm.com/what/what-is-penetration-testing

Fuzzing

- Discovers vulnerabilities by providing garbage inputs to an application

- Fully black-box!
- Discovers zero-day exploits

- Stagefright: overflow in MMS module
  - Remote code execution
  - Privilege escalation

- No crash, no detection!

https://en.wikipedia.org/wiki/Stagefright_(bug)

Dynamic Application Security Testing

1. Taint analysis
2. Reverse engineering
3. Fuzzing
4. Penetration testing

(Mostly) signature-free!
Typically slower than SAST

Summary

- Security testing assumes adversaries
- Secure-SDLC incorporates security at each phase
  - Perfect security testing is impossible
- Code analysis finds limited, but easy-to-find vulnerabilities
- Fuzzing helpful for finding zero-days without human intervention
- Reverse engineering useful for forensic analysis
- SAST might be fast, but DAST is a better choice for security testing in general!

That’s all for today!

SAST vs. DAST Performance

SAST

- ↑ False positives
- ↑ False negatives
- ↑ White-box
- ↑ Speed
- ↓ Generalizability

DAST

- ↓ False positives
- ↓ False negatives
- ↓ ↑ White-box
- ↓ Speed
- ↑ Generalizability