UNIVERSITY OF TWENTE.

M2: Software Security (Lecture 2)

Lecturer: Azqa Nadeem

a.nadeem@utwente.nl

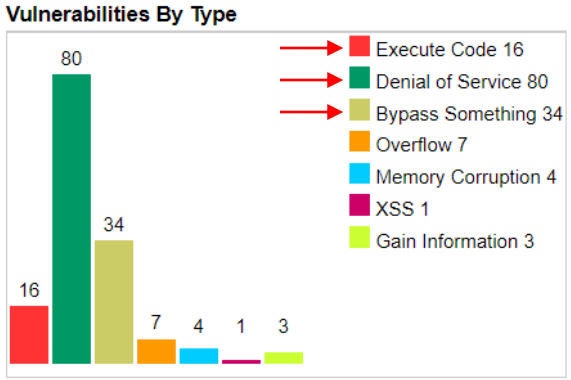
Security vulnerabilities

- 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!

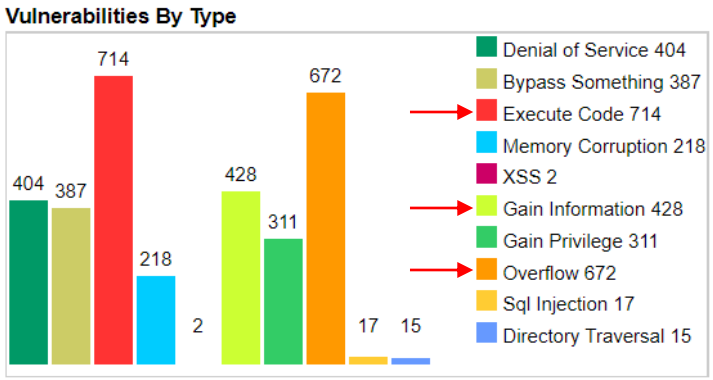
https://devopedia.org/information-security-principles

Java & Security

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



JRE



Android

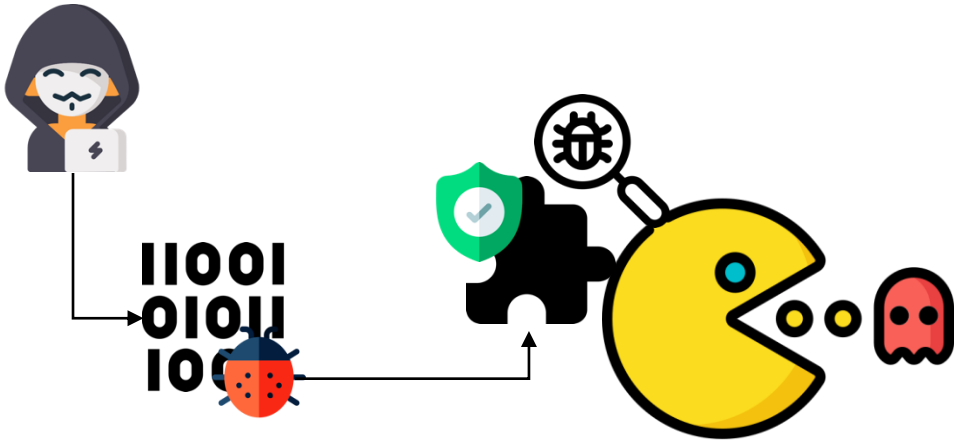
https://www.cvedetails.com/product/19117/Oracle-JRE.html?vendor_id=93
https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224

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
```

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

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

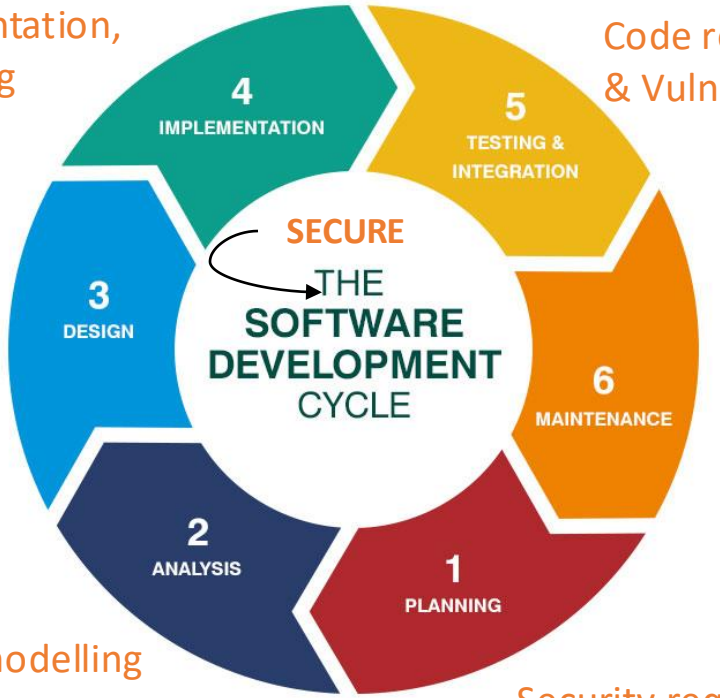
Oracle Critical Patch Update April 2018 Released

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

Secure SDLC

Secure implementation,
White-box testing

Code reviews, Black-box testing
& Vulnerability scanning



Design for security,
Testing plan

Impact analysis,
Patching & Updating

Threat modelling
& Abuse cases

Security requirements,
Risk assessment

UNIVERSITY
OF TWENTE.

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

Text encodings

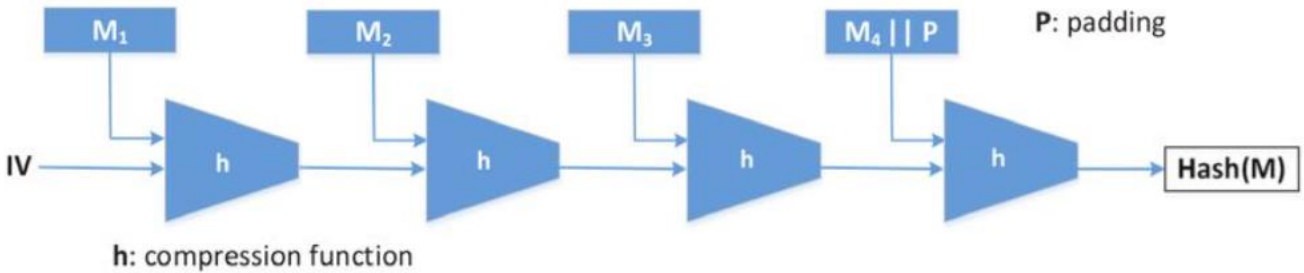
- Why do we need encodings?
  - Printable characters, standardized, portable

- How to represent ASCII/UTF-8 character ‘Z’
- Hex (base 16): $5A_{16}$ ← 1 byte
- Binary (base 2): $0101\ 1010_2$
- Decimal (base 10): 90_{10}
- Octal (base 8): 132_8

- How to represent ASCII/UTF-8 ‘Zwolle’
- Hex (base 16): 5A 77 6F 6C 6C 65 ← 12 chars
- UTF (base 64): WndvbGxl ← 8 chars

How hashing works

- Scramble data in one direction for fixed-sized output
- M → H(M) will always return the same hash



https://slideplayer.com/slide/17453038/

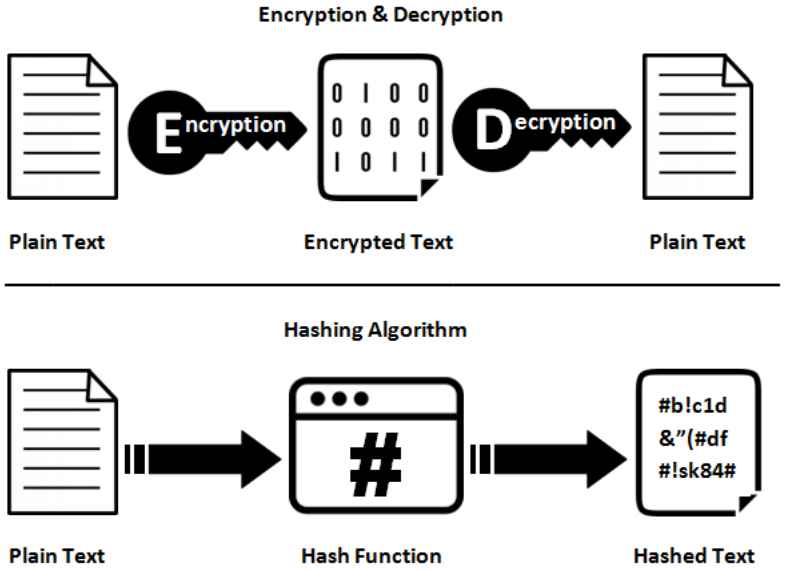
Properties of cryptographic hash functions

- One-way function
- Second preimage resistant
- Collision resistant

- How to represent ASCII/UTF-8 ‘Zwolle’
  - MD5 (Always 128-bit): 937c513dd1f5b216e7f632f9c3bdc1a5
  - SHA1 (Always 256-bit): ad7d96898adfab2b9dc6c4751ac2d5d649020fb9

- Used for
  - Integrity checks
  - Password storage

Hashing vs encryption



The problem with typical hashes

- Dictionary attacks/Rainbow table attacks
  - Pre-hash most common passwords and match leaked hashes
  - Example: https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-1000000.txt

https://medium.com/@berto168/salt-pepper-spice-up-your-hash-b48328caa2af
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Modern password storage methods

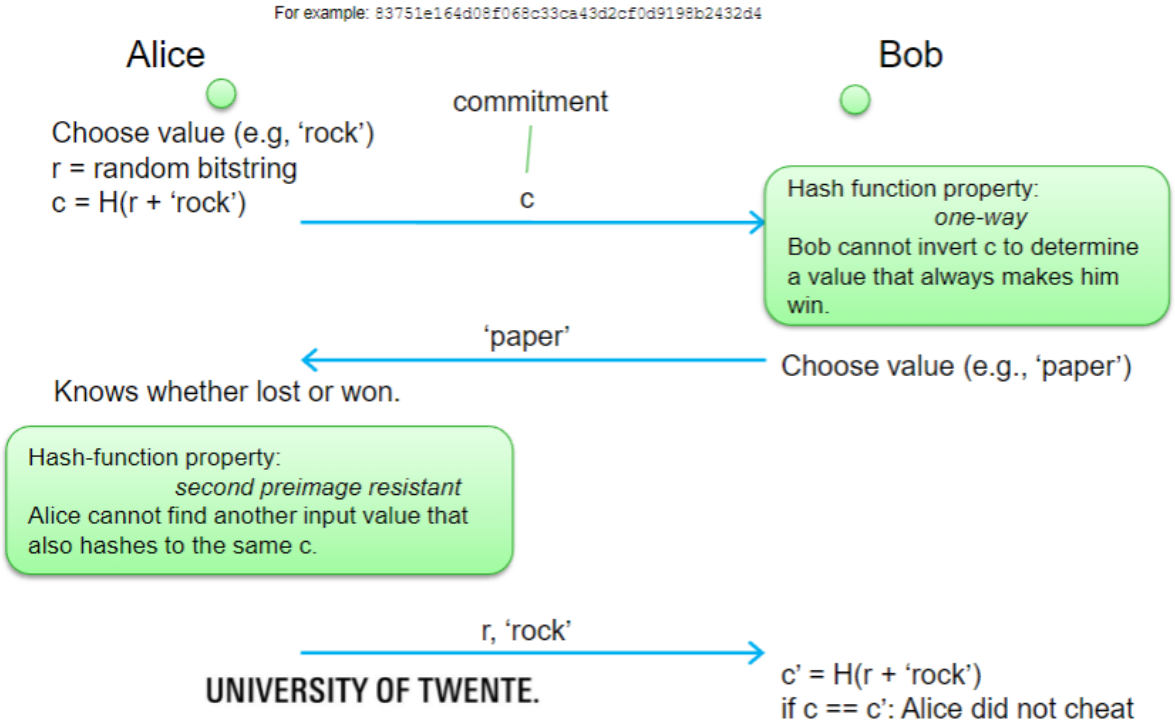
- Seasoning your hashes
  - Adding a random value to passwords before being hashed (server-side)
  - Unique hash each time
  - Computationally infeasible for attackers to match (salt x password) combos
  - Salt stored in database (plaintext is okay) | Pepper stored away from it

- Adding work factor: repeatedly hash the password

- Modern algorithms are fast enough for normal use but slow for brute-force
  - e.g., PBKDF2, bcrypt, and scrypt

https://medium.com/@berto168/salt-pepper-spice-up-your-hash-b48328caa2af
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Commitments (via rock, paper, scissors)



Summary

- Java is memory-safe but security vulnerabilities still possible
  - Safety ≠ security
- Security activities added in each step of the SDLC → Secure-SDLC
- Encodings for transmitting data | Hashing for garbling data
- Hashing for password storage and authentication
  - May be broken. Use salt/pepper/work factor