UNIVERSITÄT PASSAU

Fakultät für Informatik und Mathematik

6090: Security of Computer and Embedded Systems

Week 4: CVSS; Software Vulnerabilities & Secure Programming

Elif Bilge Kavun

elif.kavun@uni-passau.de

November 11, 2021

This Week's Outline

- Common Vulnerability Scoring System (CVSS)
  - To understand how to assess vulnerabilities
- Software Vulnerabilities & Secure Programming
  - Pointing common vulnerabilities and how to securely code not to have them

The Common Vulnerability Scoring System (CVSS)

How to Discuss and Assess Software Security Issues?

- We need
  - A common method for risk assessment
    - Common Vulnerability Scoring System (CVSS): https://www.first.org/cvss/specification-document
  - A clearly-defined language that allows us to name different security issues, i.e., well-defined concepts/names
    - OWASP Top Ten: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
    - Common Weakness Enumeration (CWE): https://cwe.mitre.org/index.html
  - A system for referencing known vulnerabilities
    - Vendor-independent, e.g.,
      - Common Vulnerabilities and Exposures (CVE) Database: https://cve.mitre.org/
      - The U.S. National Vulnerability Database (NVD): https://nvd.nist.gov/
      - HPI VulnDB: https://sec.hpi.de/vulndb/
    - Vendor-specific, e.g.,
      - Microsoft Security Bulletins and Advisories: https://docs.microsoft.com/en-us/security-updates/
      - SAP Security Notes (only available to customers): https://support.sap.com/securitynotes

CVSS: Common Vulnerability Scoring System

- Industry standard for rating the severity of software vulnerabilities
  - Version 3 is the most current one, slowly replacing Version 2
  - A “wish-list” is being collected for future Version 4
  - In the lecture, we will start with Version 2 and then move on to Version 3
  - Measures three areas of concern
    - ***Base Metrics*** for qualities intrinsic to a vulnerability
    - ***Temporal Metrics*** for characteristics that evolve over the lifetime of vulnerability
    - ***Environmental Metrics*** for vulnerabilities that depend on a particular implementation or environment
  - A numerical score is generated for each of these metric groups (often published as vector containing all three values)

CVSS: Common Vulnerability Scoring System (Version 2)

Computing the CVSS Base Metrics (Version 2): Base Metrics

- Attack Vector (AV): Shows how a vulnerability may be exploited

Value	Description	Score
Local (L)	The attacker must have physical access or a local account	0.395
Adjacent Network (A)	The attacker must have access to a neighboring network	0.646
Network (N)	The attacker only needs remote access	1.000

- Access Complexity (AC): How easy/difficult it is to exploit the discovered vulnerability

Value	Description	Score
High (H)	Specialized conditions must be fulfilled (e.g., race conditions)	0.350
Medium (M)	Some additional requirements must be fulfilled (e.g., non-default configuration)	0.610
Low (L)	No special conditions	0.710

- Authentication (Au): How often needs an attacker to authenticate to a target to exploit it

Value	Description	Score
Multiple (M)	Several authentications required	0.450
Single (S)	Single authentication required	0.560
None (N)	Unauthenticated access	0.704

CVSS: Common Vulnerability Scoring System (Version 2)

Computing the CVSS Base Metrics (Version 2): Impact Metrics

- Confidentiality (C): Impact on the confidentiality of the processed data

Value	Description	Score
None (N)	No impact on the confidentiality	0.000
Partial (P)	Considerable information disclosure (but constrained)	0.275
Complete (C)	Total information disclosure (all data/information)	0.660

- Integrity (I): Impact on the integrity of the system

Value	Description	Score
None (N)	No impact on the integrity	0.000
Partial (P)	Modification of some data (but limited)	0.275
Complete (C)	Total loss of integrity	0.660

- Availability (A): Impact on the availability of the system

Value	Description	Score
None (N)	No impact on the availability	0.000
Partial (P)	Reduced performance or some loss of functionality	0.275
Complete (C)	Total loss of availability	0.660

CVSS: Common Vulnerability Scoring System (Version 2)

Computing the CVSS Base Metrics (Version 2)

- These six metrics are used to calculate the *BaseScore*

$$Exploitability = 20 \times AttackVector \times AccessComplexity \times Authentication$$

$$Impact = 10.41 \times (1 - (1 - ConfImpact) \times (1 - IntegImpact) \times (1 - AvailImpact))$$

$$f(Impact) = \begin{cases} 0.000, \text{ if } Impact = 0 \\ 1.176, \text{ otherwise} \end{cases}$$

$$BaseScore = ((0.6 \times Impact) + (0.4 \times Exploitability) - 1.5) \times f(Impact)$$

- The *BaseScore* is rounded to one decimal

- CVSS 2.0 calculator: https://nvd.nist.gov/CVSS/v2-calculator

CVSS: Common Vulnerability Scoring System (Version 2)

Computing the CVSS Base Metrics (Version 2): Example

- ***Example:*** A buffer overflow vulnerability affects web server software that allows a remote user to gain partial control of the system, including the ability to cause it to shut down

CVSS: Common Vulnerability Scoring System (Version 2)

Computing the CVSS Base Metrics (Version 2): Example

- ***Example:*** A buffer overflow vulnerability affects web server software that allows a remote user to gain partial control of the system, including the ability to cause it to shut down

Metric	Value	Description
Attack Vector	Network	Access from any network (Internet) possible
Access Complexity	Low	No special requirements for access
Authentication	None	No authentication necessary
Confidentiality	Partial	Only partial control (e.g., reading files)
Integrity	Partial	Only partial access (e.g., modifying files)
Availability	Complete	Attacker can cause a shutdown

CVSS: Common Vulnerability Scoring System (Version 2)

Computing the CVSS Base Metrics (Version 2): Example

- ***Example:*** A buffer overflow vulnerability affects web server software that allows a remote user to gain partial control of the system, including the ability to cause it to shut down

Metric	Value	Description
Attack Vector	Network	Access from any network (Internet) possible
Access Complexity	Low	No special requirements for access
Authentication	None	No authentication necessary
Confidentiality	Partial	Only partial control (e.g., reading files)
Integrity	Partial	Only partial access (e.g., modifying files)
Availability	Complete	Attacker can cause a shutdown

- Exploitability: 10 and Impact: 8.5
- *BaseScore*: 9.0
- CVSS Vector: (AV:N/AC:L/Au:N/C:P/I:P/A:C)

CVSS: Common Vulnerability Scoring System

An Example and A Warning (1/2)

- Consider the following vulnerability in OpenSSL

The TLS and DTLS implementations in OpenSSL do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read.

A successful attack requires only sending a specially crafted message to a web server running OpenSSL. The attacker constructs a malformed "heartbeat request" with a large field length and small payload size. The vulnerable server does not validate that the length of the payload against the provided field length and will return up to 64 kB of server memory to the attacker. It is likely that this memory was previously utilised by OpenSSL. Data returned may contain sensitive information such as encryption keys or user names and passwords that could be used by the attacker to launch further attacks.

- What CVSS v2 Base Vector (AV:?/AC:?/Au:?/C:?/I:?/A:?) would you assign to this vulnerability?

CVSS: Common Vulnerability Scoring System

An Example and A Warning (2/2)

- While lower CVSS scores represent lower risk, they are not the “absolute truth”!

- Consider Heartbleed (CVE-2014-0160)
  - One of the most well-known vulnerabilities of the last decade
  - Full disclosure of server key possible
  - CVSS: 5.0/10
  - Most likely the only CVE with a CVSS warning/explanation:

“CVSS v2 scoring evaluates the impact of the vulnerability on the host where the vulnerability is located. When evaluating the impact of this vulnerability to your organization, take into account the nature of the data that is being protected and act according to your organization’s risk acceptance. While CVE-2014-0160 does not allow unrestricted access to memory on the targeted host, a successful exploit does leak information from memory locations which have the potential to contain particularly sensitive information, e.g., cryptographic keys and passwords. Theft of this information could enable other attacks on the information system, the impact of which would depend on the sensitivity of the data and functions of that system.”

CVSS: Common Vulnerability Scoring System (Version 3)

- Version 3 tries to provide a more fine-grained assessment
  - Introduces new metrics such as Scope (S) and User Interaction (UI)
  - Updates old metrics such as Authentication (Au) to newer ones, e.g., Privileges Required (PR)
- CVSS v3 Base Vector
  - Exploitability Metrics
    - Attack Vector (AV): Network, Adjacent, Local, Physical
    - Attack Complexity (AC): Low, High
    - Privileges Required (PR): None, Low, High
    - User Interaction (UI): None, Required
  - Scope (S): Unchanged, Changed
  - Impact Metrics
    - Confidentiality Impact: High, Low, None
    - Integrity Impact: High, Low, None
    - Availability Impact: High, Low, None
- CVSS 3.0 calculator: https://www.first.org/cvss/calculator/3.0

CVSS: Example 1

- Consider the following vulnerability in the database system MySQL (CVE-2013-0375)

A vulnerability in the MySQL Server database could allow a remote, authenticated user to inject SQL code that MySQL replication functionality would run with high privileges. A successful attack could allow any data in a remote MySQL database to be read or modified.

- What CVSS v3 Base Vector (AV:[N,A,L,P]/AC:[L,H]/PR:[N,L,H]/UI:[N,R]/S:[U,C]/C:[H,L,N]/I:[H,L,N]/A:[H,L,N]) would you assign to this vulnerability?

CVSS: Example 2

- Consider the following vulnerability in Apache Tomcat (CVE-2009-0783)

Apache Tomcat 4.1.0 through 4.1.39, 5.5.0 through 5.5.27, and 6.0.0 through 6.0.18 permits web applications to replace an XML parser used for other web applications, which allows local users to read or modify the (1) web.xml, (2) context.xml, or (3) tld files of arbitrary web applications via a crafted application that is loaded earlier than the target application.

- What CVSS v3 Base Vector (AV:[N,A,L,P]/AC:[L,H]/PR:[N,L,H]/UI:[N,R]/S:[U,C]/C:[H,L,N]/I:[H,L,N]/A:[H,L,N]) would you assign to this vulnerability?

CVSS: Example 3

- Consider the following vulnerability in DocuWiki (CVE-2014-9253)

DokuWiki contains a reflected cross-site scripting (XSS) vulnerability. This vulnerability allows an attacker with privileges to upload a malicious SWF file to a vulnerable site to perform XSS attacks against victims who follow crafted links to those malicious SWF files. Victims following those crafted links would execute arbitrary script in the victim's browser session within the trust relationship between their browser and the vulnerable server.

- What CVSS v3 Base Vector (AV:[N,A,L,P]/AC:[L,H]/PR:[N,L,H]/UI:[N,R]/S:[U,C]/C:[H,L,N]/I:[H,L,N]/A:[H,L,N]) would you assign to this vulnerability?

Software Vulnerabilities and Secure Programming

Software Vulnerability Classes

CWE: Common Weakness Enumeration (https://cwe.mitre.org/index.html)

- Catalog (ontology) of software weaknesses and vulnerabilities
  - A language for describing software vulnerabilities
- Very fine-grained (and not necessarily disjoint)
- Different "entry points" for browsing, e.g.,
  - https://cwe.mitre.org/index.html
  - https://cwe.mitre.org/data/index.html
- New vulnerability types added continuously
- Used in the Common Vulnerabilities and Exposures (CVE) entries
- "Most critical" vulnerabilities:
  - CWE-25 (https://cwe.mitre.org/data/definitions/25.html)

Software Vulnerability Classes

OWASP Top Ten (https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)

- Open Web Application Security Project (OWASP)
- Strictly limited to 10 vulnerabilities that developers should focus on ("most critical")
- Updated frequently (every few years), based on collecting data from industry
- OWASP Top Ten 2017:
  - A1:2017-Injection
  - A2:2017-Broken Authentication
  - A3:2017-Sensitive Data Exposure
  - A4:2017-XML External Entities (XXE)
  - A5:2017-Broken Access Control
  - A6:2017-Security Misconfiguration
  - A7:2017-Cross-Site Scripting (XSS)
  - A8:2017-Insecure Deserialization
  - A9:2017-Using Components with Known Vulnerabilities
  - A10:2017-Insufficient Logging & Monitoring

Vulnerability Databases (Reports)

CVE: Common Vulnerabilities and Exposures (https://cve.mitre.org/)

- CVE is a database of software vulnerabilities
  - Each entry has a unique id
  - Entries usually contain
    - Textual description of the vulnerability
    - Description of the affected software and version
    - Type of vulnerability (CWE)
    - Patch/fix instructions (if available)
    - Availability of an exploit (a proof of concept that shows how to make use of a vulnerability)
    - Standardized risk assessment: Common Vulnerability Scoring System (CVSS)
  - No obligation to register CVEs
    - Vendors may apply for a CVE id
    - Security researchers may apply for a CVE id
    - Most FLOSS projects register CVEs
- Vendors (or security research firms) may have their system, e g.,
  - Microsoft Security Bulletin (https://technet.microsoft.com/en-us/library/security/ms17-mar)

A Closer Look at the OWASP Top Ten

A1: 2017 – Injection

“Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.”

(OWASP Top Ten, 2017)

A Closer Look at the OWASP Top Ten

A1: 2017 – Injection: SQL Injection

Input from untrusted sources is used, without further checking, as part of a database query (i.e., a SQL expression).

- Also known as (examples)
  - CWE-89 – Failure to Preserve SQL Query Structure
- Affected languages
  - Any programming language that interfaces with a database can be affected, e.g., Python, Java, Ruby, PHP, C#, SQL (stored procedures), ...
- Possible impact
  - SQL injections can result in data loss, unauthorized access, ...

A Closer Look at the OWASP Top Ten

A1: 2017 – Injection: SQL Injection

- Vulnerable code (Python)

```
import pymysql
con = pymysql.connect('localhost', 'user17', 's$cret', 'testdb')
myid = user_input() # untrustworthy input

with con:
    cur = con.cursor()
    cur.execute("SELECT * FROM cities WHERE id=" + myid)
    cid, name, population  = cur.fetchone()
    print(cid, name, population)
```

- Secure programming recommendation
  - Use a prepared statement that ensures that query parameters are not used as SQL commands

```
    cur.execute("SELECT * FROM cities WHERE id=%s", myid)
```

A Closer Look at the OWASP Top Ten

A1: 2017 – Injection: SQL Injection

- Vulnerable code (Java)

```
String mname = request.getParameter("month");
String uid = session.getCurrentUserId();
String query = "SELECT username, startdate, transaction, amount
               FROM transaction_history
               WHERE user=" + uid + " AND month=" + mname;
ResultSet res = st.executeQuery(query);
```

- Secure programming recommendation
  - Use a prepared statement that ensures that query parameters are not used as SQL commands

```
PreparedStatement pstmt = conn.prepareStatement
             ("SELECT username, startdate, transaction,
                      amount
               FROM transaction_history
               WHERE user=? AND month=?");
pstmt.setString(1,uuid);
pstmt.setString(2,name);
ResultSet res = pstmt.executeQuery()
pstmt.close();
```

A Closer Look at the OWASP Top Ten

A1: 2017 – Injection: SQL Injection

- Prepared statements – Why do they protect us?

```
PreparedStatement pstmt = conn.prepareStatement
              ("SELECT username, startdate, transaction,
                       amount
                FROM transaction_history
                WHERE user=? AND month=?");
pstmt.setString(1,uuid);
pstmt.setString(2,name);
```

- Provide a clear separation between data (e.g., query parameters) and code (e.g., SQL statements)
- Can provide further type checks at compile time (parameter must be an integer, a string, etc.)

A Closer Look at the OWASP Top Ten

A1: 2017 – Injection: SQL Injection

- Prepared statements – A warning!

Prepared statements seem to be the universal solution to prevent SQL injections. While this is (mostly) true, they need to be used correctly!

- Consider the following (insecure!) use of a prepared statement

```
String mname = request.getParameter("month");
String uid = session.getCurrentUserId();
PreparedStatement pstmt = conn.prepareStatement
              ("SELECT username, startdate, transaction, amount
                FROM transaction_history
                WHERE user=" + uid + " AND month=" + mname);
ResultSet res = pstmt.executeQuery();
pstmt.close();
```

A Closer Look at the OWASP Top Ten

A1: 2017 – Injection: Command Injection

Input from untrusted sources is used, without further checking, as part of a system command.

- Also known as (examples)
  - CWE-78 – Improper Neutralization of Special Elements used in an OS Command
- Affected languages
  - Any programming language that allows executing system commands can be affected, e.g., Bash, Python, Java, Ruby, PHP, C#, ...
- Possible impact
  - Command injections can result in data loss, unauthorized access, ...

A Closer Look at the OWASP Top Ten

A1: 2017 – Injection: Command Injection

- Vulnerable code (Python)

```
import os
file = user_input() # untrustworthy input
cmd = 'wc -l $file'
os.system(cmd)
```

- Secure programming recommendation
  - Use system call that passes arguments as an array

```
import subprocess
file = user_input() # untrustworthy input
out = subprocess.Popen(['wc', '-l', $file],
            stdout=subprocess.PIPE,
            stderr=subprocess.STDOUT)
```

- Like in SQL injection, `subprocess.Popen()` can be used insecurely!

A Closer Look at the OWASP Top Ten

A1: 2017 – Injection: Command Injection

- Vulnerable code (Java)

```
Runtime.getRuntime().exec("os_cmd");
```

- Secure programming recommendation
  - Use whitelisting

```
String os_cmd = "..."
if (!os_cmd.matches("[a-zA-Z]++")) {
  throw new IllegalArgumentException();
}

Runtime.getRuntime().exec(os_cmd);
```

A Closer Look at the OWASP Top Ten

A2: 2017 – Broken Authentication

> **"Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently."**
>
> **(OWASP Top Ten, 2017)**

- Also known as (examples)
  - CWE-259 – Hard-Coded Password
  - CWE-261 – Weak Cryptography for Passwords
  - CWE-287 – Improper Authentication
- Affected languages
  - Any framework (or custom implementation) providing authentication, e.g., Python/Django, Ruby/Rails, Java/JSP, ...
- Note
  - Often also caused by insecure business processes (e.g., non-validated password recovery)
- Possible impact
  - Broken authentication can lead to the exposure of resources or functionality to unintended actors, possibly providing attackers with sensitive information or even execute arbitrary

A Closer Look at the OWASP Top Ten

A3: 2017 – Sensitive Data Exposure

> **"Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser."**
>
> **(OWASP Top Ten, 2017)**

- Also known as (examples)
  - CWE-200 – Information Exposure
  - CWE-201 – Information Exposure Through Sent Data
  - CWE-202 – Exposure of Sensitive Data Through Data Queries
- Affected languages
  - All
- Note
  - Often also caused by insecure business processes or using debug mode in production systems
- Possible impact
  - Sensitive data exposure can lead to the loss of data, provide additional information to attackers that build the basis for further attacks, ...

A Closer Look at the OWASP Top Ten

A4: 2017 – XML External Entities (XXE)

"Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks."

(OWASP Top Ten, 2017)

- Also known as (examples)
  - CWE-611 – Improper Restriction of XML External Entity Reference ('XXE')
- Affected languages
  - All languages processing XML encoded data, in particular web-based systems, ...
- Possible impact
  - XXE often allows attackers to read arbitrary files from the system, denial of service, or escalation of privileges (e.g., for XML-based authentication systems such as SAML), ...

A Closer Look at the OWASP Top Ten

A4: 2017 – XML External Entities (XXE)

- XML Billion Laughs Attack (https://en.wikipedia.org/wiki/Billion_laughs)

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

- This looks like an XML document with one root element `lolz`, containing the entity `&lol9`
- `&lol9` expands to a string containing ten `&lol8` entities
- Each `&lol8` expands to a string containing ten `&lol7`, ...
- After full expansion, this small XML documents will contain 10^9 "lol"s (requiring nearly 3 GB of memory)

A Closer Look at the OWASP Top Ten

A5: 2017 – Broken Access Control

> **"Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc."**
>
> **(OWASP Top Ten, 2017)**

- Also known as (examples)
  - CWE-284 - Improper Access Control (3.2)

A Closer Look at the OWASP Top Ten

A6: 2017 – Security Misconfiguration

"Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion."

(OWASP Top Ten, 2017)

A Closer Look at the OWASP Top Ten

A7: 2017 – Cross Site Scripting (XSS)

- Problem
  - User input is directly displayed in an output web page, without any sanitation. The typical attack pattern is:
    1. The attacker identifies a web site with XSS vulnerabilities
    2. The attacker creates a URL that submits malicious input (e.g., including malicious links or JavaScript code) to the attacked web site
    3. The attacker tries to induce the victim to click on the URL (e.g., by including the link in an email)
    4. The victim clicks the URL, hence submitting malicious input to the attacked web site
    5. The web site response page includes malicious links or malicious JavaScript code (executed on the victim's browser)

A Closer Look at the OWASP Top Ten

A7: 2017 – Cross Site Scripting (XSS)

- A simple example

```
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;

public class HelloServlet extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        String input = req.getHeader("USERINPUT");

        PrintWriter out = res.getWriter();
        out.println(input);  // echo User input.
        out.close();
    }
}
```

A Closer Look at the OWASP Top Ten

A7: 2017 – Cross Site Scripting (XSS)

- Fix
  - General recommendation: Use a well tested sanitization library (do not write your own)

```
import org.owasp.html.Sanitizers;
import org.owasp.html.PolicyFactory;

public class HelloServlet extends HttpServlet
{
    public void doGet (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
         String input = req.getHeader("USERINPUT");
         PolicyFactory sanitizer = Sanitizers.FORMATTING.and(Sanitizers.BLOCKS);
         String cleanInput = sanitizer.sanitize(input);

         PrintWriter out = res.getWriter();
         out.println(cleanInput);  // echo User input.
         out.close();
    }
}
```

- This example uses the OWASP Java HTML Sanitizer Project
  - (https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project)

A Closer Look at the OWASP Top Ten

A7: 2017 – Cross Site Scripting (XSS)

- Summary
  - Problem
    - User input is directly displayed in an output web page
  - Affected Languages
    - All programming languages used for building web sites, e.g., Perl, Python, Java, ASP, ASP.NET, JSP, PHP, C#, VB.Net, Ruby, ...
  - Countermeasures
    - Sanitize any user input which might reach output statements (including statements that write to a database or that save cookies)
    - Encode the output using HTML encoding, so that any malicious link or JavaScript code remains uninterpreted by the browser (use custom HTML encoding or custom HTML unencoding to preserve some safe HTML tags)

A Closer Look at the OWASP Top Ten

A8: 2017 – Insecure Deserialization

“Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.”

(OWASP Top Ten, 2017)

A Closer Look at the OWASP Top Ten

A9: 2017 – Using Components with Known Vulnerabilities

“Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.”

(OWASP Top Ten, 2017)

A Closer Look at the OWASP Top Ten

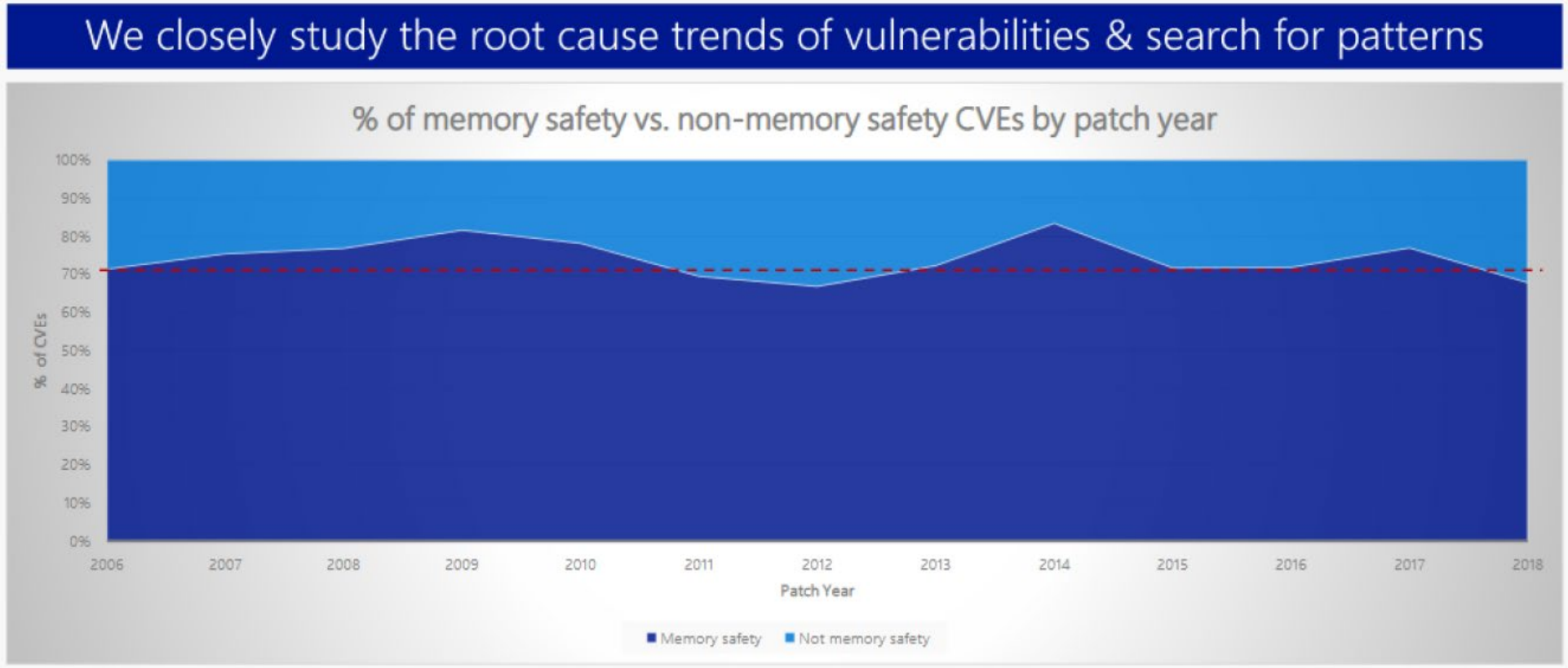
A10: 2017 – Insufficient Logging & Monitoring

"Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring."

(OWASP Top Ten, 2017)

A Special Class
Memory Corruption

- Microsoft: 70% of all security bugs are memory safety issues
  - https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/



A Special Class
Memory Corruption

- Buffer overflow: Problem
  - User data and control flow information (e.g., function pointer tables, return addresses) are mixed together on the stack and on the heap, hence user data exceeding a buffer may corrupt control flow information

A Special Class
Memory Corruption

- Buffer overflow: A simple example
  - Source code

```
1  void f(char* input) {
2      char buf[5];
3      char *hello="Hello";
4      strcpy(buf, input);
5      printf("%s %s\n", hello, buf);
6  }
```

  - Output

```
> gcc -0 buffer01 buffer01.c
> ./buffer01 Bob
Hello Bob
> ./buffer01 Alice
AWAVIAUATL Alice
> ./buffer01 Charlie
[1] segmentation fault
```

- Observations
  - `buf` only 5 characters (including string terminator \0)
  - `strcpy` copies until end of input
- What can go wrong?

A Special Class
Memory Corruption

- Buffer overflow: Memory layout

```
1  void foo (char *bar) {
2     float My_Float = 10.5; // Addr = 0x0023FF4C
3     char  c[28];           // Addr = 0x0023FF30
4     printf("My␣Float␣value␣=␣%f\n", My_Float); // Will print 10.500000
5
6     /*************************************************************
7      Memory map:
8         % : c allocated memory
9         # : My_Float allocated memory
10
11            *c                          *My_Float
12        0x0023FF30                      0x0023FF4C
13            |                               |
14            %%%%%%%%%%%%%%%%%%%%%%%%%%%%#####
15       foo("my string is too long !!!!! XXXXX");
16
17    memcpy will put 0x1010C042 (little endian) in My_Float value.
18    *************************************************************/
19
20    memcpy(c, bar, strlen(bar));  // no bounds checking...
21    printf("My␣Float␣value␣=␣%f\n", My_Float); // Will print 96.031372
22 }
```

A Special Class
Memory Corruption

- Buffer overflow: Impact
  - Denial of service (crashing process)
    - Warning: Usually this is only an indication of more severe impact vectors
  - Heartbleed (CVE-2014-0160)
    - CVSS Severity (version 2.0)
      - CVSS v2 *BaseScore*: 5.0 MEDIUM
      - Vector: (AV:N/AC:L/Au:N/C:P/I:N/A:N)
    - Impact: Allows unauthorized disclosure of information

  - CVE-2006-3444
    - CVSS Severity (version 2.0)
      - CVSS v2 *BaseScore*: 7.5 HIGH
      - Vector: (AV:N/AC:L/Au:N/C:P/I:P/A:P)
    - Impact: Allows local users to obtain privileges

A Special Class
Memory Corruption

- Buffer overflow: Prevention/Fix

```
1  #include <stdio.h>
2  #include <string.h>
3  void f(char* input) {
4      char buf[5];
5      char *hello="Hello";
6      strncpy(buf, input, 4);
7      buf[4]='\0';
8      printf("%s %s\n", hello, buf);
9  }
10 int main(int argc, char* argv[]) {
11     f(argv[1]);
12     return 0;
13 }
```

- Use counted versions of string functions (and read their documentation, see line 7)
- Use safe string libraries, or C++ strings
- Check loop termination and array boundaries
- Use C++/STL containers instead of C arrays

A Special Class
Memory Corruption

- Buffer overflow: Prevention/Fix
  - Example 1:
    - Use `fgets(buf, n, stdin)` instead of `gets(buf)`

```
1  void f() {
2       char buf[20];
3       gets(buf);
4       gets(buf);
5  }
```

```
1  void f() {
2       char buf[20];
3       fgets(buf,20,stdin) // NOT: gets(buf);
4       fgets(buf,20,stdin) // NOT: gets(buf);
5  }
```

A Special Class
Memory Corruption

- Buffer overflow: Prevention/Fix
  - Example 2:
    - Use/check the correct buffer size

```
1  void f() {
2      char buf[20];
3      char prefix[] = "http://";
4      strcpy(buf, prefix);
5      strncat(buf, path, sizeof(buf);
6  }
```

```
1  void f() {
2      char buf[20];
3      char prefix[] = "http://";
4      strcpy(buf, prefix);
5      strncat(buf, path, sizeof(buf)-7;
6  }
```

A Special Class
Memory Corruption

- Buffer overflow: Summary
  - **Problem:** User data and control flow information (e.g., function pointer tables, return addresses) are mixed together on the stack and on the heap, hence user data exceeding a buffer may corrupt control flow information

- **Affected Languages**
  - C, C++
  - Assembly languages
  - Unsafe sections of C#
  - Runtimes of safe languages (e.g., JVM)
  - OS interfaces
  - External libraries (FFI)

- **Countermeasures**
  - Use of counted versions of string functions
  - Use of safe string libraries or C++ strings
  - Carefully check loop terminations, array boundaries, etc.
  - Use of C++/STL containers (instead of arrays)
  - Check API and use it correctly
  - Use of `size_t`

A Special Class
Memory Corruption

- Keep Up-to-date With Language Development
  - Programming languages/tools continue to evolve
    - Learn
    - Avoid deprecated functions
    - Update your (old) code
    - Take compiler warnings serious
    - Use compilation protections (e.g., `-fstack-protector`)
    - Avoid old tutorials/books/stack overflow discussions

```
> man gets
GETS(3)                         Linux Programmer's Manual                        GETS(3)

NAME
       gets - get a string from standard input (DEPRECATED)

SYNOPSIS
       #include <stdio.h>

       char *gets(char *s);
```

Why Is It So Difficult?

Does this program filter <SCRIPT> tags?

```
public static String sanitiseScript(String input){
  String retval = input.toLowerCase();
  retval = retval.replaceAll("&le;", "<");
  retval = retval.replaceAll("&ge;", ">");
  retval = retval.replaceAll("␣", "");
  retval = retval.replaceAll("<script>", "");
  retval = retval.replaceAll("</script>", "");
  return retval;
}
```

- And now look at a real sanitization library
  - https://github.com/OWASP/java-html-sanitizer/
  - https://github.com/OWASP/java-html-sanitizer/blob/master/src/main/java/org/owasp/html/Encoding.java

Summary

- Many vulnerabilities are caused by missing input or output sanitization

- In general, use (in this order)
  1. Framework provided counter measures (e.g., SQL prepared statement)
  2. Whitelisting
  3. Blacklisting

- Avoid writing your own sanitizers, use well tested and widely used implementations, e.g.,
  - OWASP project: https://github.com/OWASP/java-html-sanitizer
  - HTML Rule Sanitizer: https://github.com/Vereyon/HtmlRuleSanitizer
  - ...

Reading List

- Ross J. Anderson. Security Engineering: A Guide to Building Dependable Distributed Systems. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2001.
  - The complete book is available at: http://www.cl.cam.ac.uk/~rja14/book.html
- OWASP Top 10 – 2017, 2017.

Thanks for your attention!

- Any questions or remarks?