Common Security Attacks - Webinar

Secure Your Web Application with OWASP

Lesson Objectives

By the end of this lesson, you will be able to…

1. Understand Secure Programming
2. Understand OWASP
3. Try out different attacks with this site https://www.hacksplaining.com/book (If it works)

Understand Secure Programming

Data breaches that mostly occur from **common outside attacks**.

These breaches are specifically related to **web applications**.

Why Is Security Important on Web Applications?

In 2012, Yahoo was **hacked.** Due to poor database security configuration, user passwords were released publicly.

You can imagine how much this hurt their business, and how much money they lost - not to mention the effect it had on 450,000 users! 😱

COULD IT HAVE BEEN PREVENTED?

Sure! 😎 Did you know that 80% of the data breaches and hacks committed on web applications can be traced back to poor coding practices leave the code vulnerable, giving those malicious characters an advantage?

Annualized Loss Expectancy (ALE), which is a measurement of risk taken based on the probability that the asset will be breached.

Companies will look at the value of an asset and the cost to secure it.

Also, consider the likelihood of a breach happening.

Let’s say an asset is worth $100, and there's a 20% likelihood that it will be hacked. Your SLE is $20.

Single Loss Expectancy = Net Asset Value * Exposure Factor
SLE = NAV x EF
$20 = $100 x .2

Annualized Loss Expectancy = Single Loss Expectancy * Annual Rate of Occurrence
ALE = SLE x ARO
$40 = $20 x 2

In conclusion

So it looks like this asset valued at $100 will cost the business $40 if it is breached over a year.

If it costs $5 to secure it over a year, maybe it's worth it!

Who decides to pay the cost to secure?

The CEO

Things to Remember

- Data breaches on web applications are common and happen even to large companies.
- A web application attack can cause a business to lose a lot of money and their reputation.
- Risk is measured by comparing the value of data with the cost to secure it.
- The CEO or the president of the company is held responsible for the security policies in the organization and appropriating the proper funding for security.

What Are Security Standards for Web Applications?

These core principles are a key part of how security policies are created. Can vary company to company depending the regulation they want to follow.

Confidentiality is also known as **privacy.** It is the assurance that unauthorized people do not access sensitive information.

Integrity is the assurance that the data is trustworthy and has not been altered by unauthorized people.

Availability is that there is no disruption to a service or accessibility to the data.

The CIA Triad

What are the main security regulations or regulatory bodies for Web Applications?

1) In the EU in 2018, the **General Data Protection Regulation (GDPR)** law went into effect.

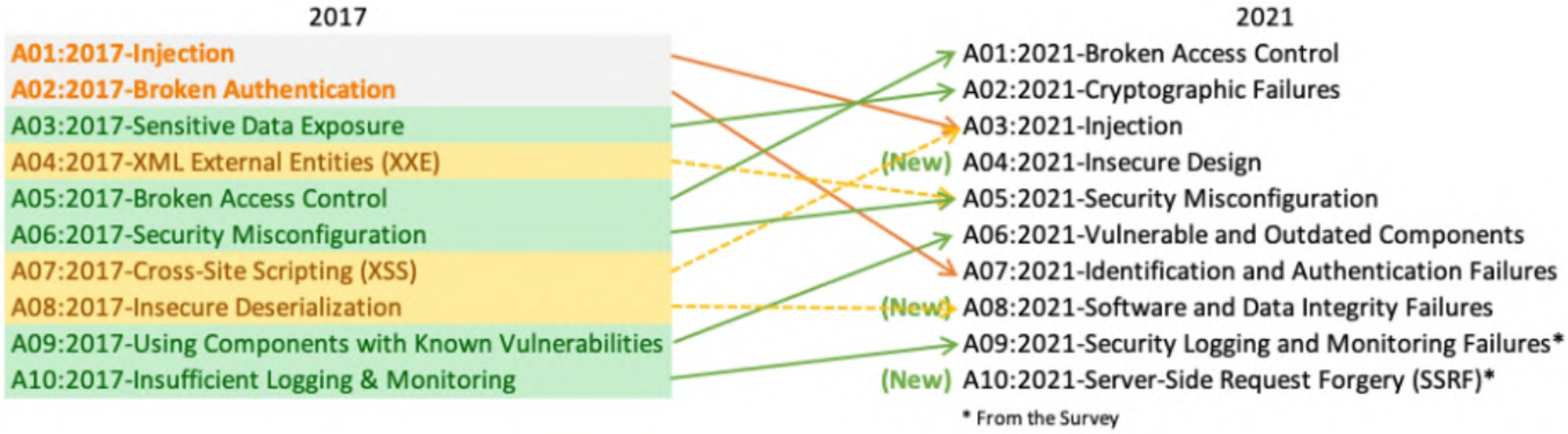
   Ensure that **you secure all Personal Identifiable Information data** from unauthorized users in transmission and at rest. All PII requested and possessed by the business must have a specified use.

   Also, provide an **option to delete PII data** upon customer request. For example, if someone wants to opt out of marketing emails, ensure there is an option to opt out and unsubscribe. Once they unsubscribe, it is essential that the email address is deleted from storage.

2) Another is HIPAA - **Health Insurance Portability and Accountability Act** for collecting confidential patient data.

3) OWASP - the one web developers are interested in.

Open Web Application Security Project (OWASP) – Top Ten (Most Common) Application Attacks



Prevent an Injection Attack

Injection is when an attacker is **injecting code,** a **script,** or **command** using your web application.

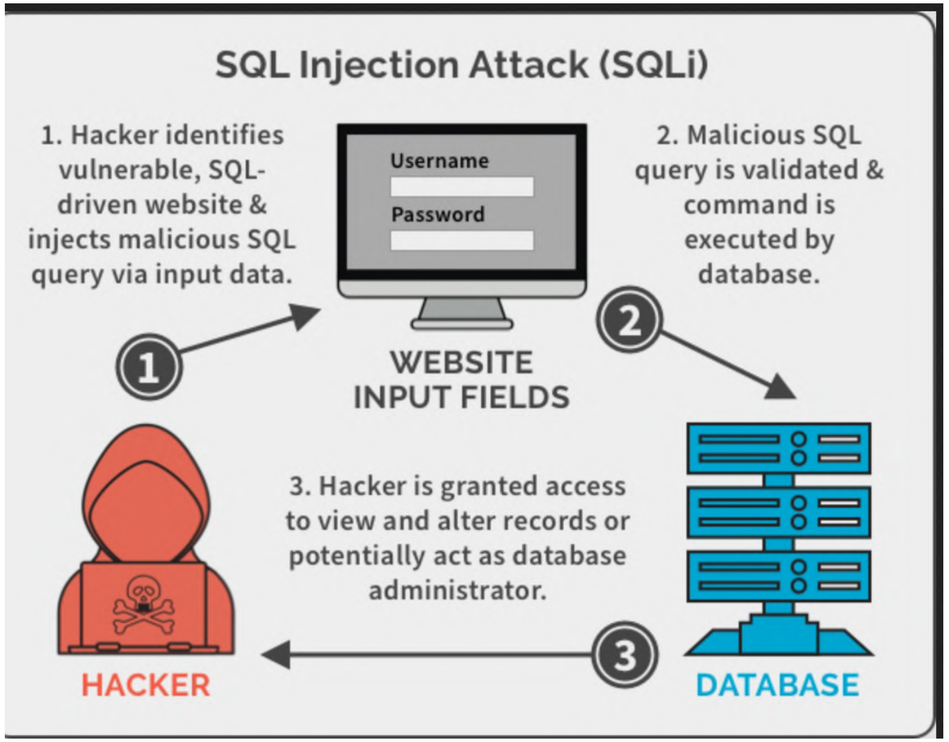
An injection attack can be used to take down an entire system, access files, and delete them. It causes a lot of damage, and all a hacker needs is a place to input code in an application.

Possible Solutions:

Input Validation - It limits what the user can put into the text box.
Sanitization and Prepared Statements

Use an Object Relational Mapper (ORM) like pg or knex
It obfuscates your query hides how the SQL really looks.

Use the OWASP Library. The OWASP organization has a library called The **OWASP Enterprise Security API (ESAPI)** can be used to secure your web applications, so check it out to see if it works with the language and database that you are using!



Prevent Broken Authentication

When logging in if the username and password match what is in the database, you will be **authenticated** for a session.
A session is a specified amount of time that an authenticated user will have access to specific pages and activities on the application.

This process can be hijacked - An unauthorized user can gain a lot of sensitive data when hijacking someone's session.

Possible Solutions: Currently, JWT is a viable alternative to the session. JWT is a stateless Auth mechanism. A Bearer token is sent in the header of every authenticated request. The payload of the JWT token contains the necessary details used for authorization. This is useful when we want to expose some part of our data as an API resource. However, unlike sessions, JWT is stateless and hence the logout code has to be implemented on the client-side. You can set an expiry timestamp in JWT payload but cannot force a logout.

Things to remember

- Broken authentication may happen when a web app isn't coded to secure cookies.
- Use input validation and limit login tries.
- Brute force or credential stuffing tools are common.
- Limit session times.

Protect Data in Transit

An HTTP URL is subject to an attack because it is **unencrypted.**

Okay. So you put a URL in the browser address bar and hit enter.

Your browser will initiate a **TCP** connection that will send a **GET** or **POST request** to connect with the web server associated with the domain name or IP address.

If the web server chooses to establish the **TCP connection** with the browser, a response with the status code and the requested file (typically the index.html file for the web page) and data transmission can begin.

A Man-In-The-Middle (MITM) Attack - Data theft and session hijacking

Free wifi sites are vulnerable

Possible Solutions:

HTTPS Everywhere! - a secure version of HTTP that uses TLS to encrypt all data transmission. Use the HTTPS module on Node.js

1. Only use GET for retrieving information.
2. Use POST for information that will be manipulated.
3. All POST requests should use HTTPS/SSL to ensure the body is encrypted (if you absolutely have to use HTTP, use it with GET).
4. Vet any third party modules you use for creating GET/POST requests and use HTTPS for all of them!
5. Use CORs if you are going outside the domain.

Things to remember

- Use HTTPs for your entire site, even if it doesn't have sensitive data.
- Use GET requests for retrieving information, and POST for changing information.
- Secure your cookies to be transmitted through the header, and via HTTPS.
- Secure your sessions by adding an expiration date, securing the ID, and not putting that ID in the URL.
- Using encryption in the transport layer can prevent MITM attacks!

Protect Data at Rest

The **data at rest** is the content in the database.

First of all, some of you may be web developers that do not work directly on **database configurations;** however, it is important to learn all of these strategies because someday, you might. Also, if there are issues that can be resolved on the database side and you can resolve them, you have saved the day!

Possible Solutions:

- Secure your database with encryption.
- DO: Use hash algorithms that are secure such as Argon5, Scrypt, Bcrypt, and PBKDF2.
- DON'T: Use just SHA and MD5 hash algorithms without a salt randomization.
- Rainbow tables automate logins with pre-encrypted passwords!
- Data masking can be used to secure sensitive data on the database.

Things to remember

Techniques to Hide Sensitive Data on a Database

Anonymization is a technique applied by the OWASP organization for hiding private data by encrypting, scrambling, and removing parts of data. For example, if a request is made for someone's date of birth as an identifier, only the year will be provided by the database.

Pseudonymisation is a GDPR process that replaces PII with artificial identifiers and pseudonyms to hide sensitive data.

Data minimization allows a business process to abide by the GDPR ruleset. The business is only to request, store, and process PII that is required. In other words, any PII requested must have a solid business reason.

Stop Broken Access Control

Access control is setting up your web application to make sure that the users of the web application can **only access the sites that are designated under that role.**

Failure to Restrict URL Access

Common attacks occur when a **URL bypasses authentication** with the guise of having already been authenticated. Hackers use format and pattern knowledge to write the URL for privileged pages that have not been securely configured.

Insecure Direct Object References (IDOR)

A malicious user has techniques to access a lot of the hard coding in a web application. Some of this code can reveal **how a database is organized** with regards to **formatting** and **pattern.** Providing a few pieces of the puzzle can allow an unauthorized user to use their knowledge to expose information for further probing.

Direct object references can also pop up in an **error code.** The hacker can start manipulating entries to see what kinds of exceptions and errors pop up and what they say. It provides more information to search in the URL!

Possible Solutions:

1. Ensure that all pages have an authentication check.
2. Customize your exceptions and error codes.

Things to remember

- Web apps with Broken Access Control do not ensure that every page is locked for authentication.
- Direct object references can lead a hacker to understand the patterns and setup of the web applications.
- Do not use predicable names or direct references to the database in the URL.
- Prevent null byte attacks by protecting your source code.
- Use indirect object references with parameters and key-value pairs.
- Customize your error codes so they do not reveal database attributes.

Stop Cross-Site Scripting (XSS)

What Is Cross-Site Scripting?
Cross-site scripting attacks are made to take over your **browser.** A hacker that accomplishes this has access to your **cookies** and **sessions** which can hold **sensitive data!** This culprit can also make unauthorized changes to the web application and create links that will take you to **malicious sites!** 😱

The outlaw trying to take over your browser creates a **script that is injected into the web application.**

What Is Cross-Site Scripting Forgery (CSRF)?
A hacker can create an **XSS link** and distribute it through social engineering to gain access to a user's browser. And **hijack a session token** in the browser (in real time) to access a bank session for example.

Things to remember

- **Cross-site scripting** is a script that can be executed in your website.
- Prevent XSS with input validation and input encoding.
- Protect your cookies by setting the HttpOnly flag.
- CSRF - Cross-Site Scripting Forgery (CSRF) attacks can happen with social engineered links.
- CSRF attacks perform transactions without the user knowing.

New for 2021

A04:2021-Insecure Design is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to "move left" as an industry, we need more threat modeling, secure design patterns and principles, and reference architectures. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks.

How to Prevent

- Establish and use a secure development lifecycle with AppSec professionals to help evaluate and design security and privacy-related controls
- Establish and use a library of secure design patterns or paved road ready to use components
- Use threat modeling for critical authentication, access control, business logic, and key flows
- Integrate security language and controls into user stories
- Integrate plausibility checks at each tier of your application (from frontend to backend)
- Write unit and integration tests to validate that all critical flows are resistant to the threat model. Compile use-cases *and* misuse-cases for each tier of your application.
- Segregate tier layers on the system and network layers depending on the exposure and protection needs
- Segregate tenants robustly by design throughout all tiers
- Limit resource consumption by user or service

New for 2021

A08:2021-Software and Data Integrity Failures is a new category for 2021, focusing on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity. One of the highest weighted impacts from Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS) data mapped to the 10 CWEs in this category. **A8:2017-Insecure Deserialization** is now a part of this larger category.

How to Prevent

- Use digital signatures or similar mechanisms to verify the software or data is from the expected source and has not been altered.
- Ensure libraries and dependencies, such as npm or Maven, are consuming trusted repositories. If you have a higher risk profile, consider hosting an internal known-good repository that's vetted.
- Ensure that a software supply chain security tool, such as OWASP Dependency Check or OWASP CycloneDX, is used to verify that components do not contain known vulnerabilities
- Ensure that there is a review process for code and configuration changes to minimize the chance that malicious code or configuration could be introduced into your software pipeline.
- Ensure that your CI/CD pipeline has proper segregation, configuration, and access control to ensure the integrity of the code flowing through the build and deploy processes.
- Ensure that unsigned or unencrypted serialized data is not sent to untrusted clients without some form of integrity check or digital signature to detect tampering or replay of the serialized data

New for 2021

A10:2021-Server-Side Request Forgery is added from the Top 10 community survey (#1). The data shows a relatively low incidence rate with above average testing coverage, along with above-average ratings for Exploit and Impact potential. This category represents the scenario where the security community members are telling us this is important, even though it's not illustrated in the data at this time.

How to Prevent

Developers can prevent SSRF by implementing some or all the following defense in depth controls:

From Network layer

- Segment remote resource access functionality in separate networks to reduce the impact of SSRF
- Enforce “deny by default” firewall policies or network access control rules to block all but essential intranet traffic.

  Hints:

  ~ Establish an ownership and a lifecycle for firewall rules based on applications.

  ~ Log all accepted *and* blocked network flows on firewalls (see A09:2021-Security Logging and Monitoring Failures).

From Application layer:

- Sanitize and validate all client-supplied input data
- Enforce the URL schema, port, and destination with a positive allow list
- Do not send raw responses to clients
- Disable HTTP redirections
- Be aware of the URL consistency to avoid attacks such as DNS rebinding and “time of check, time of use” (TOCTOU) race conditions

How Can a Business Regulate the OWASP Standard?

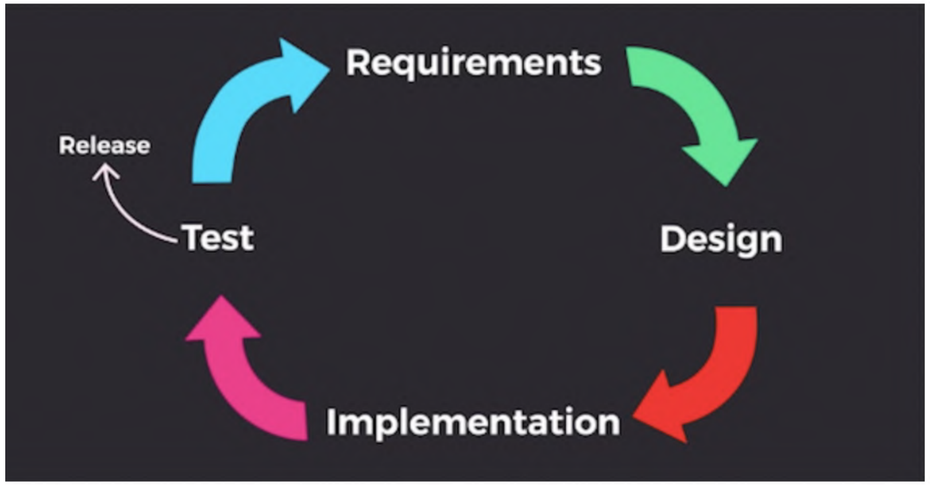
- The business can’t just accredit themselves as being compliant to regulations like OWASP, PCI, or HIPAA.
- They will need a third party vendor that specializes in examining a business for compliance in those regulations.
- These regulations can be pretty strict with larger companies, so it can take months of testing to get certified.

Take Control and Beat the Hackers at their Own Game.

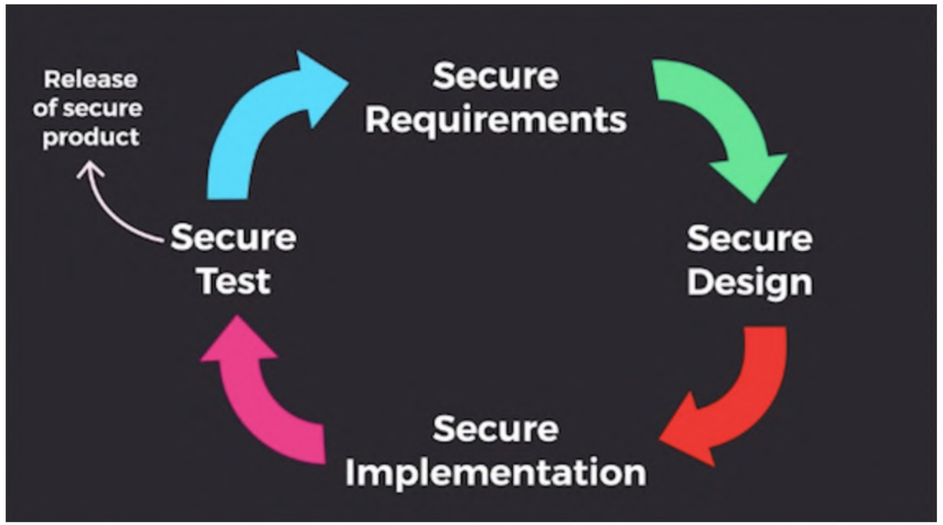
There are a lot of available tools to test that a web application is secure from common attacks.

- Vulnerability scanning, peach fuzzing, penetration testing, Metasploit, and Burp Suite are great for testing attacks on a web application.
- There are areas of the web that are not directly accessible to the general public, and require special techniques to access.
- Zero-day vulnerabilities can come up with any component to ensure that you are up-to-date and patched for new attacks.

Learn the Secure Software Development Life Cycle



The Software Development Life Cycle (SDLC)



Inject security into your development life cycle!

Learn the Secure Software Development Life Cycle

- The SDLC is used by organizations with large software projects.
- Traditionally it does not include secure coding as an integral part of the process.
- The result of not ensuring security in SDLC is additional time and money used to fix unsecured code.
- Use SSDLC to add security awareness in every phase of SDLC!

Putting it all together

OWASP TOP TEN 2017 and we looked at a few new ones for 2021

The top ten vulnerabilities in web security, identified by OWASP:

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using components with known vulnerabilities
10. Insufficient logging and monitoring

Lesson Summary

- Research major security events

SolarWinds

- Research vulnerabilities!
- Proactively secure your web app!
- Stay up to date!
- Think about specializing or training in Cyber Security