ASSIGNMENT-1

Kevin Mitnick:
Kevin Mitnick, a prominent figure in the history of computer hacking in the United States, began his professional life when he was just a teenager. In 1981, he was accused of stealing computer manuals from Pacific Bell and was arrested for the crime. His accomplishment of hacking the North American Defence Command (NORAD) in 1982 served as the impetus for the film War Games, which was released in 1983. In 1989, he broke into the network of Digital Equipment Corporation (DEC) and stole copies of the company's software. This act catapulted Mitnick to prominence at a time when DEC was the preeminent computer manufacturer in the industry. He was eventually apprehended, tried, and sentenced to time behind bars. During the time that he was out on conditional release, he broke into the voicemail systems of Pacific Bell.

Throughout his entire career as a hacker, Mitnick never made use of the access or data he obtained illegally. It's common knowledge that he once took complete command of Pacific Bell's network for no other reason than to demonstrate that it was possible. In connection with the incident involving Pacific Bell, a warrant was issued for his arrest; however, Mitnick evaded capture and remained in hiding for over two years. After being apprehended, he was sentenced to multiple terms of imprisonment for the crimes of wire fraud and computer fraud.

Even though Mitnick eventually wore the white hat, he may still fall into the grey area of wearing both hats. According to Wired, in 2014 he established "Mitnick's Absolute Zero Day Exploit Exchange," a platform that offers critical software vulnerabilities that have not been patched to whoever places the highest bid.

Type of Hacker: Kevin Mitnick is a former hacker turned cybersecurity consultant. He was primarily known as a Black Hat Hacker before his capture in 1995. He engaged in various cybercrimes, including hacking into computer networks, stealing sensitive information, and wiretapping.

Anonymous (Collective):
In 2003, Anonymous got its start in an unnamed forum on the message boards of the website 4chan. The group is not very well organized, and its attention is only tangentially concentrated on the idea of social justice. For instance, in 2008, the organization had a disagreement with the Church of Scientology and began to disable its websites. This had a negative impact on the websites' search rankings in Google, and the organization's fax machines were inundated with images that were all black. When a group of "Anons" marched past Scientology centers around the world in March 2008, they did so while wearing the now-famous Guy Fawkes mask. According to an article that was published in The New Yorker, despite the fact that the FBI and other law enforcement agencies have been successful in locating some of the group's more active

members, the absence of any real hierarchy makes it nearly impossible to identify or eradicate Anonymous as a whole.

Type of Hackers: Anonymous is a loosely organized group of hackers, hacktivists, and activists. They can be considered Hacktivists or Gray Hat Hackers, as their actions range from exposing government or corporate wrongdoing to engaging in distributed denial of service (DDoS) attacks.

Adrian Lamo:
In 2001, a young man named Adrian Lamo, then 20 years old, used an unprotected content management tool at Yahoo to alter a Reuters article and add a fake quote that was supposed to be attributed to a previous Attorney General named John Ashcroft. Lamo frequently broke into computer systems and then informed both the press and the people he had hacked. In certain situations, he would assist in cleaning up the mess in order to improve their safety. However, as Wired points out, Lamo went too far in 2002 when he hacked into the intranet of The New York Times, added himself to the list of expert sources, and started conducting research on prominent members of the public. Lamo's actions were inappropriate. Lamo was given the nickname "The Homeless Hacker" due to the fact that he frequently went about his days carrying little more than a backpack and frequently did not have a permanent address.

Type of Hacker: Adrian Lamo was known as a Gray Hat Hacker. He was involved in various activities, including ethical hacking and exposing security vulnerabilities. Lamo gained notoriety for turning in Chelsea Manning (formerly Bradley Manning), who had leaked classified information to WikiLeaks.

Albert Gonzalez:
Gonzalez, also known as "soupnazi," is said to have gotten his start as the "troubled pack leader of computer nerds" at his Miami high school, as reported by the New York Daily News. After some time, he started using the illicit trade website Shadowcrew.com and quickly rose through the ranks to become one of the site's most respected hackers and moderators. At the age of 22, Gonzalez was taken into custody in New York on charges of debit card fraud related to the theft of data from millions of card accounts. In order to avoid serving time in prison, he turned himself in to the Secret Service as an informant and ultimately assisted in the prosecution of dozens of members of Shadowcrew.

Gonzalez did not stop his involvement in illegal activities even while he was working as a paid informant. Gonzalez, along with a group of confederates, was responsible for the theft of more than 180 million payment card accounts belonging to various businesses, such as OfficeMax, Dave and Buster's, and Boston Market. According to an article published in the New York Times

Magazine, Gonzalez's attack on the American retailer TJX in 2005 was the first serial data breach of credit information. This infamous hacker and his team created back doors in several corporate networks by using a technique known as basic SQL injection. They used these back doors to steal an estimated $256 million from TJX alone. In 2015, during Gonzalez's sentencing, the federal prosecutor described the human victimization that Gonzalez caused as "unparalleled."

Type of Hacker: Albert Gonzalez was a notorious Black Hat Hacker involved in credit card and identity theft. He orchestrated the infamous Heartland Payment Systems data breach and is considered one of the most significant cybercriminals in recent history.

Matthew Bevan and Richard Pryce:
Matthew Bevan and Richard Pryce are a team of British hackers who hacked into multiple military networks in 1996, including Griffiss Air Force Base, the Defense Information System Agency and the Korean Atomic Research Institute (KARI). Bevan (Kuji) and Pryce (Datastream Cowboy) have been accused of nearly starting a third world war after they dumped KARI research onto American military systems. Bevan claims he was looking to prove a UFO conspiracy theory, and according to the BBC, his case bears resemblance to that of Gary McKinnon. Malicious intent or not, Bevan and Pryce demonstrated that even military networks are vulnerable.

Type of Hackers: Matthew Bevan and Richard Pryce were known as Hacktivists. They gained attention for hacking into U.S. military computers in the late 1990s as a form of protest against nuclear weapons testing.

Jeanson James Ancheta:
Jeanson James Ancheta had no interest in hacking systems for credit card data or crashing networks to deliver social justice. Instead, Ancheta was curious about the use of bots—software-based robots that can infect and ultimately control computer systems. Using a series of large-scale "botnets," he was able to compromise more than 400,000 computers in 2005. According to Ars Technica, he then rented these machines out to advertising companies and was also paid to directly install bots or adware on specific systems. Ancheta was sentenced to 57 months in prison. This was the first time a hacker was sent to jail for the use of botnet technology.

Type of Hacker: Jeanson James Ancheta was a Black Hat Hacker who created a network of compromised computers (a botnet) and used it to launch various attacks and distribute malware, making him a Botnet Operator.

Michael Calce (aka MafiaBoy):

In February 2000, 15-year-old Michael Calce, also known as "Mafiaboy," discovered how to take over networks of university computers. He used their combined resources to disrupt the number-one search engine at the time: Yahoo. Within one week, he'd also brought down Dell, eBay, CNN and Amazon using a distributed-denial-of-service (DDoS) attack that overwhelmed corporate servers and caused their websites to crash. Calce's wake-up call was perhaps the most jarring for cyber crime investors and internet proponents. If the biggest websites in the world—valued at over $1 billion—could be so easily sidelined, was any online data truly safe? It's not an exaggeration to say that the development of cyber crime legislation suddenly became a top government priority thanks to Calce's hack.

Type of Hacker: Michael Calce was a Black Hat Hacker known for launching large-scale DDoS attacks in the early 2000s. He targeted high-profile websites, including Yahoo!, eBay, and Amazon.

Kevin Poulsen:
In 1983, a 17-year-old Poulsen, using the alias Dark Dante, hacked into ARPANET, the Pentagon's computer network. Although he was quickly caught, the government decided not to prosecute Poulsen, who was a minor at the time. Instead, he was let off with a warning.

Poulsen didn't heed this warning and continued hacking. In 1988, Poulsen hacked a federal computer and dug into files pertaining to the deposed president of the Philippines, Ferdinand Marcos. When discovered by authorities, Poulsen went underground. While he was on the run, Poulsen kept busy, hacking government files and revealing secrets. According to his own website, in 1990, he hacked a radio station contest and ensured that he was the 102nd caller, winning a brand new Porsche, a vacation, and $20,000.

Poulsen was soon arrested and barred from using a computer for three years. He has since converted to white hat hacking and journalism, writing about cyber security and web-related socio-political causes for Wired, The Daily Beast and his own blog Threat Level. Paulson also teamed with other leading hackers to work on various projects dedicated to social justice and freedom of information. Perhaps most notably, working with Adam Swartz and Jim Dolan to develop the open-source software SecureDrop, initially known as DeadDrop. Eventually, Poulsen turned over the platform, which enabled secure communication between journalists and sources, to the Freedom of Press Foundation.

Type of Hacker: Kevin Poulsen was a Black Hat Hacker who gained notoriety for hacking into phone systems and rigging radio station contests. After serving time in prison, he became a respected journalist specializing in cybersecurity.

Jonathan James:

Using the alias cOmrade, Jonathan James hacked several companies. According to the New York Times, what really earned James attention was his hack into the computers of the United States Department of Defense. Even more impressive was the fact that James was only 15 at the time. In an interview with PC Mag, James admitted that he was partly inspired by the book The Cuckoo's Egg, which details the hunt for a computer hacker in the 1980s. His hacking allowed him to access over 3,000 messages from government employees, usernames, passwords and other sensitive data.

James was arrested in 2000 and was sentenced to a six months house arrest and banned from recreational computer use. However, a probation violation caused him to serve six months in jail. Jonathan James became the youngest person to be convicted of violating cyber crime laws. In 2007, TJX, a department store, was hacked and many customer's private information were compromised. Despite a lack of evidence, authorities suspect that James may have been involved.

In 2008, James committed suicide by gunshot. According to the Daily Mail, his suicide note stated, "I have no faith in the 'justice' system. Perhaps my actions today, and this letter, will send a stronger message to the public. Either way, I have lost control over this situation, and this is my only way to regain control."

Type of Hacker: Jonathan James was a Black Hat Hacker who, at a young age, hacked into various high-profile systems, including NASA's. He was known for his technical skills but tragically took his own life in 2008.

ASTRA (Pseudonym):
This hacker is distinct from the others on this list in that he has never been outed to the general public in any capacity. On the other hand, the Daily Mail reports that certain information regarding ASTRA has become public. In particular, the fact that he was apprehended by authorities in 2008 and identified as a 58-year-old Greek mathematician at the time of his capture. According to reports, he had been breaking into the computers of the Dassault Group for nearly half a decade. During that time, he was dishonest and stole data and software related to cutting-edge weapons technology, which he later sold to 250 different people in different parts of the world. Damages totaling 360 million euros were incurred by the Dassault Group as a result of his hacking. The reason why his full identity has never been disclosed is a mystery to everyone, but the word "astra" can be translated as "weapon" in Sanskrit.

Type of Hacker: ASTRA was a Black Hat Hacker who targeted online gaming companies, particularly in the MMO (Massively Multiplayer Online) gaming community. ASTRA stole in-game items and currency for profit.

21BCE7984

TOP 10 OWASP Vulnerabilities

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Injection

- Description: Injection vulnerabilities occur when untrusted data is sent to an interpreter as part of a command or query. This can include SQL injection, OS command injection, and more. Attackers can manipulate input to execute malicious code, access unauthorized data, or take control of the application or server.
- Business Impact: Injection attacks can lead to unauthorized data access, data manipulation, or complete system compromise. This can result in data breaches, data loss, and severe damage to an organization's reputation.

CWE-287: Improper Authentication

Broken Authentication

- Description: This vulnerability occurs when an application's authentication mechanisms are poorly implemented or misconfigured. Attackers can exploit weak authentication to impersonate users and gain unauthorized access to accounts or sensitive data.
- Business Impact: Broken authentication can lead to identity theft, unauthorized actions, financial losses, and damage to an organization's reputation.

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

Sensitive Data Exposure

- Description: Sensitive Data Exposure occurs when an application fails to adequately protect sensitive information, such as passwords or credit card details. Attackers can steal this data, leading to potential data breaches.
- Business Impact: Data breaches can result in legal consequences, financial losses, and a loss of customer trust, particularly if the data is subject to privacy regulations.

CWE-611: Improper Restriction of XML External Entity Reference

XML External Entity (XXE)

- Description: XXE vulnerabilities occur when an application processes XML input without proper validation. Attackers can use malicious XML input to access internal files, launch denial-of-service attacks, or even execute remote code.

- Business Impact: Exploiting XXE can lead to disclosure of confidential data, denial of service, and significant disruption to services.

CWE-284: Improper Access Control

Broken Access Control

- Description: Broken Access Control vulnerabilities result from inadequate enforcement of user permissions. Attackers can exploit this to gain unauthorized access to functionalities or data.
- Business Impact: Weak access controls can lead to unauthorized actions, data breaches, and a loss of confidentiality and integrity.

CWE CATEGORY: Configuration

Security Misconfiguration

- Description: Security misconfiguration occurs when security settings are not properly configured or left at default values. Attackers can exploit these weaknesses to gain unauthorized access or manipulate the system.
- Business Impact: Misconfigurations can expose sensitive data and resources, leading to unauthorized access, data breaches, service disruptions, and potential legal issues.

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

Cross-Site Scripting (XSS)

- Description: XSS vulnerabilities enable attackers to inject malicious scripts into web pages viewed by other users. These scripts can steal data, hijack sessions, or deface websites.
- Business Impact: XSS attacks can harm an organization's reputation and user trust, leading to data theft, session hijacking, and website defacement.

CWE-502: Deserialization of Untrusted Data

Insecure Deserialization

- Description: Insecure deserialization vulnerabilities occur when untrusted data is deserialized without proper validation. Attackers can exploit this to execute malicious code and compromise the application or server.

- Business Impact: Exploiting insecure deserialization can lead to service outages, data breaches, and damage to reputation.

CWE-937: Using Components with Known Vulnerabilities

Using Components with Known Vulnerabilities

- Description: This vulnerability arises when applications use outdated or known vulnerable software components. Attackers can exploit these vulnerabilities to compromise the security of the application.
- Business Impact: Relying on vulnerable components can lead to unauthorized access, service disruption, and security compromises.

CWE-798: Use of Hard-coded Credentials

Insufficient Logging and Monitoring (CWE-798)

- Description: Insufficient logging and monitoring hinder an organization's ability to detect and respond to security incidents. This can lead to delayed incident response, prolonged data breaches, and other security issues.
- Business Impact: Inadequate monitoring can result in severe security incidents going unnoticed, potentially causing significant financial losses and reputational damage.

1. **A01:2021-Broken Access Control**

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. Common access control vulnerabilities include:

- Violation of the principle of least privilege or deny by default, where access should only be granted for particular capabilities, roles, or users, but is available to anyone.
- Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests.
- Permitting viewing or editing someone else's account, by providing its unique identifier (insecure direct object references)
- Accessing API with missing access controls for POST, PUT and DELETE.
- Elevation of privilege. Acting as a user without being logged in or acting as an admin when logged in as a user.
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token, or a cookie or hidden field manipulated to elevate privileges or abusing JWT invalidation.
- CORS misconfiguration allows API access from unauthorized/untrusted origins.
- Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

How to Prevent

Access control is only effective in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- 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 limits 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.

Developers and QA staff should include functional access control unit and integration tests.

2. **A02:2021 – Cryptographic Failures**

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS). For all such data:

- Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, FTP also using TLS upgrades like STARTTLS. External internet traffic is hazardous. Verify all internal traffic, e.g., between load balancers, web servers, or back-end systems.
- Are any old or weak cryptographic algorithms or protocols used either by default or in older code?
- Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing? Are crypto keys checked into source code repositories?
- Is encryption not enforced, e.g., are any HTTP headers (browser) security directives or headers missing?
- Is the received server certificate and the trust chain properly validated?
- Are initialization vectors ignored, reused, or not generated sufficiently secure for the cryptographic mode of operation? Is an insecure mode of operation such as ECB in use? Is encryption used when authenticated encryption is more appropriate?
- Are passwords being used as cryptographic keys in absence of a password base key derivation function?
- Is randomness used for cryptographic purposes that was not designed to meet cryptographic requirements? Even if the correct function is chosen, does it need to be seeded by the developer, and if not, has the developer over-written the strong seeding functionality built into it with a seed that lacks sufficient entropy/unpredictability?

- Are deprecated hash functions such as MD5 or SHA1 in use, or are non-cryptographic hash functions used when cryptographic hash functions are needed?
- Are deprecated cryptographic padding methods such as PKCS number 1 v1.5 in use?
- Are cryptographic error messages or side channel information exploitable, for example in the form of padding oracle attacks?

How to Prevent

Do the following, at a minimum, and consult the references:

- 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.

3. A03:2021 – Injection

An application is vulnerable to attack when:

- User-supplied data is not validated, filtered, or sanitized by the application.

- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.

Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections. Automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs is strongly encouraged. Organizations can include static (SAST), dynamic (DAST), and interactive (IAST) application security testing tools into the CI/CD pipeline to identify introduced injection flaws before production deployment.

How to Prevent

Preventing injection requires keeping data separate from commands and queries:

- The preferred option is to use a safe API, which avoids using the interpreter entirely, provides a parameterized interface, or migrates to Object Relational Mapping Tools (ORMs).
  Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data or executes hostile data with EXECUTE IMMEDIATE or exec().
- Use positive server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.
  Note: SQL structures such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report- writing software.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

4. **A04:2021 – Insecure Design**

Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.

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

5. A05:2021 – Security Misconfiguration

The application might be vulnerable if the application is:

- Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords are still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, the latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
- The server does not send security headers or directives, or they are not set to secure values.
- The software is out of date or vulnerable.

Without a concerted, repeatable application security configuration process, systems are at a higher risk.

How to Prevent

Secure installation processes should be implemented, including:

- 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. 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.

6. A06:2021 – Vulnerable and Outdated Components

You are likely vulnerable:

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If the 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.
- 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. This commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.
- If you do not secure the components' configurations.

How to Prevent

There should be a patch management process in place to:

- 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.

7. A07:2021 – Identification and Authentication Failures

Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be authentication weaknesses if the application:

- 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 and forgot-password processes, such as "knowledge-based answers," which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords data stores.
- Has missing or ineffective multi-factor authentication.
- Exposes session identifier in the URL.
- Reuse session identifier after successful login.
- Does not correctly invalidate Session IDs. User sessions or authentication tokens (mainly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

How to Prevent

- 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.

8. **A08:2021 – Software and Data Integrity Failures**

Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application. Attackers could potentially upload their own updates to be distributed and run on all installations. Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.

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

9. **A09:2021 – Security Logging and Monitoring Failures**

Returning to the OWASP Top 10 2021, this category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time:

- Auditable events, such as logins, failed logins, and high-value transactions, are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts.
- The application cannot detect, escalate, or alert for active attacks in real-time or near real-time.

You are vulnerable to information leakage by making logging and alerting events visible to a user or an attacker.

How to Prevent

Developers should implement some or all the following controls, depending on the risk of the application:

- 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.

There are commercial and open-source application protection frameworks such as the OWASP ModSecurity Core Rule Set, and open-source log correlation software, such as the Elasticsearch, Logstash, Kibana (ELK) stack, that feature custom dashboards and alerting.

10. A10:2021 – Server-Side Request Forgery (SSRF)

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

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.

- 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,