OWASP Top 10 - 2021

Study online at https://quizlet.com/_foanhd

A02 Cryptographic Failures - Scenario 1	An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text.
A02 Crytographic Failures - Scenario 2	A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g., at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g., the recipient of a money transfer.
A02 Cryptographic Failures - Scenario 3	The password database uses unsalted or simple hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.
A01 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.
A01 Broken Access Control - Scenario 1	The application uses unverified data in a SQL call that is accessing account information: pstmt.setString(1, request.getParameter("acct")); ResultSet results = pstmt.executeQuery( ); An attacker simply modifies the browser's 'acct' parameter to send whatever account number they want. If not correctly verified, the attacker can access any user's account. https://example.com/app/accountInfo?acct=notmyacct
A01 Broken Access Control - Scenario 2	An attacker simply forces browses to target URLs. Admin rights are required for access to the admin page. https://example.com/app/getappInfo https://example.com/app/admin_getappInfo If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

A03 Injection - 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.
A03 Injection - Scenario 1	An application uses untrusted data in the construction of the following vulnerable SQL call: String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
A03 Injection - Scenario 2	Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)): Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'"); In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' UNION SLEEP(10);--. For example: http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);-- This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data or even invoke stored procedures.
A04 Insecure Design - 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
A04 Insecure Design - Attack Scenario 1	A credential recovery workflow might include "questions and answers," which is prohibited by NIST 800-63b, the OWASP ASVS, and the OWASP Top 10. Questions and answers cannot be trusted as evidence of identity as more than one person can know the answers, which is why they are prohibited. Such code should be removed and replaced with a more secure design.
A04 Insecure Design - Attack Scenario 2	A cinema chain allows group booking discounts and has a maximum of fifteen attendees before requiring a deposit. Attackers could threat model this flow and test if they could book six hundred

	seats and all cinemas at once in a few requests, causing a massive loss of income.
A04 Insecure Design - Attack Scenario 3	A retail chain's e-commerce website does not have protection against bots run by scalpers buying high-end video cards to resell auction websites. This creates terrible publicity for the video card makers and retail chain owners and enduring bad blood with enthusiasts who cannot obtain these cards at any price. Careful anti-bot design and domain logic rules, such as purchases made within a few seconds of availability, might identify inauthentic purchases and rejected such transactions.
A05 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 (see A06:2021-Vulnerable and Outdated Components).
A05 Security Misconfiguration - 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 (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.
A05 Security Misconfiguration - Scenario 1	The application server comes with sample applications not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. Suppose one of these applications is the admin console, and default accounts weren't changed. In that case, the attacker logs in with default passwords and takes over.
A05 Security Misconfiguration - Scenario 2	Directory listing is not disabled on the server. An attacker discovers they can simply list directories. The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a severe access control flaw in the application.

A05 Security Misconfiguration - Scenario 3	The application server's configuration allows detailed error messages, e.g., stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.
A05 Security Misconfiguration - Scenario 4	A cloud service provider (CSP) has default sharing permissions open to the Internet by other CSP users. This allows sensitive data stored within cloud storage to be accessed.
A06 Vulnerable and Outdated Components - 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.
A06 Vulnerable and Outdated Components	Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g., coding error) or intentional (e.g., a backdoor in a component). Some example exploitable component vulnerabilities discovered are: CVE-2017-5638, a Struts 2 remote code execution vulnerability that enables the execution of arbitrary code on the server, has been blamed for significant breaches. While the internet of things (IoT) is frequently difficult or impossible to patch, the importance of patching them can be great (e.g., biomedical devices). There are automated tools to help attackers find unpatched or misconfigured systems. For example, the Shodan IoT search engine can help you find devices that still suffer from Heartbleed vulnerability patched in April 2014.
A07 Identification and Authentication Failures - 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.
A07 Identification and Authentication Failures - Scenario 1	Credential stuffing, the use of lists of known passwords, is a common attack. Suppose an application does not implement automated threat or credential stuffing protection. In that case, the application can be used as a password oracle to determine if the credentials are valid.
A07 Identification and Authentication Failures - Scenario 2	Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered best practices, password rotation and complexity requirements encourage users to use and reuse weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.
A07 Identification and Authentication Failures - Scenario 3	Application session timeouts aren't set correctly. A user uses a public computer to access an application. Instead of selecting "logout," the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.
A08 Software and Data Integrity Failures - 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
A08 Software and Data Integrity Failures - Attack Scenario 1	Update without signing: Many home routers, set-top boxes, device firmware, and others do not verify updates via signed firmware. Unsigned firmware is a growing target for attackers and is expected to only get worse. This is a major concern as many times there is no mechanism to remediate other than to fix in a future version and wait for previous versions to age out.
A08 Software and Data Integrity Failures - Attack Scenario 2	SolarWinds malicious update: Nation-states have been known to attack update mechanisms, with a recent notable attack being the SolarWinds Orion attack. The company that develops the software had secure build and update integrity processes. Still, these were able to be subverted, and for several months, the firm distributed a highly targeted malicious update to more than 18,000 organizations, of which around 100 or so were affected. This is one of the most far-reaching and most significant breaches of this nature in history.
A08 Software and Data Integrity Failures - Attack Scenario 3	Insecure Deserialization: A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing the user state and passing it back and forth with each request. An attacker notices the "rO0" Java object signature (in base64) and uses the Java Serial Killer tool to gain remote code execution on the application server.

A09 Security Logging and Monitoring Failures - 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.
A09 Security Logging and Monitoring Failures - Scenario 1	A children's health plan provider's website operator couldn't detect a breach due to a lack of monitoring and logging. An external party informed the health plan provider that an attacker had accessed and modified thousands of sensitive health records of more than 3.5 million children. A post-incident review found that the website developers had not addressed significant vulnerabilities. As there was no logging or monitoring of the system, the data breach could have been in progress since 2013, a period of more than seven years.
A09 Security Logging and Monitoring Failures - Scenario 2	A major Indian airline had a data breach involving more than ten years' worth of personal data of millions of passengers, including passport and credit card data. The data breach occurred at a third-party cloud hosting provider, who notified the airline of the breach after some time.
A09 Security Logging and Monitoring Failures - Scenario 3	A major European airline suffered a GDPR reportable breach. The breach was reportedly caused by payment application security vulnerabilities exploited by attackers, who harvested more than 400,000 customer payment records. The airline was fined 20 million pounds as a result by the privacy regulator.
A10 Server Side Request Forgery - How to Prevent	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

A10 Server Side Request Forgery - Attack Scenario 1	Port scan internal servers - If the network architecture is unsegmented, attackers can map out internal networks and determine if ports are open or closed on internal servers from connection results or elapsed time to connect or reject SSRF payload connections.
A10 Server Side Request Forgery - Attack Scenario 2	Sensitive data exposure - Attackers can access local files or internal services to gain sensitive information such as file:///etc/passwd and http://localhost:28017/.
A10 Server Side Request Forgery - Attack Scenario 3	Access metadata storage of cloud services - Most cloud providers have metadata storage such as http://169.254.169.254/. An attacker can read the metadata to gain sensitive information.
A10 Server Side Request Forgery - Attack Scenario 4	Compromise internal services - The attacker can abuse internal services to conduct further attacks such as Remote Code Execution (RCE) or Denial of Service (DoS).
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - Example 1	my $dataPath = "/users/cwe/profiles";my $username = param("user");my $profilePath = $dataPath . "/" . $username;open(my $fh, "<", $profilePath) || ExitError("profile read error: $profilePath");print " \n";while (<$fh>) { print " $_ \n"; }print " \n";
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - Example 2	String filename = System.getProperty("com.domain.application.dictionaryFile");File dictionaryFile = new File(filename);
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - Example 3	my $Username = GetUntrustedInput();$Username =~ s/\.\.\///;my $filename = "/home/user/" . $Username;ReadAndSendFile($filename);

CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - Example 4	String path = getInputPath();if (path.startsWith("/safe_dir/")){
	File f = new File(path);f.delete()
	}

CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - Example 5	<form action="FileUploadServlet" method="post" enctype="multipart/form-data">Choose a file to upload:<input type="file" name="filename"/> <input type="submit" name="submit" value="Submit"/></form>
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - Example 6	import osimport sysdef main(): filename = sys.argv[1]path = os.path.join(os.getcwd(), filename)try:with open(path, 'r') as f:file_data = f.read()except FileNotFoundError as e:print("Error - file not found") main()
C7: Enforce Access Controls	Discretionary Access Control (DAC) is a means of restricting access to objects (e.g., files, data entities) based on the identity and need-to-know of subjects (e.g., users, processes) and/or groups to which the object belongs. Mandatory Access Control (MAC) is a means of restricting access to system resources based on the sensitivity (as represented by a label) of the information contained in the system resource and the formal authorization (i.e., clearance) of users to access information of such sensitivity. Role Based Access Control (RBAC) is a model for controlling access to resources where permitted actions on resources are identified with roles rather than with individual subject identities. Attribute Based Access Control (ABAC) will grant or deny user requests based on arbitrary attributes of the user and arbitrary attributes of the object, and environment conditions that may be globally recognized and more relevant to the policies at hand.
CWE-23: Relative Path Traversal (Example 1)	http://example.com.br/get-files.jsp?file=report.pdfhttp://example.com.br/get-page.php?home=aaa.htmlhttp://example.com.br/some-page.asp?page=index.html

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (Example 1)	my $username=param('username');my $password=param('password');if (IsValidUsername($username) == 1){
	if (IsValidPassword($username, $password) == 1){print "Login Successful";}else{print "Login Failed - incorrect password";}
	}else{
	print "Login Failed - unknown username";
	}

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (Example 2)	try {
	openDbConnection();
	}//print exception message that includes exception message and configuration file locationcatch (Exception $e) {
	echo 'Caught exception: ', $e->getMessage(), '\n';echo 'Check credentials in config file at: ', $Mysql_config_location, '\n';
	}

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (Example 3)	public BankAccount getUserBankAccount(String username, String accountNumber) {
	BankAccount userAccount = null;String query = null;try {if (isAuthorizedUser(username)) {query = "SELECT * FROM accounts WHERE owner = "+ username + " AND accountID = " + accountNumber;DatabaseManager dbManager = new DatabaseManager();Connection conn = dbManager.getConnection();Statement stmt = conn.createStatement();ResultSet queryResult = stmt.executeQuery(query);userAccount = (BankAccount)queryResult.getObject(accountNumber);}} catch (SQLException ex) {String logMessage = "Unable to retrieve account information from database,\nquery: " + query;Logger.getLogger(BankManager.class.getName()).log(Level.SEVERE, logMessage, ex);}return userAccount;
	}

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (Example 4)	locationClient = new LocationClient(this,
	this, this);locationClient.connect();currentUser.setLocation(locationClient.getLastLocation());...catch (Exception e) {
	AlertDialog.Builder builder = new AlertDialog.Builder(this);builder.setMessage("Sorry, this application has experienced an error.");AlertDialog alert = builder.create();alert.show();Log.e("ExampleActivity", "Caught exception: " + e + " While on User:" + User.toString());
	}

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (Example 5)	Warning: mysql_pconnect(): Access denied for user: 'root@localhost' (Using password: N1nj4) in /usr/local/www/wi-data/includes/database.inc on line 4
CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (Example 6)	Social Security Number: <%= ssn %></br>Credit Card Number: <%= ccn %>
CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (Example 7)	<% if (Boolean.getBoolean("debugEnabled")) { %>User account number: <%= acctNo %><%} %>
CWE-200: Exposure of Sensitive Information to an Unauthorized Actor (Example 8)	<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/> locationClient = new LocationClient(this, this, this);locationClient.connect();Location userCurrLocation;userCurrLocation = locationClient.getLastLocation();deriveStateFromCoords(userCurrLocation);
CWE-285: Improper Authorization (Example 1)	function runEmployeeQuery($dbName, $name){ mysql_select_db($dbName,$globalDbHandle) or die("Could not open Database".$dbName);//Use a prepared statement to avoid CWE-89$preparedStatement = $globalDbHandle->prepare('SELECT * FROM employees WHERE name = :name');$preparedStatement->execute(array(':name' => $name));return $preparedStatement->fetchAll();

	}/.../$employeeRecord = runEmployeeQuery('Employ-eeDB',$_GET['EmployeeName']);

CWE-285: Improper Authorization (Example 2)	sub DisplayPrivateMessage {
	my($id) = @_;my $Message = LookupMessageObject($id);print "From: " . encodeHTML($Message->{from}) . "
	\n";print "Subject: " . encodeHTML($Message->{subject}) . "\n";print "<hr>\n";print "Body: " . encodeHTML($Message->{body}) . "\n";
	}my $q = new CGI;# For purposes of this example, assume that CWE-309 and# CWE-523 do not apply.if (! Authentica-teUser($q->param('username'), $q->param('password'))) {
	ExitError("invalid username or password");
	}my $id = $q->param('id');DisplayPrivateMessage($id);

CWE-352: Cross-Site Request Forgery (Example 1)	<form action="/url/profile.php" method="post"><input type="text" name="firstname"/><input type="text" name="last-name"/> <input type="text" name="email"/><input type="submit" name="submit" value="Update"/></form>
CWE-359: Exposure of Private Personal Information to an Unauthorized Actor (Example 1)	pass = GetPassword();...dbmsLog.WriteLine(id + ":" + pass + ":" + type + ":" + tstamp);
CWE-359: Exposure of Private Personal Information to an Unauthorized Actor (Example 2)	<uses-permission android:name="android.permission.AC-CESS_FINE_LOCATION"/> locationClient = new LocationClient(this, this, this);loca-tionClient.connect();Location userCurrLocation;userCurrLocation = locationClient.getLastLocation();deriveStateFromCoords(user-CurrLocation);
CWE-359: Exposure of Private Personal Information to an Unauthorized Actor (Example 3)	In 2004, an employee at AOL sold approximately 92 million private customer e-mail addresses to a spammer marketing an offshore gambling web site [REF-338]. In response to such high-profile exploits, the collection and management of private data is becoming increasingly regulated.
CWE-377: Insecure Temporary File	if (tmpnam_r(filename)) { FILE* tmp = fopen(file-name,"wb+");while((recv(sock,recvbuf,DATA_SIZE, 0) > 0)&(amt!=0)) amt = fwrite(recvbuf,1,DATA_SIZE,tmp); }...
CWE-425: Direct Request (Forced Browsing)	If forced browsing is possible, an attacker may be able to directly access a sensitive page by entering a URL similar to the following. http://somesite.com/someapplication/admin.jsp
CWE-441 Unintended Proxy or Intermediary ('Confused Deputy')	The code in ring-3 (least trusted ring) of the microcontroller attempts to directly read the protected registers in IP core through MMIO transactions. However, this attempt is blocked due to the implemented access control. Now, the microcontroller configures the DMA core to transfer data from the protected registers to a memory region that it has access to. The DMA core, which is acting as an intermediary in this transaction, does not preserve the identity of the microcontroller and, instead, initiates a new transaction with its own identity. Since the DMA core has access, the transaction (and hence, the attack) is successful. The DMA core forwards this transaction with the identity of the code executing on the microcontroller, which is the original initiator of the end-to-end transaction. Now the transaction is blocked, as a result of forwarding the identity of the true initiator which lacks the permission to access the confidential MMIO mapped IP core.
CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere (Example 1)	char* path = getenv("PATH");...sprintf(stderr, "cannot find exe on path %s\n", path);
CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere (Example 2)	//assume getCurrentUser() returns a username that is guaranteed to be alphanumeric (avoiding CWE-78)$userName = getCurren-tUser();$command = 'ps aux | grep ' . $userName;system($com-mand);

CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere (Example 3)	try {
	...
	} catch (Exception e) {
	e.printStackTrace();
	}
	try {
	...
	} catch (Exception e) {
	Console.Writeline(e);
	}

CWE-497: Exposure of Sensitive System Information to an Unauthorized Control Sphere (Example 4)	string cs="database=northwind; server=mySQLServer...";SqlConnection conn=new SqlConnection(cs);...Console.Writeline(cs);
CWE-538: Insertion of Sensitive Information into Externally-Accessible File or Directory	logger.info("Username: " + usernme + ", CCN: " + ccn);
CWE-540: Inclusion of Sensitive Information in Source Code (Example 1)	database.inc <?php$dbName = 'usersDB';$dbPassword = 'skjdh#67nkjd3$3$';?> login.php <?phpinclude('database.inc');$db = connectToDB($dbName, $dbPassword);$db.authenticateUser($username, $password);?>
CWE-540: Inclusion of Sensitive Information in Source Code (Example 2)	<!-- FIXME: calling this with more than 30 args kills the JDBC server -->
CWE-552: Files or Directories Accessible to External Parties (Example 1)	az storage account update --name <storage-account> --resource-group <resource-group> --allow-blob-public-access true
CWE-552: Files or Directories Accessible to External Parties (Example 2)	gsutil iam get gs://BUCKET_NAME
CWE-566: Authorization Bypass Through User-Controlled SQL Primary Key	...conn = new SqlConnection(_ConnectionString);conn.Open();int16 id = System.Convert.ToInt16(invoiceID.Text);SqlCommand query = new SqlCommand("SELECT * FROM invoices WHERE id = @id", conn);query.Parameters.AddWithValue("@id", id);SqlDataReader objReader = objCommand.ExecuteReader();...
CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (Example 1)	$redirect_url = $_GET['url'];header("Location: " . $redirect_url);

CWE-601: URL Redirection to Untrusted Site ('Open Redirect')(Example 2)	public class RedirectServlet extends HttpServlet {
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {String query = request.getQueryString();if (query.contains("url")) {String url = request.getParameter("url");response.sendRedirect(url);}}
	}

CWE-639: Authorization Bypass Through User-Controlled Key	...conn = new SqlConnection(_ConnectionString);conn.Open();int16 id = System.Convert.ToInt16(invoiceID.Text);SqlCommand query = new SqlCommand("SELECT * FROM invoices WHERE id = @id", conn);query.Parameters.AddWithValue("@id", id);SqlDataReader objReader = objCommand.ExecuteReader();...
CWE-862: Missing Authorization (Example 1)	function runEmployeeQuery($dbName, $name){ mysql_select_db($dbName,$globalDbHandle) or die("Could not open Database".$dbName);//Use a prepared statement to avoid CWE-89$preparedStatement = $globalDbHandle->prepare('SELECT * FROM employees WHERE name = :name');$preparedStatement->execute(array(':name' => $name));return $preparedStatement->fetchAll();

	}/.../$employeeRecord = runEmployeeQuery('EmployeeDB',$_GET['EmployeeName']);
CWE-862: Missing Authorization (Example 2)	sub DisplayPrivateMessage {my($id) = @_;my $Message = LookupMessageObject($id);print "From: " . encodeHTML($Message->{from}) . " \n";print "Subject: " . encodeHTML($Message->{subject}) . "\n";print "<hr>\n";print "Body: " . encodeHTML($Message->{body}) . "\n";}my $q = new CGI;# For purposes of this example, assume that CWE-309 and# CWE-523 do not apply.if (! AuthenticateUser($q->param('username'), $q->param('password'))) {ExitError("invalid username or password");}my $id = $q->param('id');DisplayPrivateMessage($id);

CWE-863: Incorrect Authorization	$role = $_COOKIES['role'];if (!$role) {
	$role = getRole('user');if ($role) {// save the cookie to send out in future responsessetcookie("role", $role, time()+60*60*2);}else{ShowLoginScreen();die("\n");}
	}if ($role == 'Reader') {
	DisplayMedicalHistory($_POST['patient_ID']);
	}else{
	die("You are not Authorized to view this record\n");
	}

CWE-1275: Sensitive Cookie with Improper SameSite Attribute	let sessionId = generateSessionId()let cookieOptions = { domain: 'example.com' }response.cookie('sessionid', sessionId, cookieOptions)
CWE-261: Weak Encoding for Password (Example 1)	...Properties prop = new Properties();prop.load(new FileInputStream("config.properties"));String password = Base64.decode(prop.getProperty("password"));DriverManager.getConnection(url, usr, password);...
CWE-261: Weak Encoding for Password (Example 2)	...string value = regKey.GetValue(passKey).ToString();byte[] decVal = Convert.FromBase64String(value);NetworkCredential netCred = newNetworkCredential(username,decVal.toString(),domain);...

CWE-296: Improper Following of a Certificate's Chain of Trust	if ((cert = SSL_get_peer_certificate(ssl)) && host)
	foo=SSL_get_verify_result(ssl);
	if ((X509_V_OK==foo) ||
	X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo))
	// certificate looks good, host can be trusted

CWE-319: Cleartext Transmission of Sensitive Information (Example 1)	try {
	URL u = new URL("http://www.secret.example.org/");HttpURLConnection hu = (HttpURLConnection) u.openConnection();hu.setRequestMethod("PUT");hu.connect();OutputStream os = hu.getOutputStream();hu.disconnect();
	}catch (IOException e) {
	//...
	}

CWE-319: Cleartext Transmission of Sensitive Information (Example 2)	In 2022, the OT:ICEFALL study examined products by 10 different Operational Technology (OT) vendors. The researchers reported 56 vulnerabilities and said that the products were "insecure by design" [REF-1283]. If exploited, these vulnerabilities often allowed adversaries to change how the products operated, ranging from denial of service to changing the code that the products executed. Since these products were often used in industries such as power, electrical, water, and others, there could even be safety implications. Multiple vendors used cleartext transmission of sensitive information in their OT products.
CWE-319: Cleartext Transmission of Sensitive Information (Example 3)	A TAP accessible register is read/written by a JTAG based tool, for internal use by authorized users. However, an adversary can connect a probing device and collect the values from the unencrypted

	channel connecting the JTAG interface to the authorized user, if no additional protections are employed.
CWE-319: Cleartext Transmission of Sensitive Information (Example 4)	az storage account show -g {ResourceGroupName} -n {StorageAccountName}

CWE-321: Use of Hard-coded Cryptographic Key (Example 1)	int VerifyAdmin(char *password) {
	if (strcmp(password,"68af404b513073584c4b6f22b6c63e6b")) {printf("Incorrect Password!\n");return(0);}printf("Entering Diagnostic Mode...\n");return(1);
	}
	
	public boolean VerifyAdmin(String password) {
	if (password.equals("68af404b513073584c4b6f22b6c63e6b")) {System.out.println("Entering Diagnostic Mode...");return true;}System.out.println("Incorrect Password!");return false;
	
	int VerifyAdmin(String password) {
	if (password.Equals("68af404b513073584c4b6f22b6c63e6b")) {Console.WriteLine("Entering Diagnostic Mode...");return(1);}Console.WriteLine("Incorrect Password!");return(0);
	}

CWE-321: Use of Hard-coded Cryptographic Key (Example 2)	In 2022, the OT:ICEFALL study examined products by 10 different Operational Technology (OT) vendors. The researchers reported 56 vulnerabilities and said that the products were "insecure by design" [REF-1283]. If exploited, these vulnerabilities often allowed adversaries to change how the products operated, ranging from denial of service to changing the code that the products executed. Since these products were often used in industries such as power, electrical, water, and others, there could even be safety implications. Multiple vendors used hard-coded keys for critical functionality in their OT products.

CWE-323: Reusing a Nonce, Key Pair in Encryption (Example 1)	void encryptAndSendPassword(char *password){
	char *nonce = "bad";...char *data = (unsigned char*)malloc(20);int para_size = strlen(nonce) + strlen(password);char *paragraph = (char*)malloc(para_size);SHA1((const unsigned char*)paragraph,parsize,(unsigned char*)data);sendEncryptedData(data)
	}

CWE-323: Reusing a Nonce, Key Pair in Encryption (Example 2)	String command = new String("some command to execute");MessageDigest nonce = MessageDigest.getInstance("SHA");nonce.update(String.valueOf("bad nonce"));byte[] nonce = nonce.digest();MessageDigest password = MessageDigest.getInstance("SHA");password.update(nonce + "secretPassword");byte[] digest = password.digest();sendCommand(digest, command)

CWE-324: Use of a Key Past its Expiration Date	if (cert = SSL_get_peer_certificate(ssl)) {
	foo=SSL_get_verify_result(ssl);if ((X509_V_OK==foo) || (X509_V_ERRCERT_NOT_YET_VALID==foo))//do stuff
	}

CWE-325: Missing Cryptographic Step	logic [511:0] bigData;...hmac hmac(.clk_i(clk_i),.rst_ni(rst_ni && ~rst_4),.init_i(startHash && ~startHash_r),.key_i(key),.ikey_hash_i(ikey_hash),.okey_hash_i(okey_ pass_i(key_hash_bypass),.message_i(bigData),.hash_o(hash),.ready_o(ready),.hash_valid_o(hashValid)
CWE-327: Use of a Broken or Risky Cryptographic Algorithm (Example 1)	EVP_des_ecb(); Cipher des=Cipher.getInstance("DES...");des.initEncrypt(key2); function encryptPassword($password){ $iv_size = mcrypt_get_iv_size(MCRYPT_DES, MCRYPT_MODE_ECB);$iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);$key = "This is a password encryp-

	tion key";$encryptedPassword = mcrypt_encrypt(MCRYPT_DES, $key, $password, MCRYPT_MODE_ECB, $iv);return $encrypt-edPassword; }
CWE-327: Use of a Broken or Risky Cryptographic Algorithm (Example 2)	The manufacturer chooses a SHA1 hardware accelerator for to implement the scheme because it already has a working SHA1 Intellectual Property (IP) that the manufacturer had created and used earlier, so this reuse of IP saves design cost.

CWE-328: Use of Weak Hash (Example 1)	unsigned char *check_passwd(char *plaintext) {
	ctext = simple_digest("sha1",plaintext,strlen(plaintext), ...);//Login if hash matches stored hashif (equal(ctext, secret_password())) {login_user();}
	}
	
	String plainText = new String(plainTextIn);MessageDigest encer = MessageDigest.getInstance("SHA");encer.update(plain-TextIn);byte[] digest = password.digest();//Login if hash matches stored hashif (equal(digest,secret_password())) {
	login_user();
	}

CWE-328: Use of Weak Hash (Example 2)	...logic [31:0] data_d, data_qlogic [512-1:0] pass_data;... Write: begin...if (pass_mode) beginpass_data = { {60{8'h00}}, data_d};state_d = PassChk;pass_mode = 1'b0;...end ...

CWE-329: Generation of Predictable IV with CBC Mode	EVP_CIPHER_CTX ctx;char key[EVP_MAX_KEY_LENGTH];char iv[EVP_MAX_IV_LENGTH];RAND_bytes(key, b);memset(iv,0,EVP_MAX_IV_LENGTH);EVP_Encryp-tInit(&ctx,EVP_bf_cbc(), key,iv);
	
	public class SymmetricCipherTest {
	public static void main() {byte[] text ="Secret".getBytes();byte[] iv ={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};KeyGenera-tor kg = KeyGenerator.getInstance("DES");kg.init(56);Se-cretKey key = kg.generateKey();Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");IvParameter-Spec ips = new IvParameterSpec(iv);cipher.init(Cipher.EN-CRYPT_MODE, key, ips);return cipher.doFinal(inpBytes);}
	}

CWE-330: Use of Insufficiently Random Values (Example 1)	function generateSessionID($userID){
	srand($userID);return rand();
	}

CWE-330: Use of Insufficiently Random Values (Example 2)	String GenerateReceiptURL(String baseUrl) {
	Random ranGen = new Random();ranGen.setSeed((new Date()).getTime());return(baseUrl + ranGen.nextInt(400000000) + ".html");
	}

CWE-331: Insufficient Entropy (Example 1)	function generateSessionID($userID){
	srand($userID);return rand();
	}

CWE-331: Insufficient Entropy (Example 2)	String GenerateReceiptURL(String baseUrl) {
	Random ranGen = new Random();ranGen.setSeed((new Date()).getTime());return(baseUrl + ranGen.nextInt(400000000) + ".html");
	}

CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)(Example 1)	private static final long SEED = 1234567890;public int gener-ateAccountID() {
	Random random = new Random(SEED);return random.nextInt();
	}

CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)(Example 2)	Random random = new Random(System.currentTimeMillis());int accountID = random.nextInt(); srand(time());int randNum = rand();
CWE-335: Incorrect Usage of Seeds in Pseudo-Random Number Generator (PRNG)(Example 3)	# getting 2 bytes of randomness for the seeding the PRNGseed = os.urandom(2)random.seed(a=seed)key = random.getrandbits(128)

CWE-336: Same Seed in Pseudo-Random Number Generator (PRNG)(Example 1)	private static final long SEED = 1234567890;public int generateAccountID() {
	Random random = new Random(SEED);return random.nextInt();
	}

CWE-336: Same Seed in Psuedo-Random Number Generator (PRNG) (Example 2)	function generateSessionID($userID){
	srand($userID);return rand();
	}

CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG)	Random random = new Random(System.currentTimeMillis());int accountID = random.nextInt(); srand(time());int randNum = rand();
CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)	Random random = new Random(System.currentTimeMillis());int accountID = random.nextInt(); srand(time());int randNum = rand();

CWE-340: Generation of Predictable Numbers or Identifiers	function generateSessionID($userID){
	srand($userID);return rand();
	}

CWE-347: Improper Verification of Cryptographic Signature	File f = new File(downloadedFilePath);JarFile jf = new JarFile(f);

CWE-759: Use of a One-Way Hash without a Salt (Example 1)	unsigned char *check_passwd(char *plaintext) {
	ctext = simple_digest("sha1",plaintext,strlen(plaintext), ...);//Login if hash matches stored hashif (equal(ctext, secret_password())) {login_user();}
	}
	
	String plainText = new String(plainTextIn);MessageDigest encer = MessageDigest.getInstance("SHA");encer.update(plainTextIn);byte[] digest = password.digest();//Login if hash matches stored hashif (equal(digest,secret_password())) {
	login_user();
	}

CWE-759: Use of a One-Way Hash without a Salt (Example 2)	def storePassword(userName,Password): hasher = hashlib.new('md5')hasher.update(Password)hashedPassword = hasher.digest()# UpdateUserLogin returns True on success, False otherwisereturn updateUserLogin(userName,hashedPassword)

CWE-780: Use of RSA Algorithm without OAEP	public Cipher getRSACipher() {
	Cipher rsa = null;try {rsa = javax.crypto.Cipher.getInstance("RSA/NONE/NoPadding");}catch (java.security.NoSuchAlgorithmException e) {log("this should never happen", e);}catch (javax.crypto.NoSuchPaddingException e) {log("this should never happen", e);}return rsa;
	}

CWE-916: Use of Password Hash With Insufficient Computational Effort	def storePassword(userName,Password): hasher = hashlib.new('md5')hasher.update(Password)hashedPassword = hasher.digest()# UpdateUserLogin returns True on success, False otherwisereturn updateUserLogin(userName,hashedPassword)

C8: Protect Data Everywhere - Data Classification	It's critical to classify data in your system and determine which level of sensitivity each piece of data belongs to. Each data category can then be mapped to protection rules necessary for each level of sensitivity. For example, public marketing information that is not sensitive may be categorized as public data which is okay to place on the public website. Credit card numbers may be classified as private user data which may need to be encrypted while stored or in transit.
C8: Protect Data Everywhere - Encrypting Data in Transit	When transmitting sensitive data over any network, end-to-end communications security (or encryption-in-transit) of some kind should be considered. TLS is by far the most common and widely supported cryptographic protocol for communications security. It is used by many types of applications (web, web service, mobile) to communicate over a network in a secure fashion. TLS must be properly configured in a variety of ways in order to properly defend secure communications. The primary benefit of transport layer security is the protection of web application data from unauthorized disclosure and modification when it is transmitted between clients (web browsers) and the web application server, and between the web application server and back end and other non-browser based enterprise components.
Mobile Application: Secure Local Storage	Mobile applications are at particular risk of data leakage because mobile devices are regularly lost or stolen yet contain sensitive data. As a general rule, only the minimum data required should be stored on the mobile device. But if you must store sensitive data on a mobile device, then sensitive data should be stored within each mobile operating systems specific data storage directory. On Android this will be the Android keystore and on iOS this will be the iOS keychain.
Secret Life Cycle	Secret keys can be used in a number of sensitive functions. For example, they can be used to sign JWTs, encrypt credit cards, sign hash, provide various forms of authentication and more. In managing keys, a number of precautious should be adhered including but not limited to the following: Ensure that any secret key is protected from unauthorized access Store keys in a proper secrets vault as described in Application Secrets Management Use independent keys when multiple keys are required Build support for changing algorithms and keys when needed Build application features to handle a key rotation
Secret Datatype	When an immutable datatype such as string is used to store secrets, secrets can remain plaintext in the memory for a long time. Even if you try to nullify the string value, it still remains in the memory. string is an immutable type and cannot be changed. When you modify a string (try to overwrite it), a new copy of it is created. This means another copy of the unprotected secret will remain in the memory. Furthermore, there is no guarantee when garbage collector is going to clean up the secret. This increases exposure of plaintext secrets in the memory. If secrets remain unprotected in the memory, they can get disclosed on the disk or external log aggregators through a number of scenarios: server crash logs, caching, serialisation or memory paging. A safe way to handle secret is by using Read Once pattern. Read once is a defensive design pattern where a value can be only accessed once. After the first read, value is cleared from memory, and subsequent access is not possible. For an example implementation see this post.

Application Secrets Management	Applications contain numerous "secrets" that are needed for security operations. These include certificates, SQL connection passwords, third party service account credentials, passwords, SSH keys, encryption keys and more. The unauthorized disclosure or modification of these secrets could lead to complete system compromise. In managing application secrets, consider the following. Don't store secrets in code, config files or pass them through environment variables. Use tools like GitRob or TruffleHog to scan code repositories for secrets. Keep keys and your other application-level secrets in a secrets vault like KeyWhiz or Hashicorp's Vault project or Amazon KMS to provide secure storage and access to application-level secrets at run-time.