

Experiment – 2

Aim: To learn Case study for SDLC.

Prerequisite: Cyber security, software engineering

Requirements: PC and Internet

Theory:

What Is Secure SDLC and Why Is Important ?

Security System Development Life Cycle (SecSDLC) is defined as the set of procedures that are executed in a sequence in the software development cycle (SDLC). It is designed such that it can help developers to create software and applications in a way that reduces the security risks at later stages significantly from the start. The Security System Development Life Cycle (SecSDLC) is similar to Software Development Life Cycle (SDLC), but they differ in terms of the activities that are carried out in each phase of the cycle. SecSDLC eliminates security vulnerabilities. Its process involves identification of certain threats and the risks they impose on a system as well as the needed implementation of security controls to counter, remove and manage the risks involved. Whereas, in the SDLC process, the focus is mainly on the designs and implementations of an information system.

Phases involved in SecSDLC are:

- **System Investigation:** This process is started by the officials/directives working at the top level management in the organization. The objectives and goals of the project are considered priorly in order to execute this process. An Information Security Policy is defined which contains the descriptions of security applications and programs installed along with their implementations in organization's system.
- **System Analysis:** In this phase, detailed document analysis of the documents from the System Investigation phase are done. Already existing security policies, applications and software are analyzed in order to check for different flaws and vulnerabilities in the system. Upcoming threat possibilities are also analyzed. Risk management comes under this process only.
- **Logical Design:** The Logical Design phase deals with the development of tools and following blueprints that are involved in various information security policies, their applications and software. Backup and recovery policies are also drafted in order to prevent future losses. In case of any disaster, the steps to take in business are also planned. The decision to outsource the company project is decided in this phase. It is analyzed whether the project can be completed in the company itself or it needs to be sent to another company for the specific task.
- **Physical Design:** The technical teams acquire the tools and blueprints needed for the implementation of the software and application of the system security. During this phase, different solutions are investigated for any unforeseen issues which may be encountered in the future. They are analyzed and written down in order to cover most of the vulnerabilities that were missed during the analysis phase.
- **Implementation:** The solution decided in earlier phases is made final whether the project is in-house or outsourced. The proper documentation is provided of the product in order to meet the requirements specified for the project to be met. Implementation and integration process of

the project are carried out with the help of various teams aggressively testing whether the product meets the system requirements specified in the system documentation.

- **Maintenance:** After the implementation of the security program it must be ensured that it is functioning properly and is managed accordingly. The security program must be kept up to date accordingly in order to counter new threats that can be left unseen at the time of design.

1. Laboratory Exercise

i. Procedure

- i. Study any of the case study from references and the difference between software development life cycle and security development life cycle

2. Post-Experiments Exercise

A. Extended Theory:

1. Describe how secure coding can be incorporated into the software development process.
2. List the major types of coding errors and their root cause.
3. Describe good software development practices and explain how they impact application security.

.

B. Questions:

1. List and discuss Secure SDLC Best Practices

C. Conclusion:

1. Write what was performed in the experiment.
2. Write the significance of the topic studied in the experiment.

D. References:

1. Case study 1:
<https://quod.lib.umich.edu/j/jsais/11880084.0001.103/--case-study-of-the-application-of-the-systems-development?rgn=main;view=fulltext>
2. <https://snyk.io/learn/secure-sdlc/>

Experiment – 3

Aim: Design Threat Model for configure HTTP and HTTPS traffic within organisation network.

Prerequisite: Cyber security, software engineering

Requirements: Windows OS, Threat Modeling Tool 2016

Theory:

Threat modeling is an exercise designed to help an organization identify potential threats and cyber security risks within their organization and systems. This is an essential first step toward designing defenses and solutions to help eliminate or reduce these risks. Threat modeling is a four-step process:

1. Create the design
2. Apply zones of trust
3. Discover threats with STRIDE
4. Explore mitigations and controls

The table below outlines the nodes and connections in the scenario used in this walkthrough.

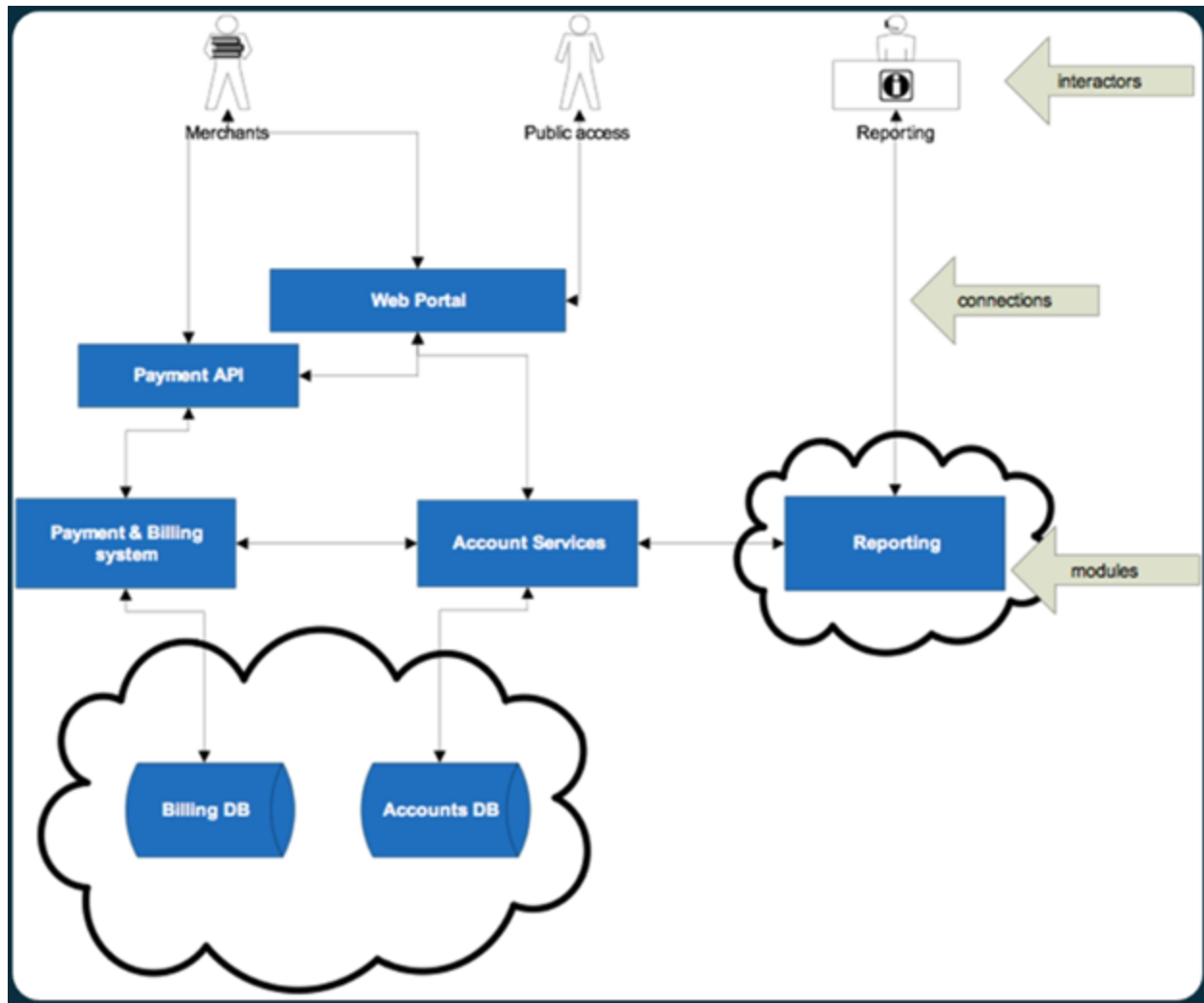
Node	Type	Connections (flows)	Cloud?
Merchants	Interactors	Web portal, via website Payment API, via XML	
Public access	Interactors	Web portal, via website	
Reporting users	Interactors	Reporting, via website	
Web portal	Website	Merchants, via website Public access, via website Payment API, via XML Account services	
Payment API	Web API	Merchants, via XML Web portal, via XML Payment & billing system, via XML	
Payment & billing system	Process	Payment API, via XML Account services, via XML Billing DB, via SQL	
Account services	Internal API	Payment & billing system Web portal, via XML Accounts DB, via SQL Reporting, via JSON	
Reporting	Third-party website	Account services, via JSON Reporting users, via website	Yes
Billing DB	Datastore	Payment & billing system, via SQL	Yes
Accounts DB	Datastore	Account services, via SQL	Yes

Learn Threat Modeling

Get hands-on experience with six threat modeling courses covering defense-in-depth, frameworks like STRIDE and Rapid Threat Model Prototyping (RTMP), agile architecture and more.

1. Design the threat model

The first step in the threat modeling process is designing the threat model. This stage involves diagramming the various interactors and nodes within the modeled ecosystem and identifying the links and data flows between these various parties and systems.



The image above shows a sample diagram derived from the table shown previously. This diagram has three types of content:

- **Interactors:** These are the people who will interact with the systems. These interactors may have different duties or user stories and each is included separately within the diagram.
- **Modules:** A module is a component of the system under test. This can include applications (like a web portal or payment API) and other endpoints or systems within the environment.
- **Connections:** Connections are links between interactors and modules. This includes interactor-module flows (like the use of a web portal) and module-module flows (such as an application querying a database).

Developing this diagram is a crucial first step in the threat modeling process. It provides a visual representation of the system being assessed and the trust relationships between various actors and systems.

In this exercise, we'll be using the STRIDE threat model, which was created by Microsoft

employees. This threat model labels threats based upon their goals and impacts to the target:

- **Spoofing:** Masquerading as another user or system.
- **Tampering:** Modifying data on a system or network.
- **Repudiation:** Denying that you did something that you did.
- **Information disclosure:** Providing unauthorized access to sensitive data.
- **Denial of Service:** Degrading or destroying a system's ability to provide services.
- **Elevation of privileges:** Gaining the ability to do something that you are not authorized to do.

These different labels can be applied based upon the features of the threat diagram:

- **Spoofing:** Nodes where an interactor labeled “not in control of system” connects to any other node.
- **Tampering:** Flows between a less critical and more critical interactor or module.
- **Repudiation:** Any node where both spoofing and tampering are applicable.
- **Information disclosure:** Any flows from a more critical to a less critical node.
- **Denial of Service:** Any nodes or flows where an interactor labeled “not in control of system” connects to any other node.
- **Elevation of privilege:** Any node that is connected to a less-critical node.

Three of the STRIDE threat categories apply to network flows:

- **Tampering:** Less-critical to more-critical
- **Information disclosure:** More-critical to less-critical
- **Denial of Service:** “Not in control of system” to any other

In this stage of the exercise, the goal is to identify the OWASP vulnerabilities that are most applicable to each of the STRIDE threats labeled on the threat diagram. Use the following labels for selected mitigations recommended by OWASP:

1. Injection

- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).
- 1.2 Use positive or “whitelist” 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.

2. Broken Authentication

- Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force and stolen credential reuse attacks.
- Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle and absolute timeouts.

3. Sensitive Data Exposure

- Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
- Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2.

4. XML External Entities

- Whenever possible, use less complex data formats such as JSON and avoid serialization of sensitive data.
- Verify that XML or XSL file upload functionality validates incoming XML using XSD validation or similar.

5. Broken Access Control

- With the exception of public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing CORS usage.

6. Security Misconfiguration

- A segmented application architecture that provides effective, secure separation between components or tenants, with segmentation, containerization or cloud security groups (ACLs).

7. Cross-Site Scripting (XSS)

- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS or URL) will resolve Reflected and Stored XSS vulnerabilities. The OWASP Cheat Sheet 'XSS Prevention' has details on the required data escaping techniques.

8. Insecure Deserialization

- Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering.

9. Using Components with Known Vulnerabilities

- Only obtain components from official sources over secure links. Prefer signed packages to reduce the chance of including a modified, malicious component.

10. Insufficient Logging & Monitoring

- Ensure that logs are generated in a format that can be easily consumed by centralized log management solutions.
- Ensure high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.

3. Laboratory Exercise

- ii. Design threat model for HTTP and HTTPs communication for threat detection.
- iii. Generate threat detection Report

Output:

Attach threat detection Report

Experiment – 4

Aim: Study of SAST Tools - Netsparker and VirusTotal

Prerequisite: Cyber security, software engineering

Requirements: Windows OS, Netsparker and VirusTotal

Theory:

Netsparker helps web application developers or penetration testers to secure web applications .With preset

scan profiles, you can quickly start scanning your applications and get instant reports. Netsparker, Next Generation Web Application Security Scanner. Netsparker is a powerful web application security scanner, which can crawl, attack and identify vulnerabilities in all types of web application whatever platform and technology it is built on. It can identify web application vulnerabilities like SQL Injection, Cross-site Scripting (XSS), Remote Code Execution and many more with easy to use and intuitive user interface.

During the startup of Netsparker, it scans for C# code files (*.cs) in "ReportTemplates" directory located under Netsparker's installation directory. Every identified file will be visible in the "Reporting" menu as a custom report.

This report will generate an XML file which includes:

- All vulnerabilities
- Vulnerable Parameter and type (GET/POST)
- Vulnerability Details
- Confirmation Status
- Extra exploitation data
- Scan time
- Vulnerability severity etc.

Command Line Arguments:

/a, /auto	When other parameters are given correctly, scan is carried out, report is saved and the program is closed.
/p, /profile	Name of the profile to be used during the scan. If not specified, the preset profile will be used.
/u, /url	Address of the website to be scanned. If the profile file includes another website address, the address specified with this parameter will be taken into consideration. If two different URLs are specified in the profile and within this parameter, the one given with this parameter will be taken into consideration.
/pr, /proxy	Proxy server address. If the profile file includes another proxy server address, the address specified with this parameter will be taken into consideration. A valid proxy server address should be as follows: http://user:password@proxy.address/ If a user name and password are required for logging on the proxy server, these should be given in the shown format.
/r, /report	File path the report will be saved. It should be used with the "-a" parameter. The full physical file path can be given; if only the file name is given, the created report will be saved into the folder the command is run.
/rf, /reportformat	File format of the created report. If not specified, the report is created in "pdf" format; rtf, pdf, text, csv, xls or html formats are also supported.
/rt, /reporttemplate	Type of the created report. If not specified, first type in the list will be valid.

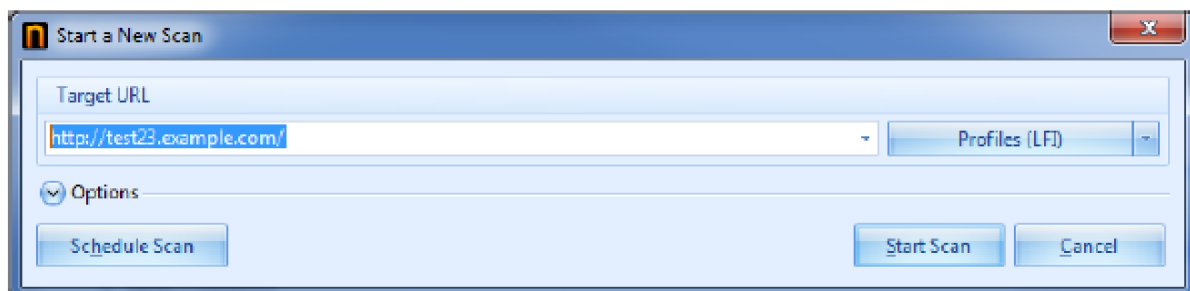
Sample Usages:

Scan `http://test23.example.com` and save the report to `C:\reports\report.pdf`.

Netsparker /a /url `http://test23.example.com` /rf pdf /r `C:\reports\scanreport.pdf`

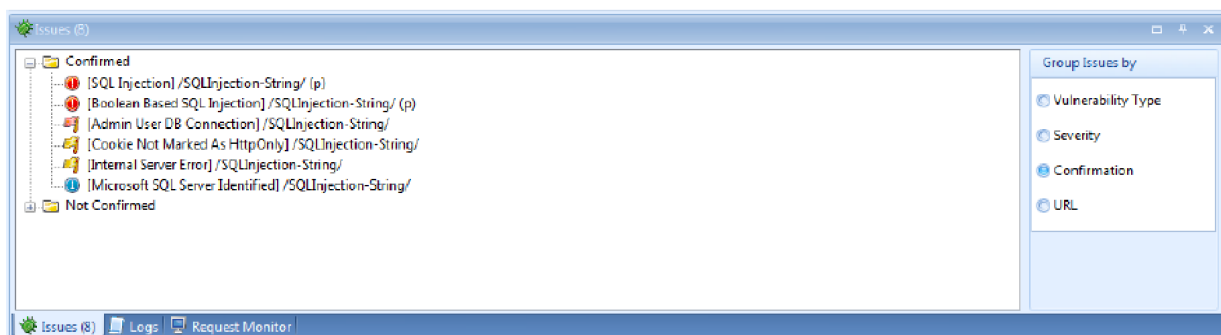
Launch a new scan parameter with a custom profile and URL:

Netsparker /url `http://test23.example.com` /p LFI



Exploiting SQL Injections:

You can exploit the Error Based or Boolean Based SQL Injection vulnerabilities identified in the web application and run custom SQL queries in the application's database via Netsparker's SQL Injection panel.



From the issues in the Issues panel choose a confirmed “SQL Injection” or “Boolean Based SQL Injection” issue, click the Execute SQL Commands button and the SQL Injection panel will appear where you can run custom SQL queries.

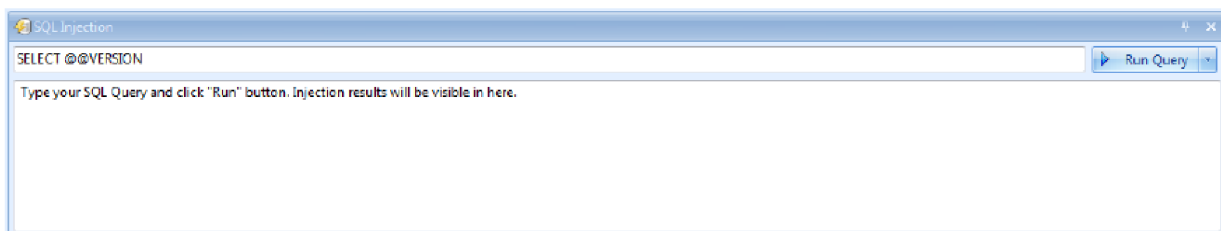


Fig: Running Custom SQL Queries in SQL Injection Panel

Afterwards, you can type an SQL query and run the query with the **Run Query** button. Response of the query will be displayed in the panel.

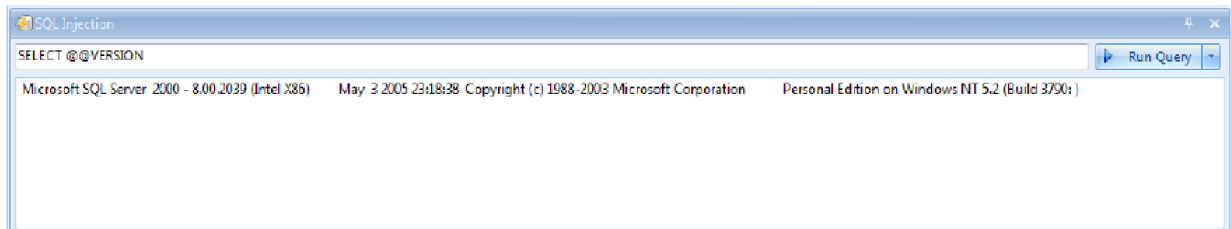


Fig:Sample Custom SQL Query Output

VirusTotal:

VirusTotal is analyzes suspicious files, URLs, domains and IP addresses to detect malware and other types of threats, and automatically shares them with the security community. To view VirusTotal reports, you'll be submitting file attachment hashes, IP addresses, or domains to VirusTotal.



Community Score

✓ No security vendors flagged this URL as malicious



<http://www.way2sms.com/>
www.way2sms.com

200
Status

2022-09-21 09:45:59 UTC
12 days ago



DETECTION

DETAILS

LINKS

COMMUNITY

Security Vendors' Analysis

Abusix	✓ Clean	Acronis	✓ Clean
ADMINUSLabs	✓ Clean	AICC (MONITORAPP)	✓ Clean
AlienVault	✓ Clean	alphaMountain.ai	✓ Clean
Antiy-AVL	✓ Clean	Artists Against 419	✓ Clean
Avira	✓ Clean	BADWARE.INFO	✓ Clean
benkow.cc	✓ Clean	Bfore AI PreCrime	✓ Clean

Experiment – 5

Aim: Study and implement at least any 5 methodologies of OWASP

Prerequisite: Cyber security, software engineering

Requirements: Windows OS, OWASP TOP 10 tools

Theory:

Testing for OWASP vulnerabilities is a crucial part of secure application development. The sheer number of risks and potential fixes can seem overwhelming but are easy to manage if you follow a few simple steps:

- Build security into your development process, rather than making it an afterthought
- Test your code against security standards repeatedly throughout development
- Use IDE and CI Pipeline integrations to automate testing
- Identify known vulnerabilities in third-party code to ensure your program does not rely on insecure dependencies

OWASP Top 10 Vulnerabilities:

Injection

Injection occurs when an attacker exploits insecure code to insert (or inject) their own code into a program. Because the program is unable to determine code inserted in this way from its own code, attackers are able to use injection attacks to access secure areas and confidential information as though they are trusted users. Examples of injection include SQL injections, command injections, CRLF injections, and LDAP injections.

Application security testing can reveal injection flaws and suggest remediation techniques such as stripping special characters from user input or writing parameterized SQL queries.

2. Broken Authentication

Incorrectly implemented authentication and session management calls can be a huge security risk. If attackers notice these vulnerabilities, they may be able to easily assume legitimate users' identities.

Multifactor authentication is one way to mitigate broken authentication. Implement DAST and SCA scans to detect and remove issues with implementation errors before code is deployed.

3. Sensitive Data Exposure

APIs, which allow developers to connect their application to third-party services like Google Maps, are great time-savers. However, some APIs rely on insecure data transmission methods, which attackers can exploit to gain access to usernames, passwords, and other sensitive information.

Data encryption, tokenization, proper key management, and disabling response caching can all help reduce the risk of sensitive data exposure.

4. XML External Entities

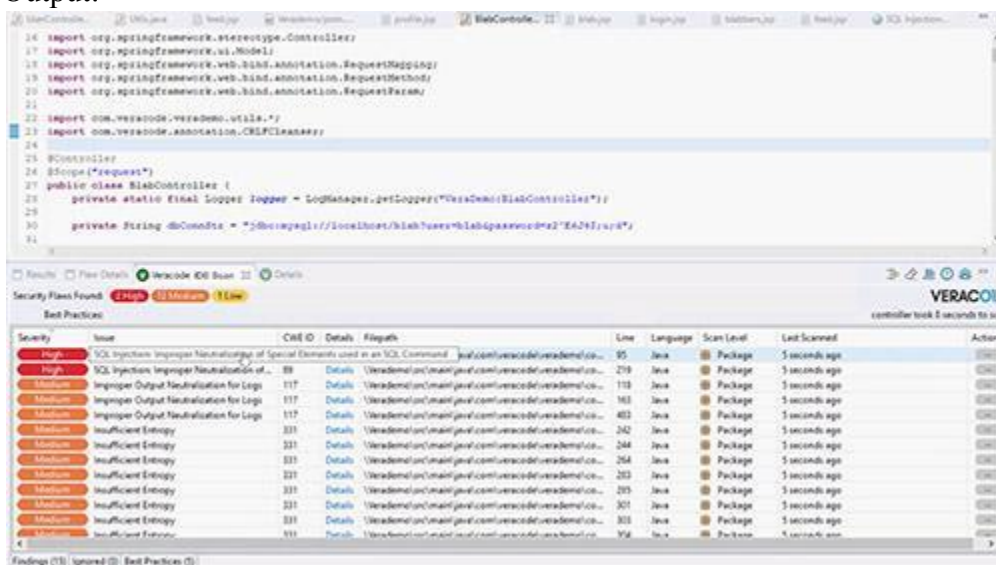
This risk occurs when attackers are able to upload or include hostile XML content due to insecure code, integrations, or dependencies. An SCA scan can find risks in third-party components with known vulnerabilities and will warn you about them. Disabling XML external entity processing also reduces the likelihood of an XML entity attack.

5. Broken Access Control

If authentication and access restriction are not properly implemented, it's easy for attackers to take whatever they want. With broken access control flaws, unauthenticated or unauthorized users may have access to sensitive files and systems, or even user privilege settings.

Configuration errors and insecure access control practices are hard to detect as automated processes cannot always test for them. Penetration testing can detect missing authentication, but other methods must be used to determine configuration problems. Weak access controls and issues with credentials management are preventable with secure coding practices, as well as preventative measures like locking down administrative accounts and controls and using multi-factor authentication.

Output:



The screenshot displays the Veracode IDE interface with a Java file open in the editor. The file contains code for a Spring MVC controller. Below the editor, the 'Results' tab is active, showing a list of security issues found during the scan. The issues are categorized by severity (High, Medium, Low) and include details such as the issue name, CWE ID, file path, line number, language, scan level, and last scanned date. The Veracode logo and 'controller took 2 seconds to scan' are visible in the bottom right corner of the results panel.

Severity	Issue	CWE ID	Details	Filepath	Line	Language	Scan Level	Last Scanned	Action
High	SQL Injection: Improper Neutralization of Special Elements used in an SQL Command	89	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	95	Java	Package	5 seconds ago	[Icon]
High	SQL Injection: Improper Neutralization of Special Elements used in an SQL Command	89	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	219	Java	Package	5 seconds ago	[Icon]
Medium	Improper Output Neutralization for Logs	117	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	118	Java	Package	5 seconds ago	[Icon]
Medium	Improper Output Neutralization for Logs	117	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	163	Java	Package	5 seconds ago	[Icon]
Medium	Improper Output Neutralization for Logs	117	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	403	Java	Package	5 seconds ago	[Icon]
Medium	Insufficient Entropy	331	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	242	Java	Package	5 seconds ago	[Icon]
Medium	Insufficient Entropy	331	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	244	Java	Package	5 seconds ago	[Icon]
Medium	Insufficient Entropy	331	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	254	Java	Package	5 seconds ago	[Icon]
Medium	Insufficient Entropy	331	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	253	Java	Package	5 seconds ago	[Icon]
Medium	Insufficient Entropy	331	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	295	Java	Package	5 seconds ago	[Icon]
Medium	Insufficient Entropy	331	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	301	Java	Package	5 seconds ago	[Icon]
Medium	Insufficient Entropy	331	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	303	Java	Package	5 seconds ago	[Icon]
Low	Insufficient Entropy	331	Details	\\Veracode\src\main\java\com\veracode\veracode\ide\...	316	Java	Package	5 seconds ago	[Icon]

Findings (13) | Ignored (0) | Best Practices (5)