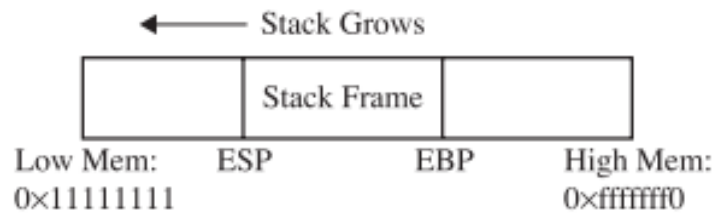
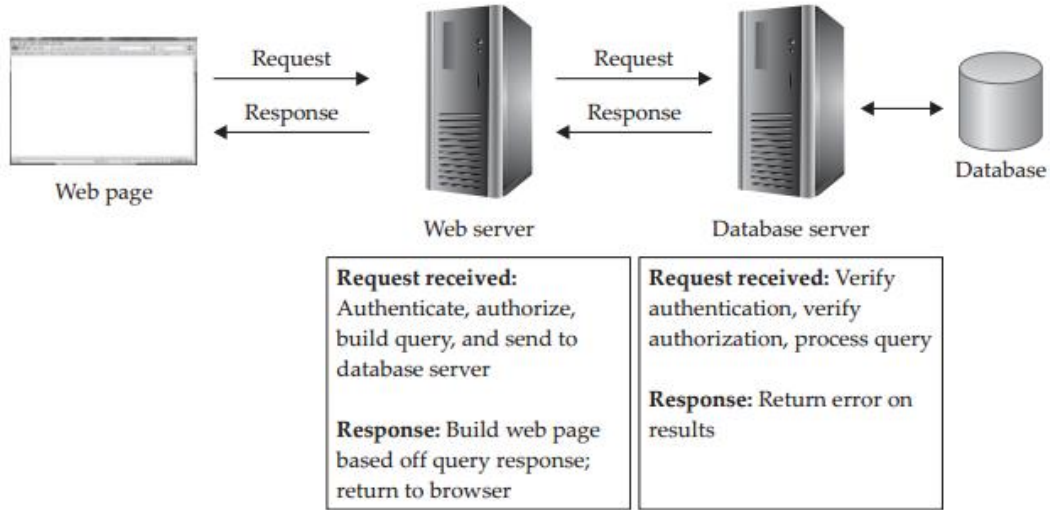
	RV College of Engineering® Department of Computer Science and Engineering CIE - III		
Course & Code	Vulnerability Assessment & Penetration Testing (CY255TBD) Professional Core Course Elective-I		Semester: V
Date: 29/01/2025	Duration: 120 minutes	Max.Marks : 10 Marks (Quiz) + 50 Marks (Test)	
Scheme and Solution			

Sl.no.	Part A Quiz	Marks
1	Scope	01
2	Stack	01
3	Payload creation	01
4	Cross-site scripting (XSS)	01
5	Server-side	01
6	Passive	01
7	Ghidra	01
8	Codebase	01
9	A buffer overflow exploit works by overwriting adjacent memory (e.g., the return address) with malicious data, enabling an attacker to execute arbitrary code.	01
10	Injection vulnerabilities.	01

Sl.no.	Part B Test	Marks
1.a	<p>Definition of buffer overflow and explanation for how it occurs. stack operations: push and pop, and their importance in memory management. Buffer overflow exploits manipulate the stack, such as overwriting the return address. Example or diagram of a buffer overflow on the stack. Implications of buffer overflows and measures to prevent them (e.g., stack canaries, address space randomization).</p>  <p style="text-align: center;"> ← Stack Grows </p> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid black; width: 100px; height: 40px;"></div> <div style="border: 1px solid black; width: 100px; height: 40px; text-align: center;">Stack Frame</div> <div style="border: 1px solid black; width: 100px; height: 40px;"></div> </div> <div style="display: flex; justify-content: space-between; margin-top: 5px;"> Low Mem: 0x11111111 ESP EBP High Mem: 0xffffffff </div>	06
1.b	<ul style="list-style-type: none"> ○ Passive Analysis: <ul style="list-style-type: none"> ▪ Advantages: <ol style="list-style-type: none"> 1. Non-intrusive and stealthy. 2. Ideal for monitoring systems in real-time or long-term. ▪ Limitations: <ol style="list-style-type: none"> 1. May miss vulnerabilities that require active exploitation. 2. Limited in scope and may not cover all potential attack vectors. ▪ Use Cases: <ol style="list-style-type: none"> 1. Continuous monitoring of networks for suspicious behavior. 2. Detecting passive vulnerabilities like weak encryption or misconfigured security settings. ○ Source Code Analysis: <ul style="list-style-type: none"> ▪ Advantages: <ol style="list-style-type: none"> 1. Allows early detection of vulnerabilities during the development phase. 2. Comprehensive review of all code, ensuring thorough vulnerability assessment. ▪ Limitations: <ol style="list-style-type: none"> 1. Requires access to the source code, which may not always be available (e.g., closed-source software). 2. May generate false positives or require manual validation. ▪ Use Cases: <ol style="list-style-type: none"> 1. Reviewing application code for security flaws before deployment. 2. Identifying insecure coding practices and logic errors. ○ Binary Analysis: 	06

	<ul style="list-style-type: none"> ▪ Advantages: <ol style="list-style-type: none"> 1. Can be applied to closed-source software (no access to source code required). 2. Useful for detecting vulnerabilities in compiled code that may not be present in source code. ▪ Limitations: <ol style="list-style-type: none"> 1. Requires expertise in reverse engineering and understanding machine-level code. 2. May not identify all vulnerabilities, especially in highly obfuscated or protected binaries. ▪ Use Cases: <ol style="list-style-type: none"> 1. Analyzing third-party software or closed-source applications for vulnerabilities. 	
2.	 <p>The OWASP Top Ten is a list of the most critical security risks for web applications, compiled by the Open Web Application Security Project (OWASP). These vulnerabilities represent common and impactful security flaws that developers and organizations should address to ensure robust web application security.</p> <p>Addressing OWASP Vulnerabilities Enhances Application Security</p> <ol style="list-style-type: none"> 1. Prevention of Data Breaches: Securing vulnerabilities like injection and broken access control prevents unauthorized data access or exposure. 2. Increased User Trust: A secure application builds trust among users, improving reputation and customer retention. 3. Compliance with Standards: Mitigating these risks helps organizations comply with regulations like GDPR, HIPAA, and PCI-DSS. 4. Cost Reduction: Early detection and remediation of vulnerabilities reduce costs associated with post-breach incident handling. 5. Improved Application Reliability: Secure systems experience fewer disruptions from cyberattacks, ensuring continuous service availability. 	10
3	<p>Binary Analysis is the process of inspecting and analyzing the compiled binary code of an application to identify potential security flaws. Unlike source code analysis, which examines human-readable code, binary analysis involves working with the compiled version of an application, which may not include the original source code. This method is particularly useful for security assessments when the source code is unavailable or when vulnerabilities exist at the compiled level.</p>	10

Process of Binary Analysis

1. **Obtaining the Binary:**

The first step in binary analysis is to acquire the compiled binary of the application. This can be an executable (.exe, .elf) or a library (.dll, .so) that contains the program's logic. For web applications, it could also involve analyzing compiled scripts like those written in Java or .NET.

2. **Disassembly:**

The next step is to **disassemble** the binary. Disassembly converts machine code (or bytecode) back into assembly language, which is more readable to humans. This allows security analysts to examine the structure of the program, including functions, calls, and memory handling.

3. **Reverse Engineering:**

After disassembly, **reverse engineering** is used to interpret the logic of the program. Analysts attempt to reconstruct the original program logic, identify critical operations, and locate areas where vulnerabilities could be present (such as buffer overflows or improper memory handling). Reverse engineering may involve identifying strings, functions, and system calls that can indicate potential vulnerabilities.

4. **Dynamic Analysis:**

In addition to static disassembly, **dynamic analysis** is often performed. This involves running the binary in a controlled environment (such as a sandbox or virtual machine) and observing its behavior during execution. Analysts monitor runtime behavior, input validation, system calls, and network communication to detect anomalies, vulnerabilities, or malicious actions.

5. **Vulnerability Identification:**

During binary analysis, analysts look for common vulnerabilities such as:

- **Buffer Overflows:** Identifying areas where input data can overflow and overwrite memory.
- **Use-After-Free:** Detecting instances where memory is used after it has been freed, leading to potential exploits.
- **Code Injection:** Looking for areas where an attacker might inject malicious code.
- **Privilege Escalation:** Finding ways the binary might allow unauthorized users to gain higher privileges.
- **Uninitialized Memory:** Identifying areas where memory is not initialized properly, which can lead to unpredictable behavior.

6. **Patching and Recommendations:**

After detecting vulnerabilities, the next step is to **recommend fixes**. These might involve altering memory handling, using safer libraries, implementing proper bounds checking, or other methods to mitigate identified risks.

Detecting Security Flaws in a Compiled Application

Binary analysis helps in detecting flaws that may not be visible through source code inspection. Key methods used to detect security flaws in a compiled application include:

1. **Memory Corruption Vulnerabilities:**

Analyzing how the application handles memory helps to detect issues like buffer overflows, heap corruption, and use-after-free errors that can be exploited for arbitrary code execution.

2. **Control Flow Hijacking:**

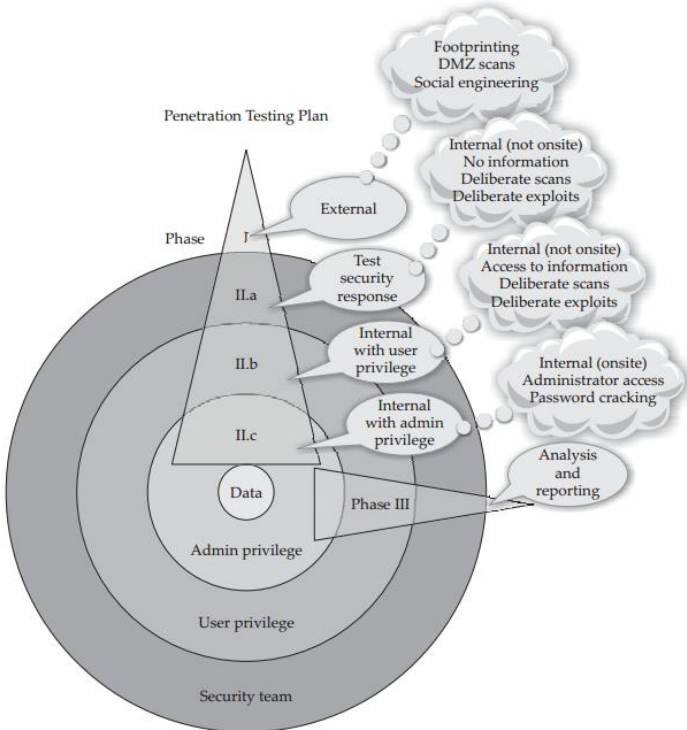
Attackers can manipulate the control flow of a program, for instance, by exploiting stack-based buffer overflows to overwrite return addresses. By analyzing how the program's control flow is structured, these vulnerabilities can be identified.

3. **Improper Input Handling:**

Binary analysis also detects vulnerabilities related to poor input validation, which can lead to SQL injection, command injection, and other similar attacks.

4. **Exploiting Third-Party Libraries:**

Compiled applications often rely on external libraries, which may contain

	<p>vulnerabilities. Binary analysis can help identify issues in these libraries, such as known exploits or misconfigurations.</p> <p>Commonly Used Tools for Binary Analysis</p> <p>Several tools are employed in binary analysis to automate and facilitate the process of inspecting compiled applications. These tools help reverse engineers and security professionals to disassemble, debug, and analyze binaries for security flaws.</p> <ol style="list-style-type: none"> 1. IDA Pro (Interactive Disassembler): IDA Pro is one of the most popular disassemblers and debuggers, allowing analysts to reverse engineer a binary into a human-readable assembly language. It also offers features like function graphing and debugging capabilities. 2. Ghidra: Ghidra is a free, open-source reverse engineering tool developed by the NSA. It provides powerful disassembling, decompiling, and debugging capabilities for analyzing binaries. 3. Radare2: Radare2 is an open-source framework for reverse engineering, offering a wide range of tools for disassembly, debugging, and exploiting vulnerabilities in binaries. 4. OllyDbg: OllyDbg is a popular 32-bit debugger used to analyze the runtime behavior of a program. It's useful for dynamic analysis, such as detecting buffer overflows and code injection. 5. Binary Ninja: Binary Ninja is a reverse engineering platform that automates parts of the analysis, such as identifying vulnerabilities like control-flow hijacking or memory corruption. 6. Frida: Frida is a dynamic instrumentation toolkit that allows analysts to interact with a running process, enabling the monitoring of function calls, system calls, and memory regions in real time. 	
4	<p>List of three main phases along with sub phases, Diagram- 4 marks, brief explanation- 4 marks</p> 	08

5.a	<p>Dradis Server: The Dradis framework is an open source system for information sharing</p> <p>Scope of a Penetration Test 03 marks each</p>	06
5.b	<ul style="list-style-type: none"> ○ Server-Side Defenses: <ol style="list-style-type: none"> 1. Input Validation and Sanitization: Sanitize user inputs by removing dangerous characters like <, >, and &, ensuring they can't be executed as code. 2. Output Encoding: Encode data before rendering it in the browser, ensuring that potentially harmful characters are displayed safely. For example, encoding < as &lt; in HTML output. 3. Content Security Policy (CSP): Implement a CSP to restrict the types of content that can be loaded, preventing the execution of malicious scripts. ○ Client-Side Defenses: <ol style="list-style-type: none"> 1. HTTP-Only and Secure Cookies: Ensure cookies are marked as <code>HttpOnly</code> and <code>Secure</code> to prevent JavaScript from accessing sensitive data. 2. JavaScript Frameworks: Use secure frameworks like AngularJS and React, which automatically escape user inputs and protect against XSS attacks. 3. Subresource Integrity (SRI): Implement SRI to ensure that external scripts haven't been tampered with. 	04