# SEC-202: Secure Start-Up

## **Lecture 5** – Application Security and Supply Chain

*"Building Security In."*
*Moving from "Security as a Gatekeeper" (blocking release) to "Security as a Guardrail" (helping developers code safely).*

Instructed By:

Dr. Charnon Pattiyanon

Assistant Director of IT and Instructor
**CMKL University**

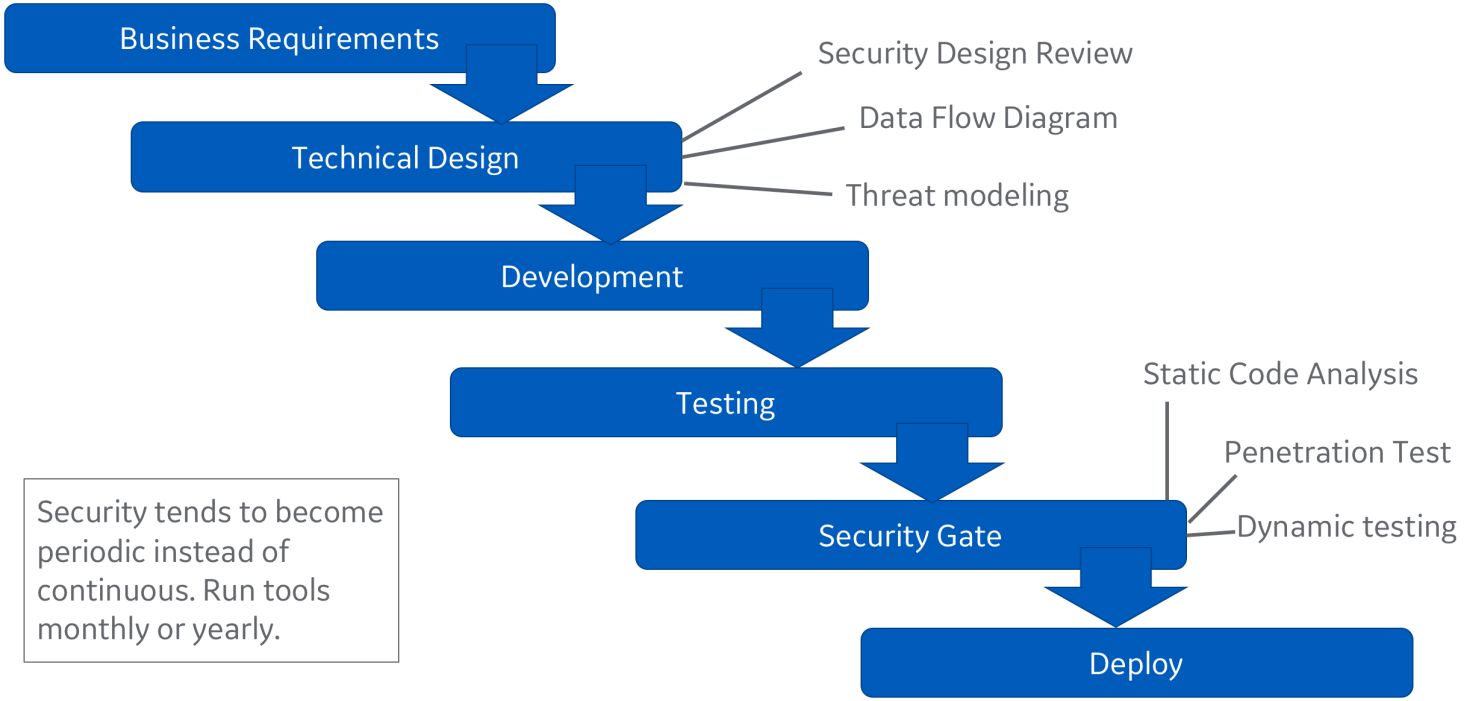
Artificial Intelligence and Computer Engineering (AICE) Program

# Class Agenda

- The Shift Left
- The OWASP Top 10
- Automated Testing Tools
- Supply Chain Security
- Secrets Detection
- Penetration Testing
- API Security
- Third Party Risk Management (TPRM)

# The Shift Left

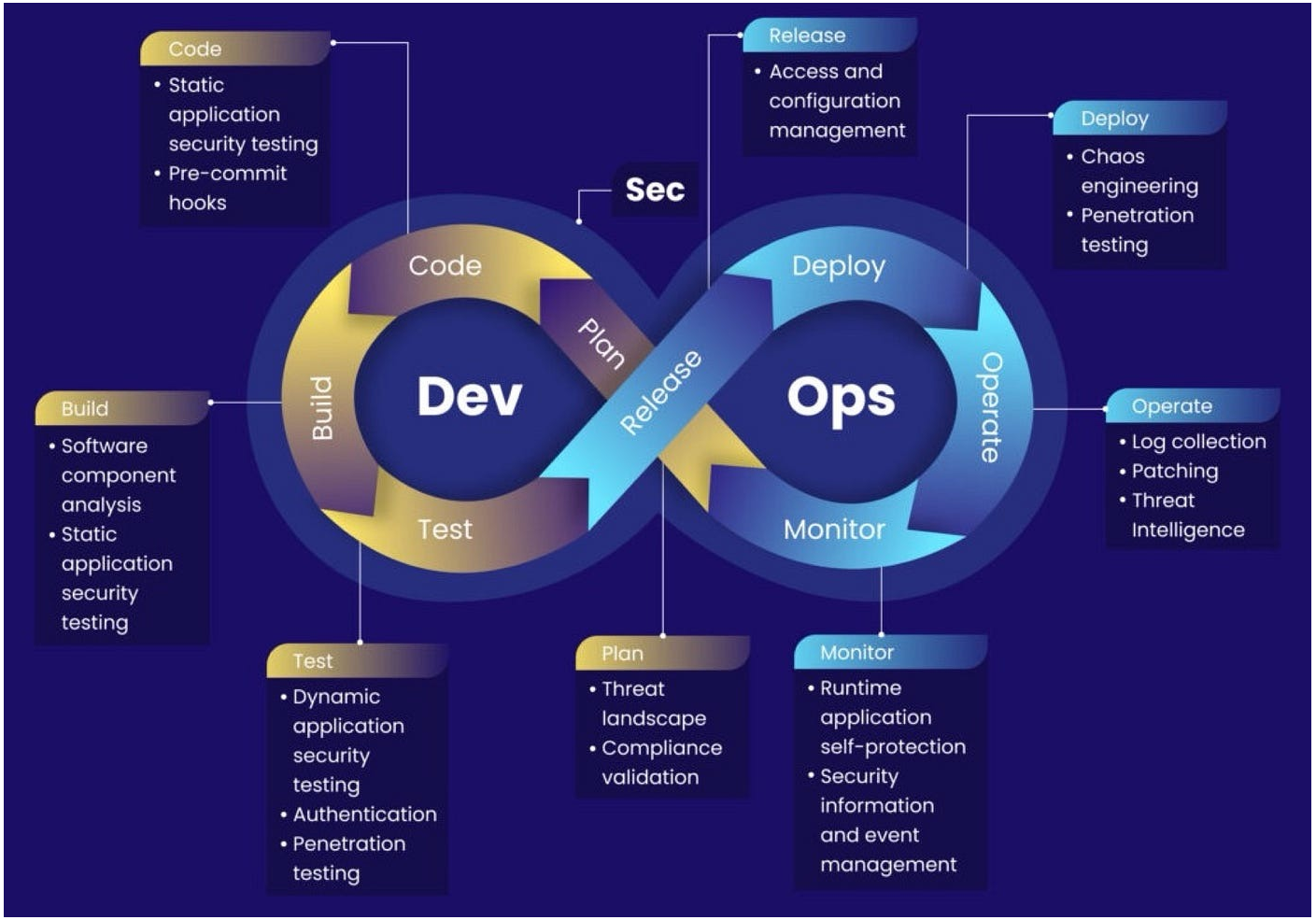
- **The Old Way (Waterfall):**
  - Design -> Code -> Test -> Security Audit (2 weeks before launch) -> Panic -> Delay Launch.



# The Shift Left

- **The New Way (DevSecOps):**
  - Security is integrated into every stage of the loop.
  - Automated tools run every time a developer saves code.

**Goal:** Find bugs when they are cheap to fix (during coding), not when they are expensive (in production).



# The OWASP Top 10 Web Application Security Risks

- **Significance:** The industry-standard list of the most critical web application security risks.
- **Key Examples:**
  - Broken Access Control: A user can see someone else's data (e.g., changing `user_id=100` to `user_id=101` in the URL).
  - Injection (SQLi): Attackers tricking the database into revealing all data.
  - Cryptographic Failures: Storing passwords in plain text or using weak encryption.

https://owasp.org/Top10/2025/

# The OWASP Top 10 Web Application Security Risks

## A01:2025 Broken Access Control

### Background.

Maintaining its position at #1 in the Top Ten, 100% of the applications tested were found to have some form of broken access control. Notable CWEs included are *CWE-200: Exposure of Sensitive Information to an Unauthorized Actor*, *CWE-201: Exposure of Sensitive Information Through Sent Data*, *CWE-918 Server-Side Request Forgery (SSRF)*, and *CWE-352: Cross-Site Request Forgery (CSRF)*. This category has the highest number of occurrences in the contributed data, and second highest number of related CVEs.

### Score table.

| CWEs Mapped | Max Incidence Rate | Avg Incidence Rate | Max Coverage | Avg Coverage | Avg Weighted Exploit | Avg Weighted Impact | Total Occurrences | Tota CVE |
|---|---|---|---|---|---|---|---|---|
| 40 | 20.15% | 3.74% | 100.00% | 42.93% | 7.04 | 3.84 | 1,839,701 | 32,6 |

### Description.

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, commonly known as 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 that modifies API requests.
- Permitting viewing or editing someone else's account by providing its unique identifier (insecure direct object references)
- An accessible API with missing access controls for POST, PUT, and DELETE.
- Elevation of privilege. Acting as a user without being logged in or or gaining privileges beyond those expected of the logged in user (e.g. admin access).
- Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token, a cookie or hidden field manipulated to elevate privileges, or abusing JWT invalidation.
- CORS misconfiguration allows API access from unauthorized or untrusted origins.
- Force browsing (guessing URLs) to authenticated pages as an unauthenticated user or to privileged pages as a standard user.

# The OWASP Top 10 Web Application Security Risks

## How to prevent.

Access control is only effective when implemented in trusted server-side code or serverless APIs, where the attacker cannot modify the access control check or metadata.

- Except for public resources, deny by default.
- Implement access control mechanisms once and reuse them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.
- Model access controls should enforce record ownership rather than allowing users to 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).
- Implement rate limits on 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 be short-lived to minimize the window of opportunity for an attacker. For longer-lived JWTs, consider using refresh tokens and following OAuth standards to revoke access.
- Use well-established toolkits or patterns that provide simple, declarative access controls.

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

## Example attack scenarios.

**Scenario #1:** The application uses unverified data in an SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker can simply modify the browser's 'acct' parameter to send any desired account number. If not correctly verified, the attacker can access any user's account.

```
https://example.com/app/accountInfo?acct=notmyacct
```

**Scenario #2:** An attacker simply forces browsers to target URLs. Admin rights are required for access to the admin page.

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

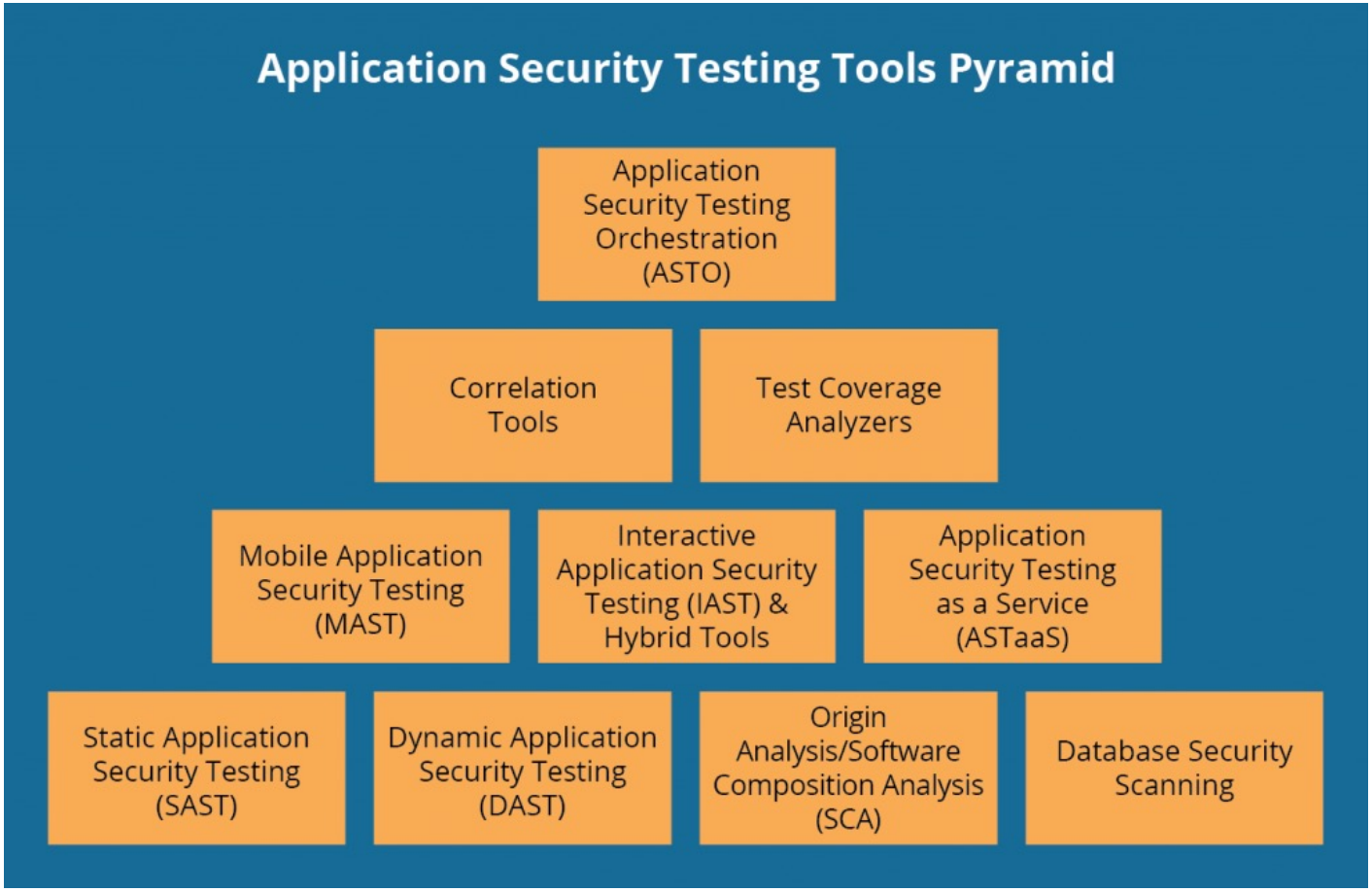
If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

**Scenario #3:** An application puts all of their access control in their front-end. While the attacker cannot get to `https://example.com/app/admin_getappInfo` due to JavaScript code running in the browser, they can simply execute:

```
$ curl https://example.com/app/admin_getappInfo
```

from the command line.

# Application Security Testing Tools



# Application Security Testing Tools (SAST vs. DAST)

| Static Application Security Testing (SAST) | | Dynamic Application Security Testing (DAST) |
|---|---|---|
| Analyzes source code, bytecode, or binary for vulnerabilities without executing the application. | **Definition** | Examines the application in a running state by simulating attacks to find vulnerabilities. |
| Early in the SDLC during the coding phase. | **When?** | After deployment, during testing or staging environments. |
| White-box testing (access to source code). | **Types** | Black-box testing (no access to source code). |
| Identifies coding flaws, hardcoded secrets, and potential vulnerabilities like SQL injection or XSS. | **Detection** | Detects vulnerabilities like XSS, CSRF, and SQL injection by interacting with the application. |
| Code-level vulnerabilities before deployment. | **Focus** | Application behavior under simulated attacks. |
| Reviewing source code for vulnerabilities before deployment. | **Use Cases** | Penetration testing. |
| Ensuring compliance with secure coding practices. | | Validating application security in staging environments. |
| | | Identifying vulnerabilities in live applications. |

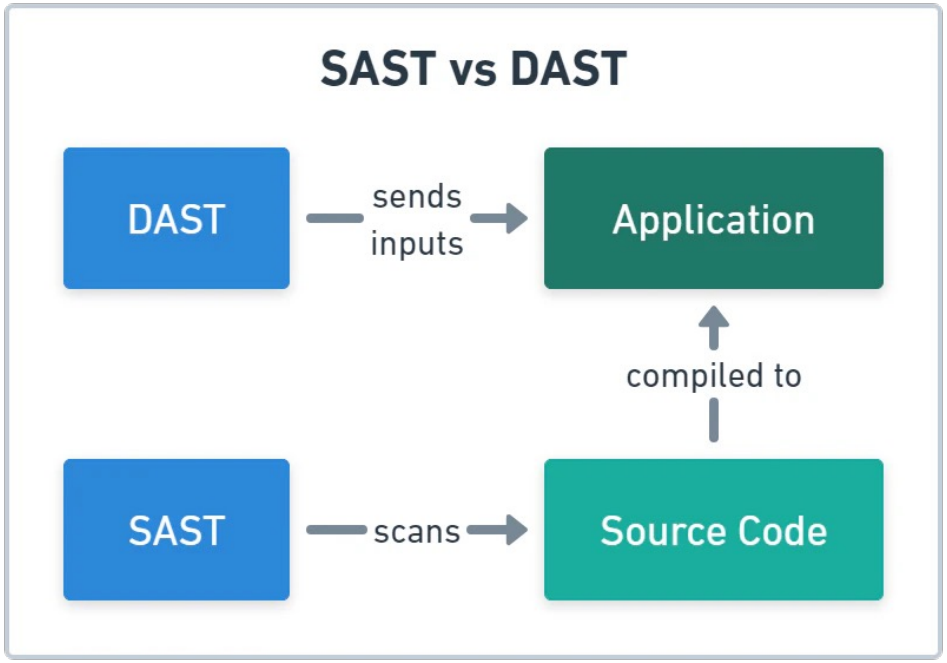
# Application Security Testing Tools (SAST vs. DAST)

| Tools | Features | Language Supported |
|---|---|---|
| SonarQube | • Detects vulnerabilities, code smells, and bugs.<br>• Integrates with CI/CD pipelines. | Java, C#, JavaScript, Python, etc. |
| Checkmarx | • Offers deep code analysis.<br>• Highly customizable for specific projects. | Multiple languages. |
| Fortify Static Code Analyzer | • Enterprise-grade tool for static analysis.<br>• Provides detailed vulnerability insights. | Over 25 languages. |
| Veracode Static Analysis | • Cloud-based SAST.<br>• Easy integration with CI/CD pipelines. | Java, .NET, Python, etc. |
| Codacy | • Focuses on code quality and security issues.<br>• Integrates with GitHub, GitLab, etc. | Multiple languages. |

| Tools | Features | Use Cases |
|---|---|---|
| OWASP ZAP | • Open-source DAST tool.<br>• Automated vulnerability scanning.<br>• Active and passive scanning. | Penetration testing, security assessments. |
| Burp Suite | • Advanced manual and automated DAST capabilities.<br>• Highly extensible with plugins | Web application security testing. |
| Acunetix | • Automated web application scanner.<br>• Detects over 7,000 vulnerabilities. | Comprehensive web vulnerability scanning. |
| Netsparker | • Accurate DAST with minimal false positives.<br>• Supports automation in CI/CD pipelines. | Scanning for web vulnerabilities like XSS and SQL injection. |
| AppScan | • Enterprise-grade DAST.<br>• Integration with DevSecOps workflows.<br>• Focus on OWASP Top 10. | Validating security in staging environments. |

# How to Integrate SAST and DAST Tools in DevSecOps

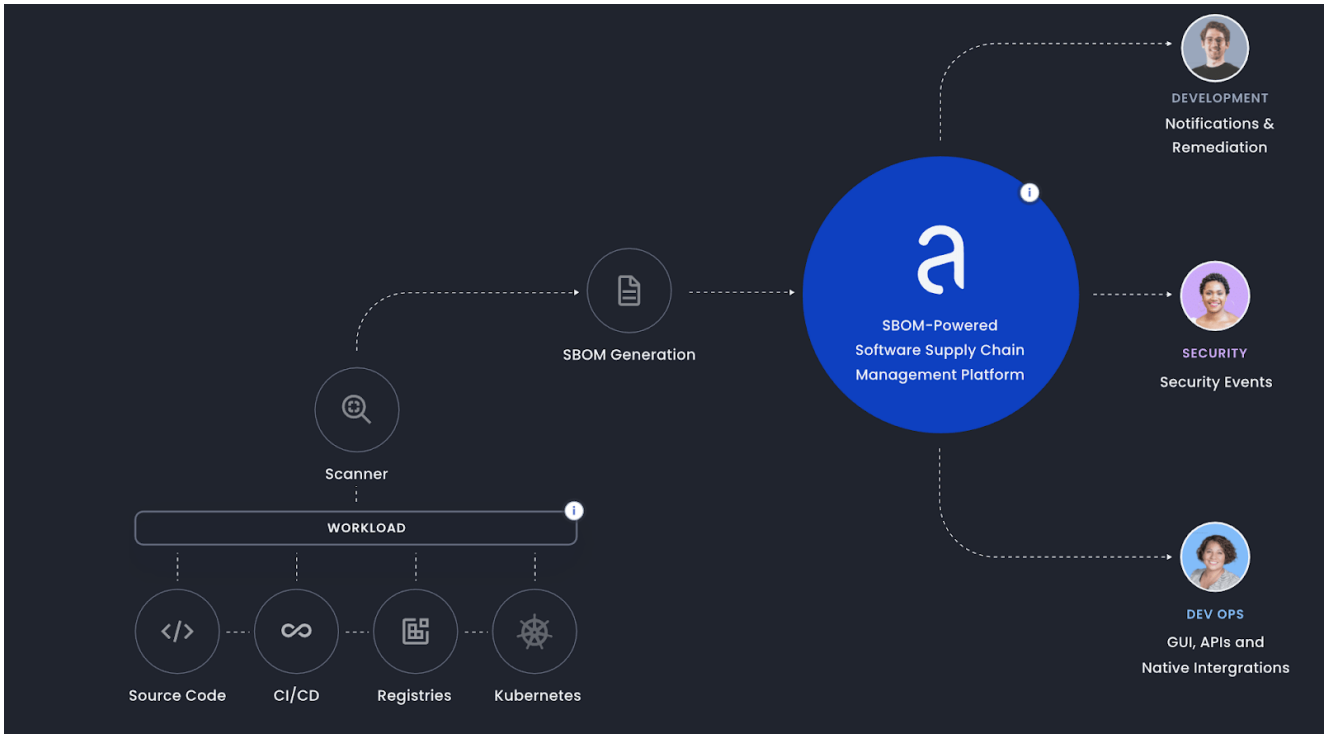
- **Integrating SAST:**
  - Use SAST tools early in the development lifecycle during code writing and reviews.
  - Automate SAST in CI/CD pipelines to prevent introducing vulnerabilities.
  - **Example:** Run SonarQube as part of Jenkins builds.
- **Integrating DAST:**
  - Conduct regular scans in staging and pre-production environments.
  - Use DAST tools during integration testing or user acceptance testing (UAT).
  - **Example:** Automate OWASP ZAP scans in CI/CD pipelines.



# Supply Chain Security

- **The Reality:** Modern software is 90% Open-Source code (Legos) and 10% custom code (Glue).
- **The Risk:** You didn't write the vulnerability; you imported it. (e.g., Log4j).

- **Software Composition Analysis:**
  - Tools that scan your `package.json` or `requirements.txt`.
  - **Alert:** "You are using library v1.0 which has a critical flaw. Upgrade to v1.1."
- **SBOM (Software Bill of Materials):** A formal list of "ingredients" in your software.

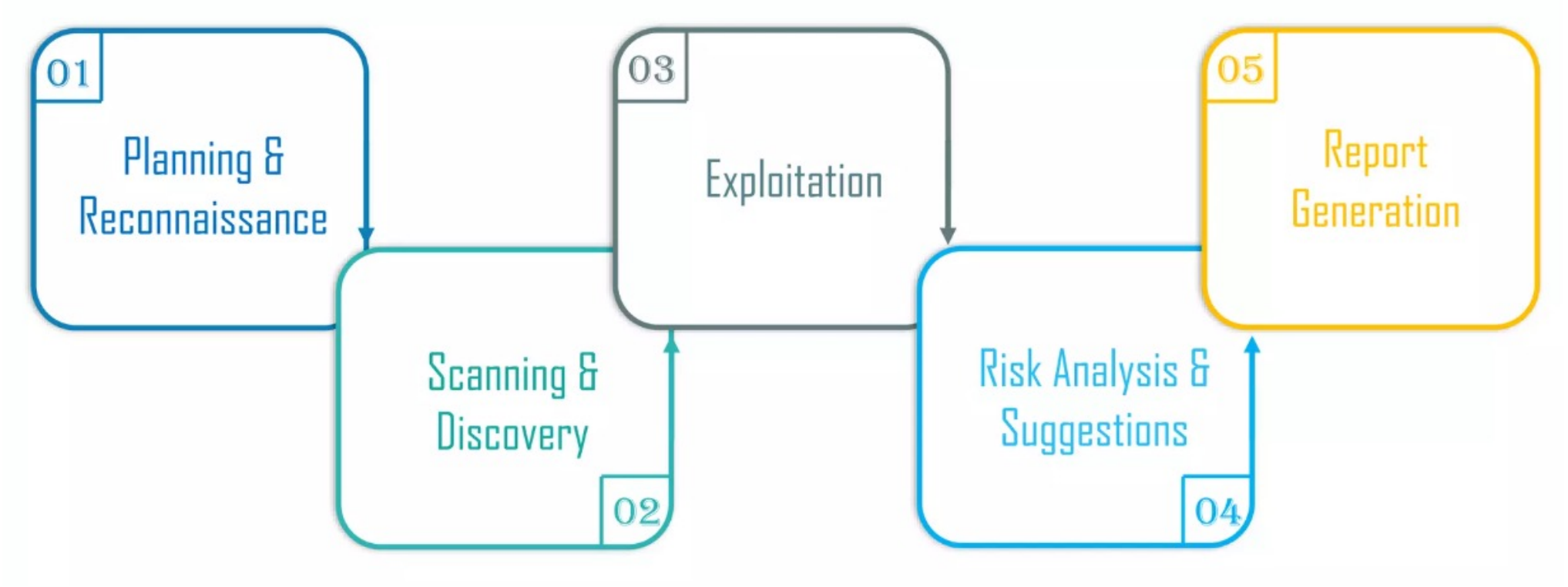


# Penetration Testing

**Penetration Testing** is an attempt to exploit the vulnerabilities to determine whether unauthorized access or other malicious activity is possible.

**Purpose**

# Penetration Testing Phases

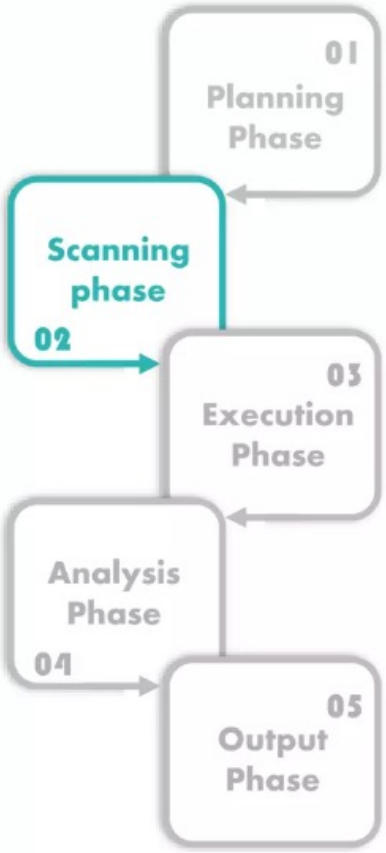


# Penetration Testing Phases


Activities involved in this phase

- Defining goals & scope of a test
- Gathering Intelligence
- Deciding on testing methods to be use

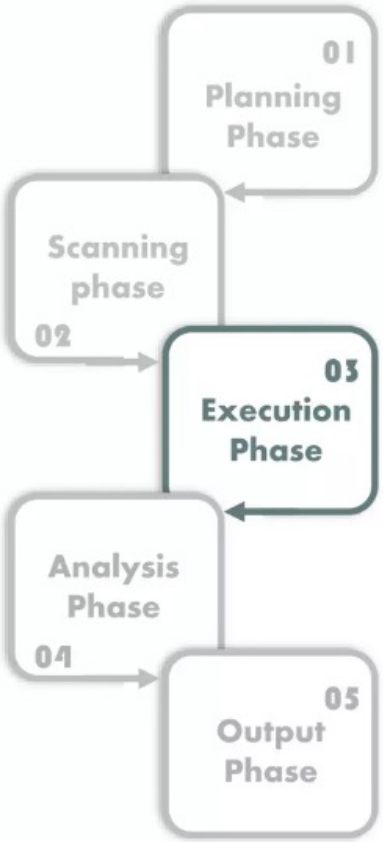
# Penetration Testing Phases




Mapping the attack vectors & identifying vulnerabilities

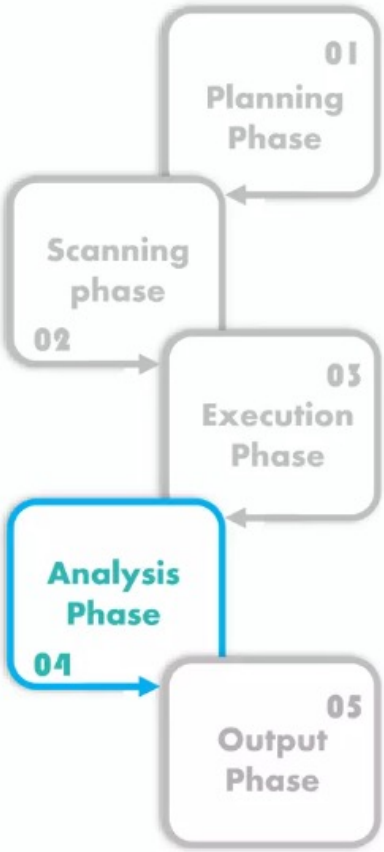
| Static Analysis | Dynamic Analysis |
| --- | --- |
| Inspecting application code logic, functions etc | Inspecting application code in running state |

# Penetration Testing Phases




Crucial phase where actual damage is done

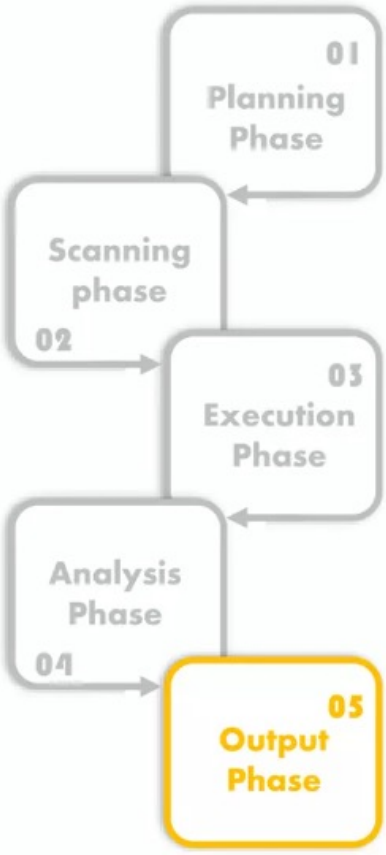
# Penetration Testing Phases

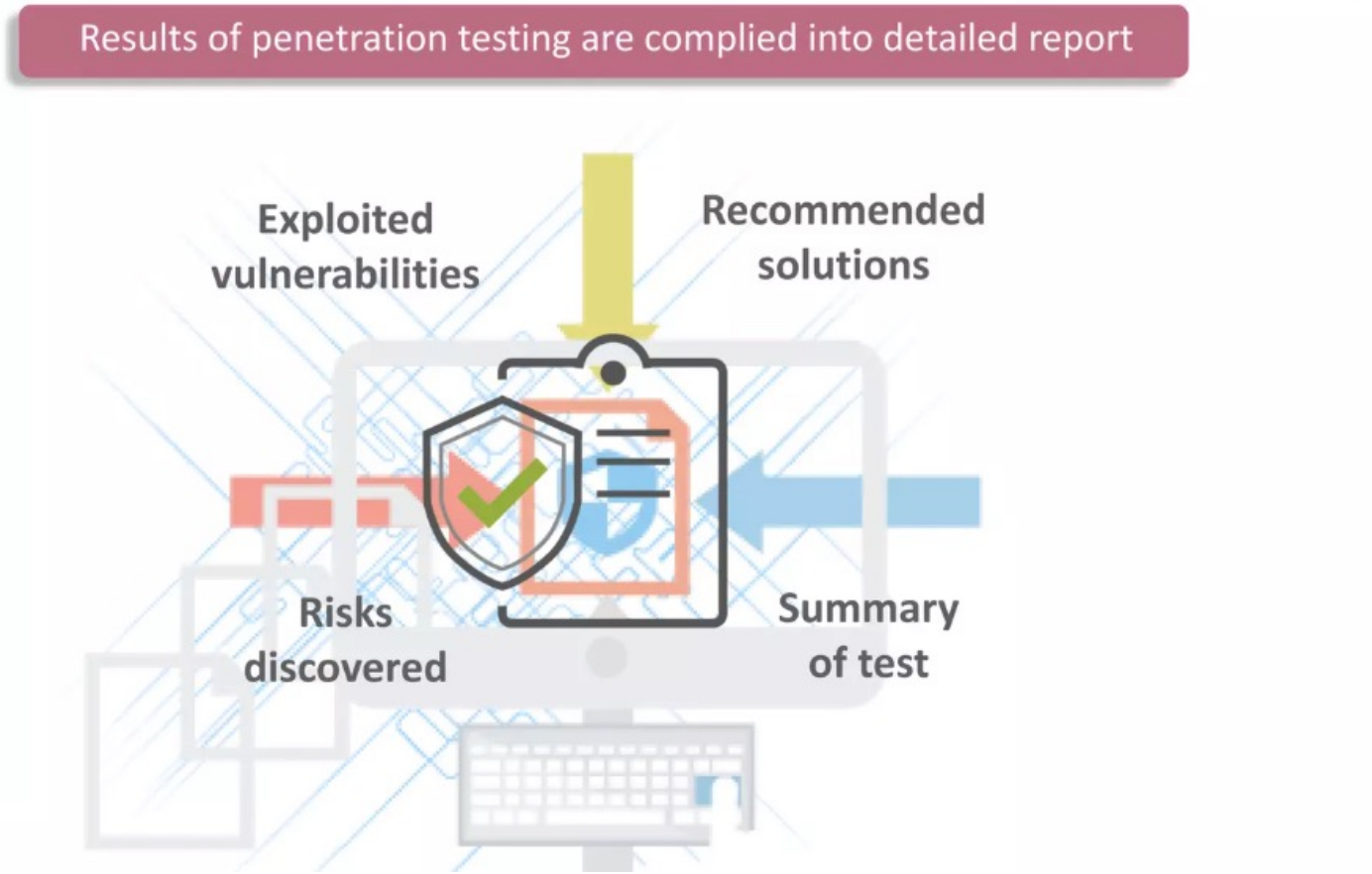



**Activities involved in this phase**

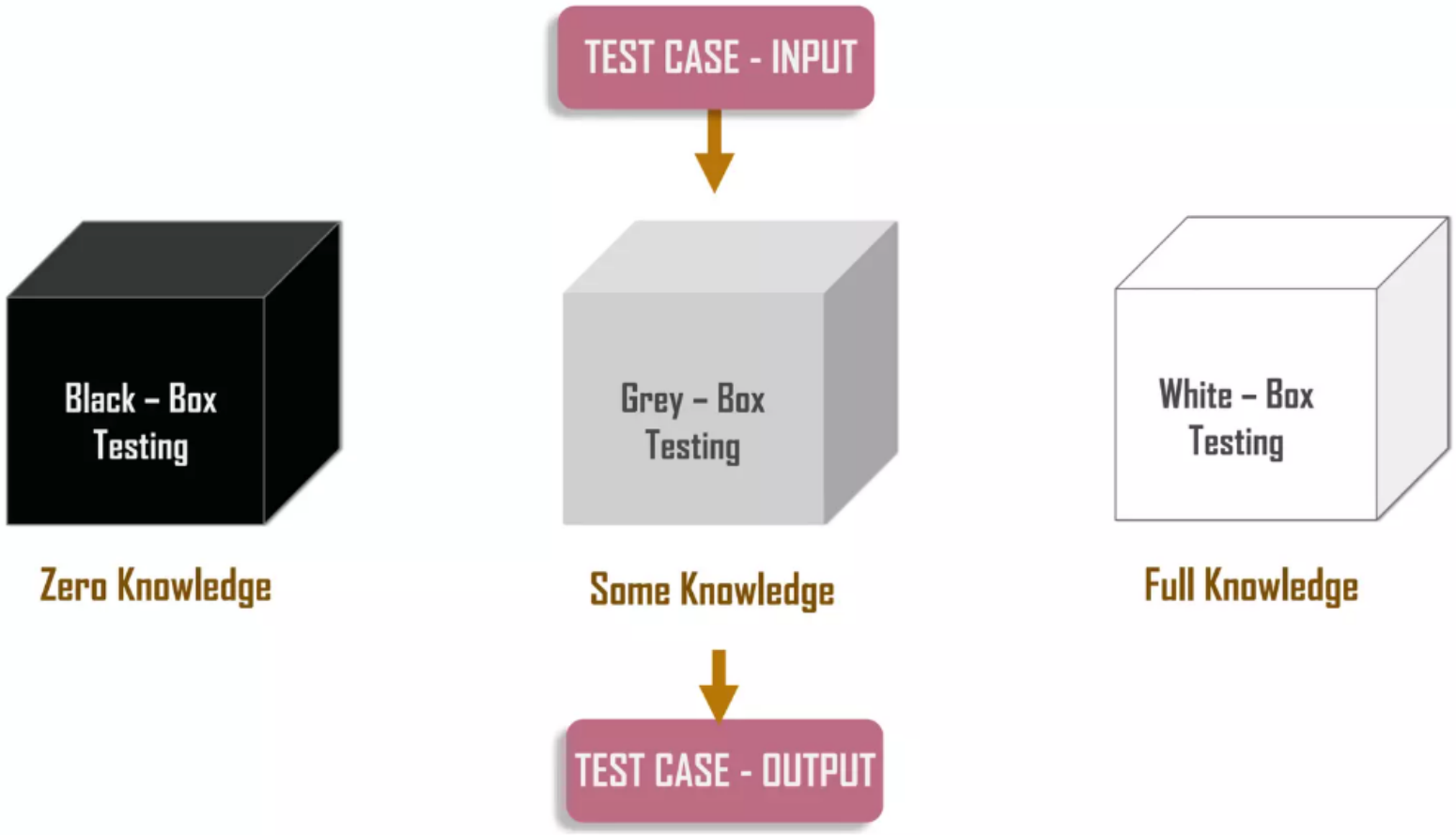
- Collect the evidence of exploited vulnerabilities
- Categorize the risks to – Critical, High, Medium & Low
- Reporting results to executive management

# Penetration Testing Phases



# Types of Penetration Testing



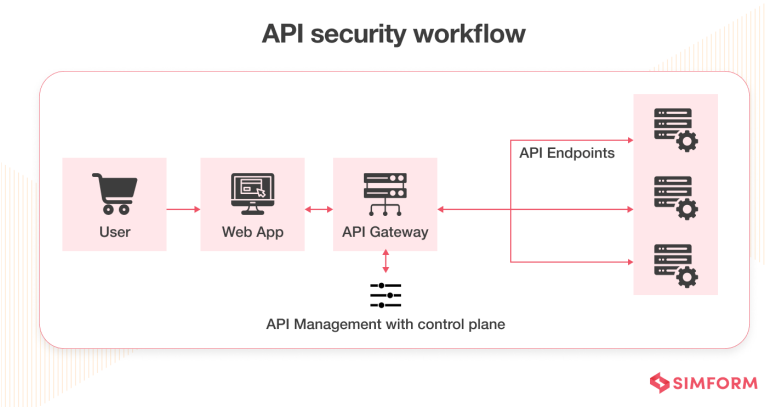
# Penetration Testing Tools

## Why do we need penetration tools?

- [ ] Saves time & effort
- [ ] Accurate results
- [ ] Advanced analysis
- [ ] Gather bulk data
- [ ] Automate manual tasks

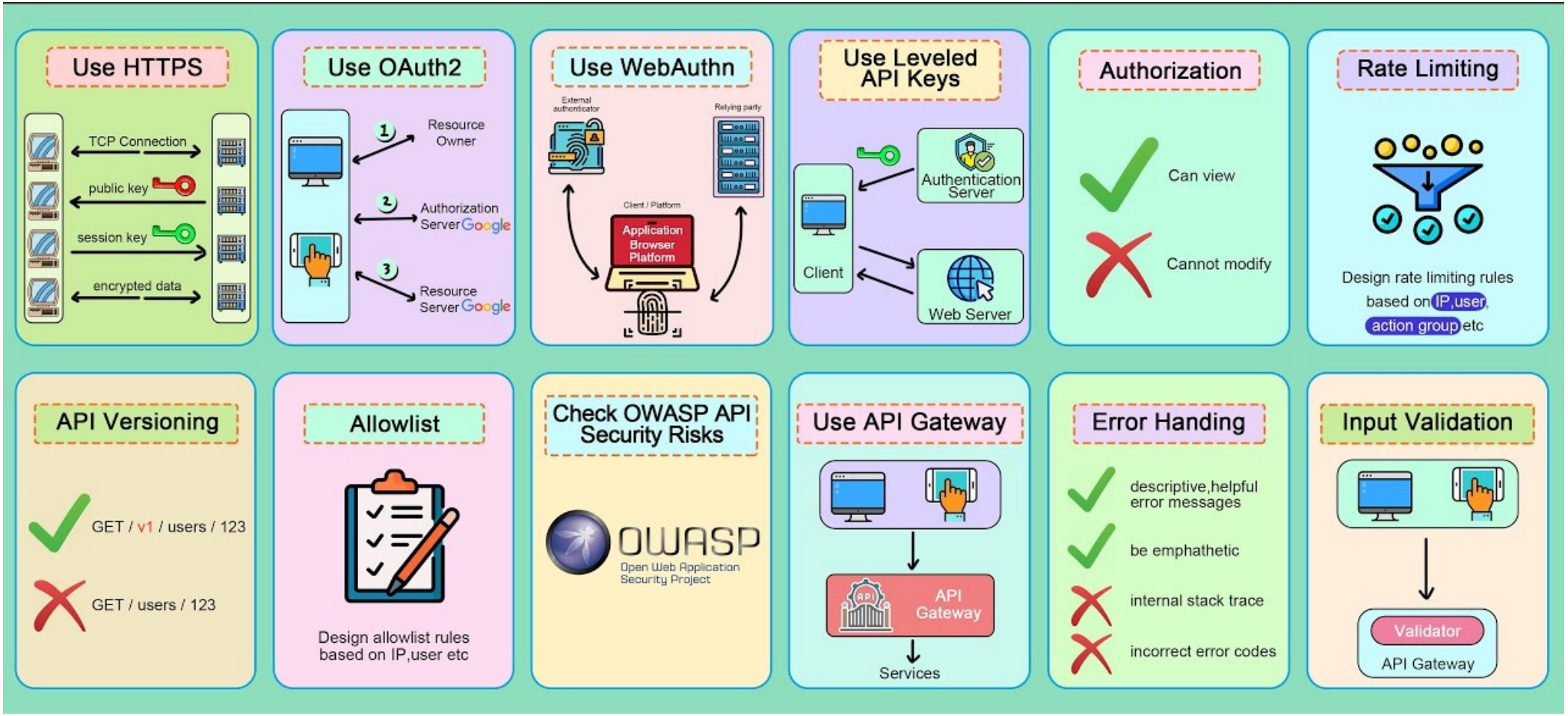
## Popular penetration tools

# API Security



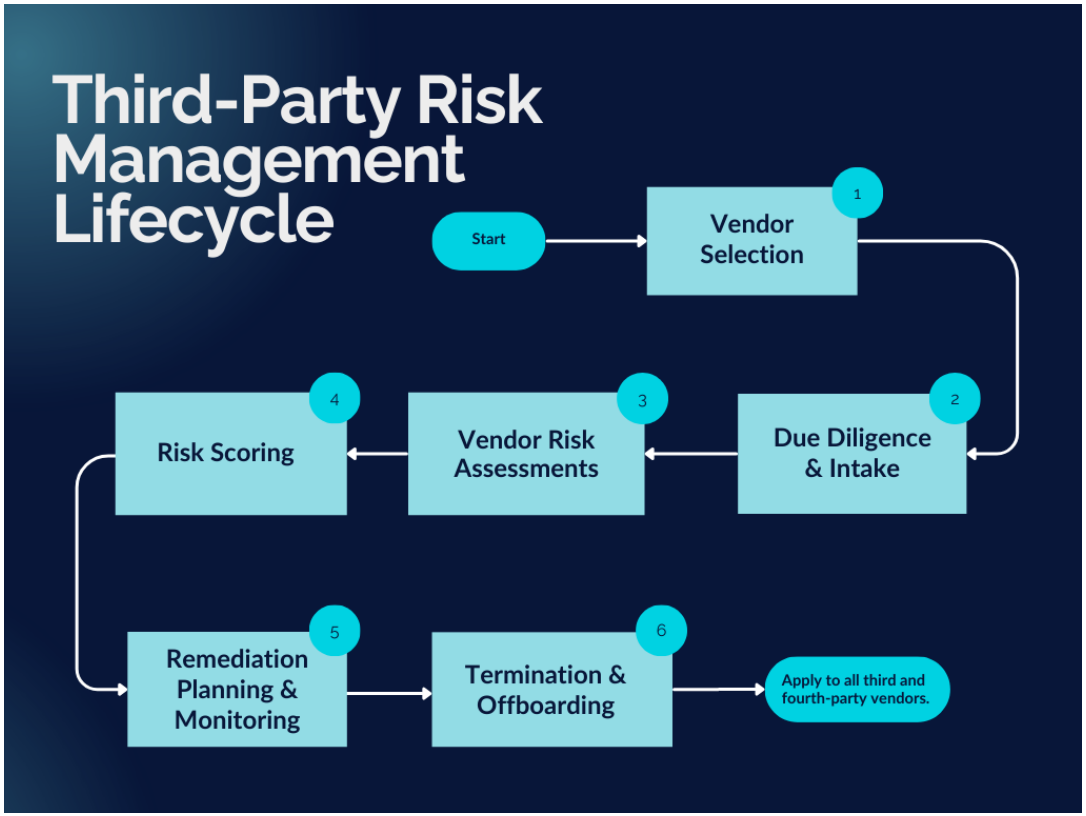

- **The Hidden Door:** APIs are often less protected than the main website.
- **Common Risks:**
  - Shadow APIs: Old endpoints developers forgot to turn off.
  - BOLA (Broken Object Level Authorization): The API lets User A read User B's object.
- **Best Practices:**
  - Authentication & Authorization: Verifying user identity and ensuring they only access authorized data/services.
  - Encryption: Using HTTPS/TLS to protect data in transit from eavesdropping.
  - Rate Limiting & Throttling: Preventing Denial-of-Service (DoS/DDoS) attacks and abuse by limiting request volume.
  - Input Validation & Sanitization: Preventing injection attacks by filtering malicious input.
  - API Gateways: Using gateways to centralize traffic management, security policies, and monitoring.
  - Monitoring & Testing: Continuously monitoring for anomalies and conducting regular security tests to identify vulnerabilities.

# API Security

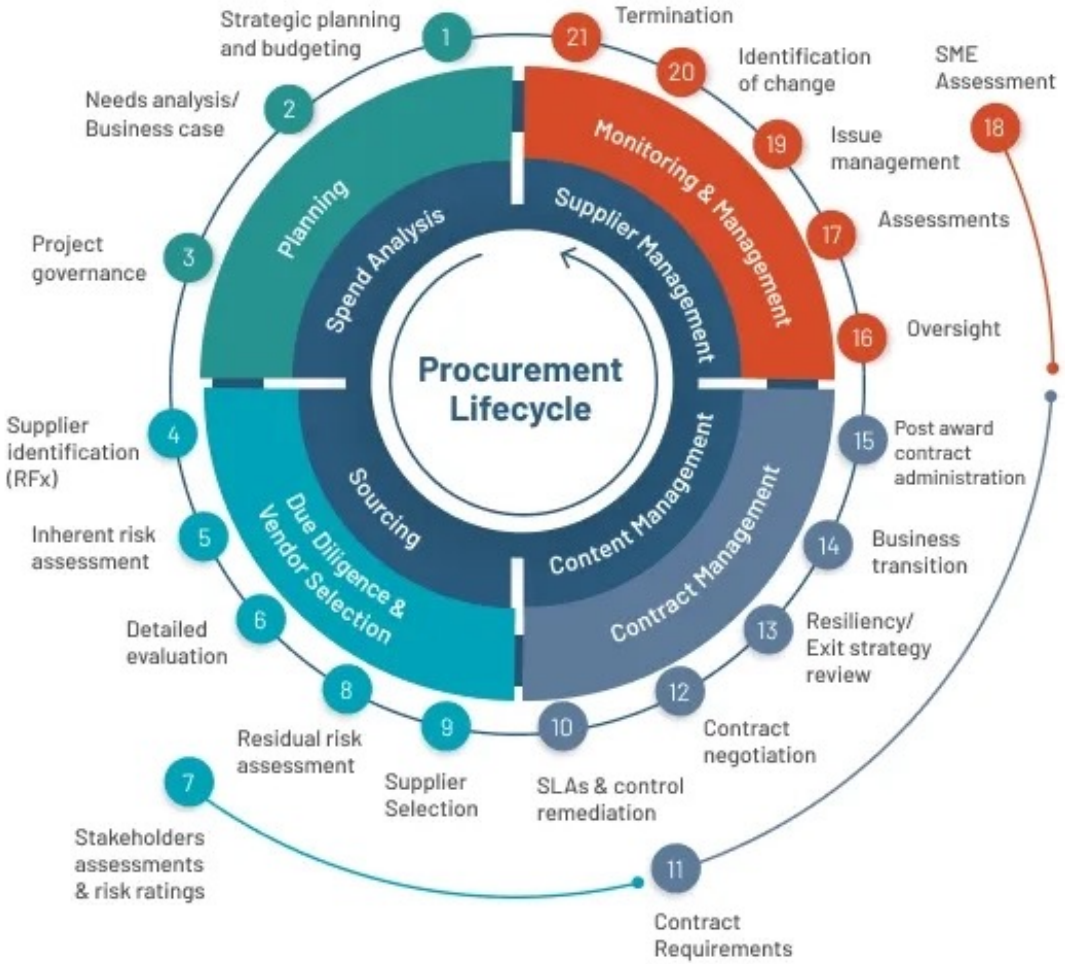


# Third Party Risk Management (TPRM)

- **Vendor Assessment:**
  - Before buying software, ask: "Do they have a SOC 2?" "Do they encrypt data?"
- **Continuous Monitoring:**
  - Using tools (SecurityScorecard/BitSight) to watch vendor security ratings.
- **Contractual Clauses:**
  - **Right to Audit:** "We can check your security."
  - **Notification:** "You must tell us within 24 hours if you are breached."



# Third Party Risk Management (TPRM)



# Key Takeaways

- **Shift Left:** Catch bugs early to save money.
- **Supply Chain:** You are responsible for the code you import.
- **Logic Flaws:** Scanners find syntax errors; Humans (Pen Testers) find logic errors.
- Security in Every Step of Application Development


# End of the Lecture

Please do not hesitate to ask any questions to free your curiosity,

If you have any further questions after the class, please contact me via email (charnon@cmkl.ac.th).