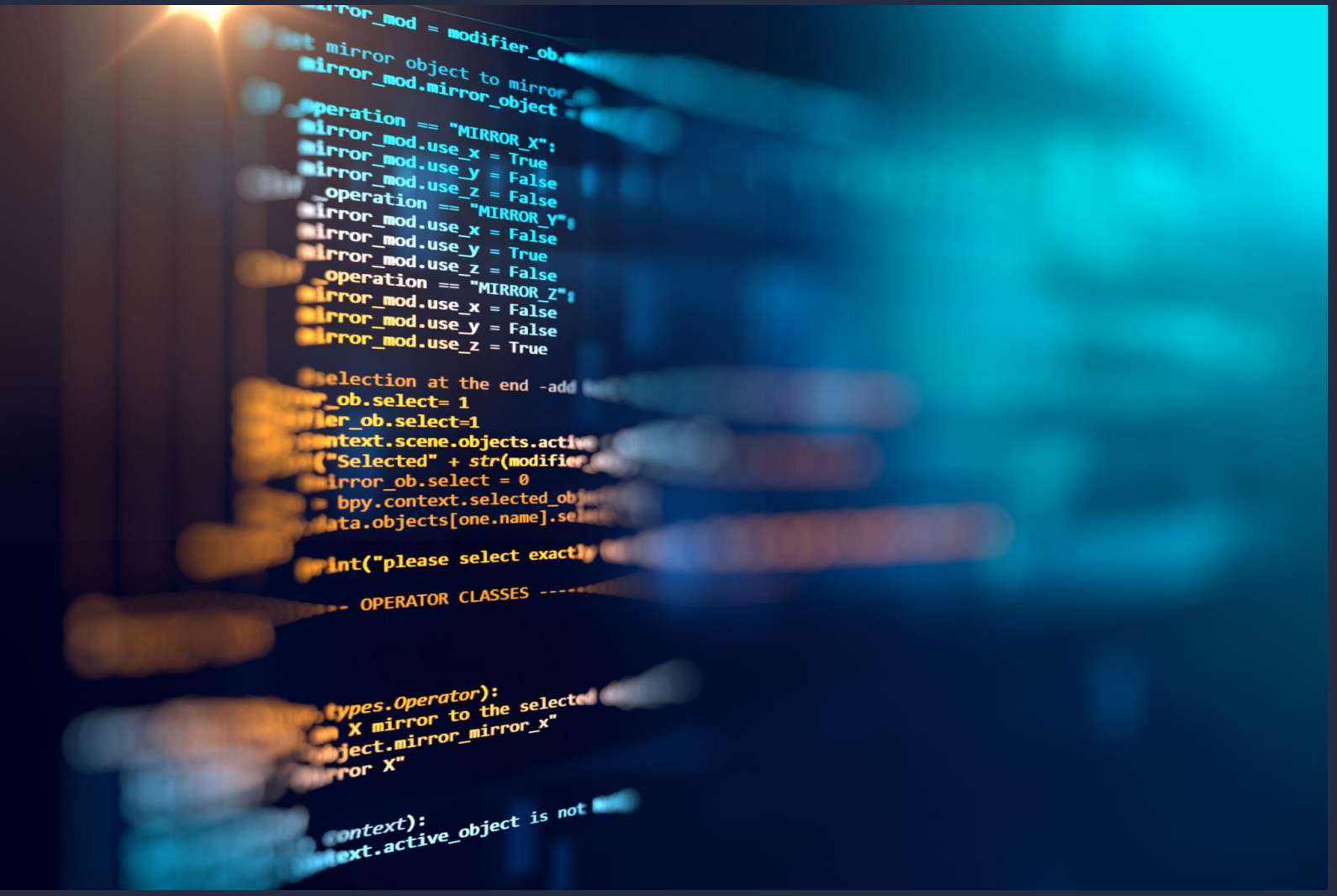


APPLICATION SECURITY INTRODUCTION - OVERVIEW

Joas Antnio

Details

This pdf brings some concepts and study materials for those who want to get started in the field of application security.

https://www.linkedin.com/in/joas-antonio-dos-santos


ATTACKS AND
VULNERABILITIES
https://www.linkedin.com/in/joas-antonio-dos-santos

Most Commons Application Attacks - XSS

A recent study by Precise Security found that the XSS attack is the most common cyberattack making up **approximately 40% of all attacks**. Even though it's the most frequent one, most of these attacks aren't very sophisticated and are executed by amateur cyber criminals using scripts that others have created.

Cross-site scripting targets the users of a site instead of the web application itself. The malicious hacker inserts a piece of code into a vulnerable website, which is then executed by the website's visitor. The code can compromise the user's accounts, activate Trojan horses or modify the website's content to trick the user into giving out private information.

You can protect your website against XSS attacks by setting up a web application firewall (WAF). WAF acts as a filter that identifies and blocks any malicious requests to your website. Usually, web hosting companies already have WAF in place when you purchase their service, but you can also set it up yourself.

https://www.tripwire.com/state-of-security/featured/most-common-website-security-attacks-and-how-to-protect-yourself/

XSS (Cross Site Scripting) - Types

Stored XSS (AKA Persistent or Type I)

Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser. With the advent of HTML5, and other browser technologies, we can envision the attack payload being permanently stored in the victim's browser, such as an HTML5 database, and never being sent to the server at all.

Reflected XSS (AKA Non-Persistent or Type II)

Reflected XSS occurs when user input is immediately returned by a web application in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request, without that data being made safe to render in the browser, and without permanently storing the user provided data. In some cases, the user provided data may never even leave the browser (see DOM Based XSS next).

DOM Based XSS (AKA Type-0)

As defined by Amit Klein, who published the first article about this issue [1], DOM Based XSS is a form of XSS where the entire tainted data flow from source to sink takes place in the browser, i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser. For example, the source (where malicious data is read) could be the URL of the page (e.g., document.location.href), or it could be an element of the HTML, and the sink is a sensitive method call that causes the execution of the malicious data (e.g., document.write).

https://owasp.org/www-community/Types_of_Cross-Site_Scripting

Most Commons Application Attacks - Injection

The Open Web Application Security Project (OWASP) in their latest Top Ten research named injection flaws as **the highest risk factor for websites**. The SQL injection method is the most popular practice used by cyber criminals in this category.

The injection attack methods target the website and the server's database directly. When executed, the attacker inserts a piece of code that reveals hidden data and user inputs, enables data modification and generally compromises the application.

Protecting your website against injection-based attacks mainly comes down to how well you've built your codebase. For example, the number one way to mitigate a SQL injection risk is to always use parameterized statements where available, among other methods. Furthermore, you can consider using a third-party authentication workflow to out-source your database protection.

https://www.tripwire.com/state-of-security/featured/most-common-website-security-attacks-and-how-to-protect-yourself/

Most Commons Application Attacks – Unvalidated Redirects and Forwards

This category of vulnerabilities is used in phishing attacks in which the victim is tricked into navigating to a malicious site. Attackers can manipulate the URLs of a trusted site to redirect to an unwanted location.

https://securityintelligence.com/the-10-most-common-application-attacks-in-action/

Most Commons Application Attacks – SQL Injection

An SQL injection attack is when attackers inject malicious SQL scripts[1] into a web application to gain access to the database stored in the server. A common way for hackers to do that is by injecting hidden SQL queries[2] in web forms (e.g. login form). Usually, when a user inputs their information in the form and hits the “login” button, an SQL query would be sent to the database to request that user’s information. However, when hackers inject a malicious SQL query, they could request all kinds of data from the database. By then, the hacker would be able to easily view, change, or delete data and potentially paralyze the entire system from functioning. Since most web applications have databases stored in their servers, these applications become attractive targets for SQL injection, leading to breaches of sensitive information.

https://www.pentasecurity.com/blog/top-7-common-types-cyberattacks-web-applications/

SQL Injection - Types

In-band SQLi

The attacker uses the same channel of communication to launch their attacks and to gather their results. In-band SQLi's simplicity and efficiency make it one of the most common types of SQLi attack. There are two sub-variations of this method:

- **Error-based SQLi**—the attacker performs actions that cause the database to produce error messages. The attacker can potentially use the data provided by these error messages to gather information about the structure of the database.
- **Union-based SQLi**—this technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database to get a single HTTP response. This response may contain data that can be leveraged by the attacker.

https://www.imperva.com/learn/application-security/sql-injection-sqli/

SQL Injection – Types 2

Inferential (Blind) SQLi

The attacker sends data payloads to the server and observes the response and behavior of the server to learn more about its structure. This method is called blind SQLi because the data is not transferred from the website database to the attacker, thus the attacker cannot see information about the attack in-band.

Blind SQL injections rely on the response and behavioral patterns of the server so they are typically slower to execute but may be just as harmful. Blind SQL injections can be classified as follows:

•**Boolean**—that attacker sends a SQL query to the database prompting the application to return a result. The result will vary depending on whether the query is true or false. Based on the result, the information within the HTTP response will modify or stay unchanged. The attacker can then work out if the message generated a true or false result.

•**Time-based**—attacker sends a SQL query to the database, which makes the database wait (for a period in seconds) before it can react. The attacker can see from the time the database takes to respond, whether a query is true or false. Based on the result, an HTTP response will be generated instantly or after a waiting period. The attacker can thus work out if the message they used returned true or false, without relying on data from the database.

https://www.imperva.com/learn/application-security/sql-injection-sqli/

SQL Injection – Types 3

Out-of-band SQLi

The attacker can only carry out this form of attack when certain features are enabled on the database server used by the web application. This form of attack is primarily used as an alternative to the in-band and inferential SQLi techniques.

Out-of-band SQLi is performed when the attacker can't use the same channel to launch the attack and gather information, or when a server is too slow or unstable for these actions to be performed. These techniques count on the capacity of the server to create DNS or HTTP requests to transfer data to an attacker.

https://www.imperva.com/learn/application-security/sql-injection-sqli/

Most Commons Application Attacks – Path Traversal

A path traversal (or directory traversal) attack is an application attack that targets the root directory of an application. Normally a result of a manipulated dot-slash sequence, path traversal attacks trick applications into allowing access into server files where all of the information within a system rests. Accessed data can include user credentials, access tokens, and even entire system backups that hold everything from sensitive data to system access controls.

https://www.contrastsecurity.com/knowledge-hub/glossary/application-attacks

Most Commons Application Attacks – Session Hijacking

A session hijacking attack tampers with session IDs. This unique ID is used to label a user's time online, keeping track of all activity for faster and more efficient future logins. Depending on the strength of the session ID, attackers could capture and manipulate the session ID, launching a session hijacking attack. If successful, attackers will have access to all information passed through the server for that particular session, getting ahold of user credentials to access personal accounts.

https://www.contrastsecurity.com/knowledge-hub/glossary/application-attacks

Most Commons Application Attacks – CSRF

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

https://owasp.org/www-community/attacks/csrf

Most Commons Application Attacks – DDoS

The DDoS attack alone doesn't allow the malicious hacker to breach the security but will temporarily or permanently render the site offline. Kaspersky Lab's IT Security Risks Survey in 2017 concluded that a single DDoS attack costs small businesses $123K and large enterprises $2.3M on average.

The DDoS attack aims to overwhelm the target's web server with requests, making the site unavailable for other visitors. A botnet usually creates a vast number of requests, which is distributed among previously infected computers. Also, DDoS attacks are often used together with other methods; the former's goal is to distract the security systems while exploiting a vulnerability.

Protecting your site against a DDoS attack is generally multi-faceted. First, you need to mitigate the peaked traffic by using a Content Delivery Network (CDN), a load balancer and scalable resources. Secondly, you also need to deploy a Web Application Firewall in case the DDoS attack is concealing another cyberattack method, such as an injection or XSS.

https://www.tripwire.com/state-of-security/featured/most-common-website-security-attacks-and-how-to-protect-yourself/

Most Commons Application Attacks – IDOR

Insecure **D**irect **O**bject **R**eference (called **IDOR** from here) occurs when a application exposes a reference to an internal implementation object. Using this way, it reveals the real identifier and format/pattern used of the element in the storage backend side. The most common example of it (although is not limited to this one) is a record identifier in a storage system (database, filesystem and so on).

https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html

Most Commons Application Attacks – CRLF

The term CRLF refers to Carriage Return (ASCII 13, \r) Line Feed (ASCII 10, \n). They're used to note the termination of a line, however, dealt with differently in today's popular Operating Systems. For example: in Windows both a CR and LF are required to note the end of a line, whereas in Linux/UNIX a LF is only required. In the HTTP protocol, the CR-LF sequence is always used to terminate a line.

A CRLF Injection attack occurs when a user manages to submit a CRLF into an application. This is most commonly done by modifying an HTTP parameter or URL.

https://owasp.org/www-community/vulnerabilities/CRLF_Injection

Most Commons Application Attacks – Race Condition

In any computing system, there are some tasks that need to be completed in a specific order. For example, before allowing someone to log in, a security system first receives their username and password and then checks it against a database before allowing access. Attackers can exploit this fact by interfering with processes to access secure areas and content in what's known as a race condition attack.

Race condition attacks (also called Time of Check to Time of Use, or TOCTTOU attacks) take advantage of the need that computing systems must execute some tasks in a specific sequence. In any such sequence, there is a small period of time when the system has carried out the first task but not started on the second. If this period is long enough or the attacker is lucky and knowledgeable, a race condition vulnerability exists where an attacker can trick the system into carrying out unauthorized actions in addition to its normal processes.

https://www.veracode.com/security/race-condition

Most Commons Application Attacks – Insecure Deserialization

Insecure deserialization is when user-controllable data is deserialized by a website. This potentially enables an attacker to manipulate serialized objects in order to pass harmful data into the application code.

It is even possible to replace a serialized object with an object of an entirely different class. Alarmingly, objects of any class that is available to the website will be deserialized and instantiated, regardless of which class was expected. For this reason, insecure deserialization is sometimes known as an "object injection" vulnerability.

An object of an unexpected class might cause an exception. By this time, however, the damage may already be done. Many deserialization-based attacks are completed **before** deserialization is finished. This means that the deserialization process itself can initiate an attack, even if the website's own functionality does not directly interact with the malicious object. For this reason, websites whose logic is based on strongly typed languages can also be vulnerable to these techniques.

https://portswigger.net/web-security/deserialization

Common Reasons for Existence of Application Vulnerabilities

- Most of the software development related curriculum often do not address security issues
- No proper guidance provided to relevant stakeholders at different phases of the project development
- Failure to gather application security requirements in the inception phase
- Improper application of security principles in the design phase
- Insecure coding techniques give space to various vulnerabilities
- Lack of security testing in the testing phase
- Security negligence in the deployment phase

Common Reasons for Existence of Application Vulnerabilities

An application vulnerability is a system flaw or weakness in an application that could be exploited to compromise the security of the application. Once an attacker has found a flaw, or application vulnerability, and determined how to access it, the attacker has the potential to exploit the application vulnerability to facilitate a cyber crime. These crimes target the confidentiality, integrity, or availability (known as the “CIA triad”) of resources possessed by an application, its creators, and its users. Attackers typically rely on specific tools or methods to perform application vulnerability discovery and compromise. According to Gartner Security, the application layer currently contains 90% of all vulnerabilities.

https://www.veracode.com/security/application-security-vulnerability-code-flaws-insecure-code

https://www.toptal.com/security/10-most-common-web-security-vulnerabilities

Most Commons Application Attacks – Failure to Restrict URL

If your application fails to appropriately restrict URL access, security can be compromised through a technique called forced browsing. Forced browsing can be a very serious problem if an attacker tries to gather sensitive data through a web browser by requesting specific pages, or data files.

Using this technique, an attacker can bypass website security by accessing files directly instead of following links. This enables the attacker to access data source files directly instead of using the web application. The attacker can then guess the names of backup files that contain sensitive information, locate and read source code, or other information left on the server, and bypass the "order" of web pages.

Simply put, Failure to Restrict URL Access occurs when an error in access-control settings results in users being able to access pages that are meant to be restricted or hidden. This presents a security concern as these pages frequently are less protected than pages that are meant for public access, and unauthorized users are able to reach the pages anonymously. In many cases, the only protection used for hidden or restricted pages is not linking to the pages or not publicly showing links to them.

https://www.veracode.com/security/failure-restrict-url-access

3W's Application Security

Why	Why should we care about application security?
Due to its globally accessible nature, applications are becoming more popular targets for attackers to compromise an organization's security	

What	What do we need for application security ?
A constant security vigilance at various phase of the application development lifecycle	

Who	Who is responsible for application security?
Managers, Architects, Developers, Testers, and Administrators	

Most Commons Application Attacks – XXE

XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data. It often allows an attacker to view files on the application server filesystem, and to interact with any back-end or external systems that the application itself can access.

In some situations, an attacker can escalate an XXE attack to compromise the underlying server or other back-end infrastructure, by leveraging the XXE vulnerability to perform server-side request forgery (SSRF) attacks.

https://portswigger.net/web-security/xxe

Most Commons Application Attacks – SSRF

Server-side request forgery (also known as SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing.

In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.

https://portswigger.net/web-security/ssrf

Most Commons Application Attacks – Command Injection

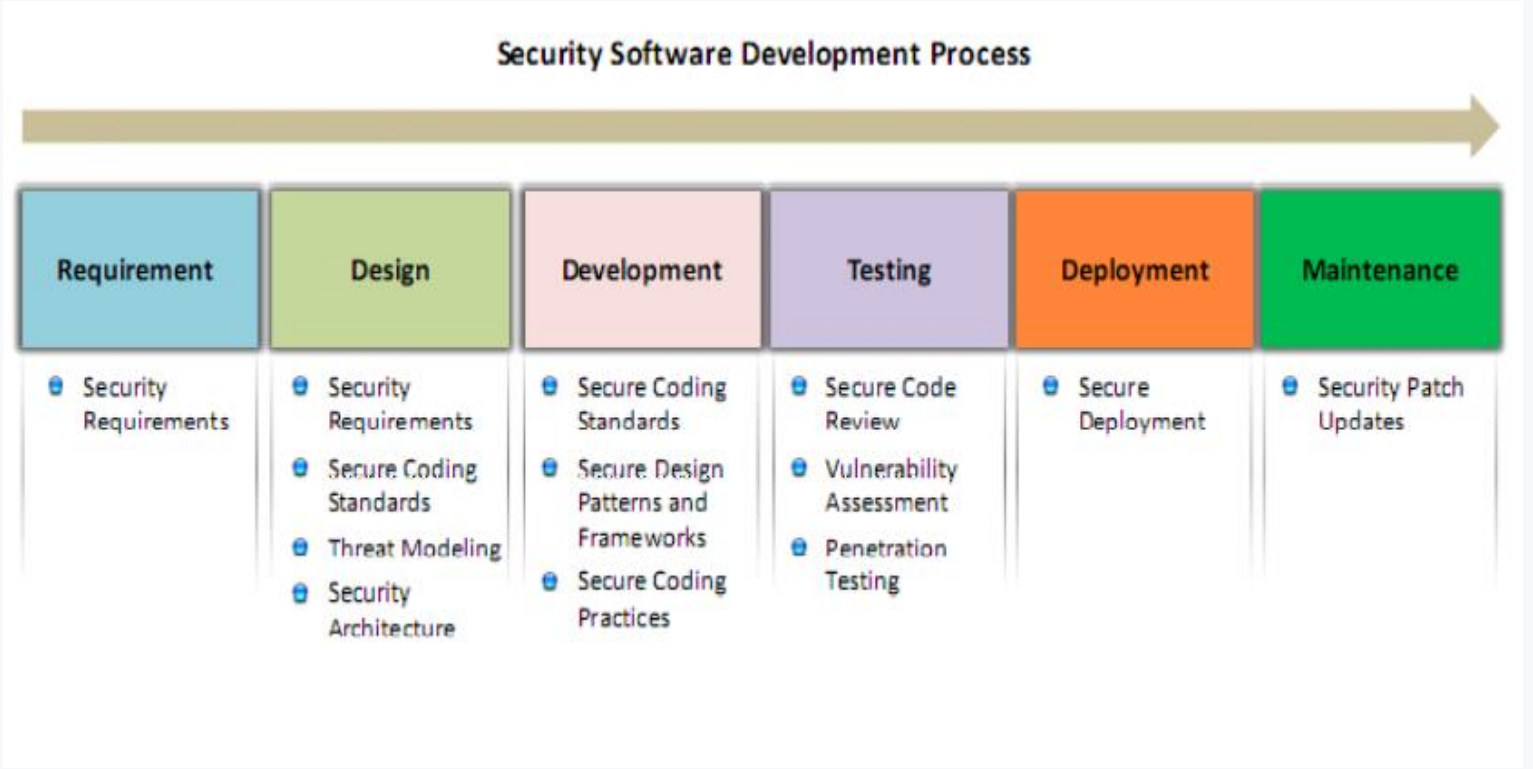
Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

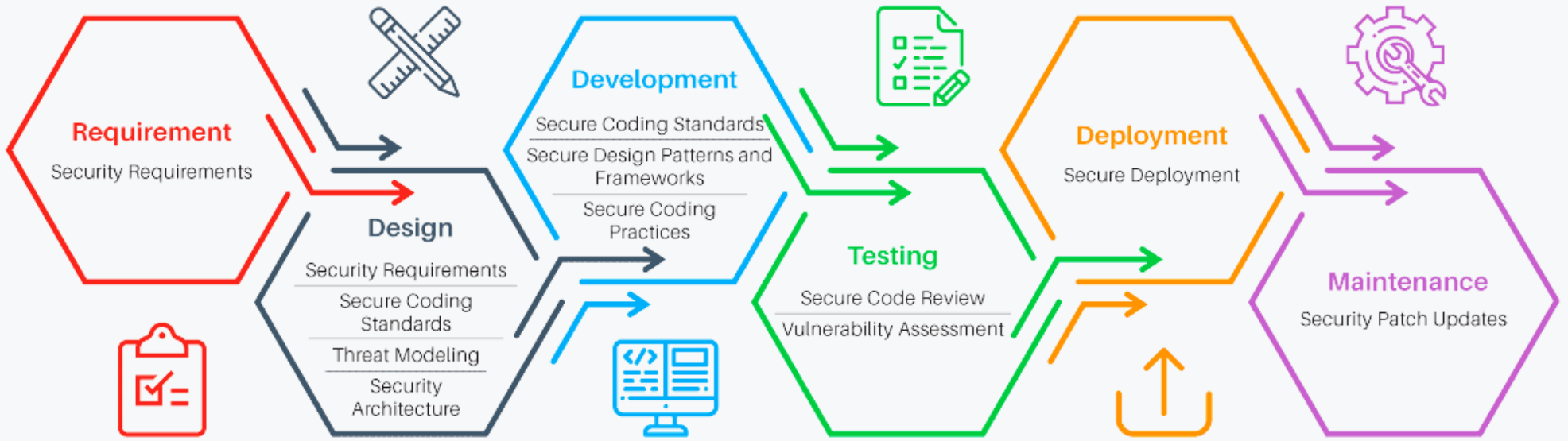
This attack differs from Code Injection, in that code injection allows the attacker to add their own code that is then executed by the application. In Command Injection, the attacker extends the default functionality of the application, which execute system commands, without the necessity of injecting code.

https://owasp.org/www-community/attacks/Command_Injection


APPLICATION
SECURITY
https://www.linkedin.com/in/joas-antonio-dos-santos

Security Software Development Process




Requirement
Security Requirements
Design
Security Requirements
Secure Coding Standards
Threat Modeling
Security Architecture
Development
Secure Coding Standards
Secure Design Patterns and Frameworks
Secure Coding Practices
Testing
Secure Code Review
Vulnerability Assessment
Deployment
Secure Deployment
Maintenance
Security Patch Updates

SDLC

OWASP TOP 10

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1–Injection	→	A1:2017–Injection
A2–Broken Authentication and Session Management	→	A2:2017–Broken Authentication
A3–Cross-Site Scripting (XSS)	↘	A3:2017–Sensitive Data Esposure
A4–Insecure Direct Object References [Merged+A7]	U	A4:2017–XML External Entities (XXE) [NEW]
A5–Security Misconfiguration	↘	A5:2017–Broken Access Control [Merged]
A6–Sensitive Data Exposure	↗	A6:2017–Security Misconfiguration
A7–Missing Function Level Access Contr [Merged+A4]	U	A7:2017–Cross-Site Scripting (XSS)
A8–Cross-Site Request Forgery (CSRF)	✕	A8:2017–Insecure Deserialization [NEW, Community]
A9–Using Components with Known Vulnerabilities	→	A9:2017–Using Components with Known Vulnerabilities
A10–Unvalidated Redirects and Forwards	✕	A10:2017–Insufficient Logging & Monitoring [NEW, Community]

https://www.synopsys.com/glossary/what-is-owasp-top-10.html

WASC Threat

Attacks		Weaknesses
Abuse of Functionality	Path Traversal	Application Misconfiguration
Brute Force	Predictable Resource Location	Directory Indexing
Buffer Overflow	Remote File Inclusion (RFI)	Improper File system Permissions
Content Spoofing	Routing Detour	Improper Input Handling
Credential/Session Prediction	Session Fixation	Improper Output Handling
Cross-Site Scripting	SOAP Array Abuse	Information Leakage
Cross-Site Request Forgery	SSI Injection	Insecure Indexing
Denial of Service	SQL Injection	Insufficient Anti-automation
Fingerprinting	URL Redirector Abuse	Insufficient Authentication
Format String	XPath Injection	Insufficient Authorization
HTTP Response Smuggling	XML Attribute Blowup	Insufficient Password Recovery
HTTP Response Splitting	XML External Entities	Insufficient Process Validation
HTTP Request Smuggling	XML Entity Expansion	Insufficient Session Expiration
Integer Overflows	XML Injection	Insufficient Transport Layer Protection
LDAP Injection	XQuery Injection	Server Misconfiguration
Mail Command Injection		
Null Byte Injection		
OS Commanding		

SAMM

Software Assurance Maturity Model
Our mission is to provide an **effective and measurable** way for you to analyze and improve your **secure development lifecycle**. SAMM supports the complete software lifecycle and is **technology and process agnostic**. We built SAMM to be **evolutive and risk-driven** in nature, as there is no single recipe that works for all organizations.

https://owasp.org/www-project-samm/

BSIMM

The Building Security In Maturity Model (BSIMM, pronounced “bee simm”) is a study of existing software security initiatives. By quantifying the practices of many different organizations, we can describe the common ground shared by many as well as the variations that make each unique.
BSIMM is not a how-to guide, nor is it a one-size-fits-all prescription. Instead, it is a reflection of software security.

https://www.bsimm.com/about.html

BSIMM vs SAMM

BSIMM	OpenSAMM
Descriptive model	Prescriptive model
Has 12 security practices consisting of 112 activities	Has 12 security practices consisting of 72 activities
Based on things actually followed in an organization	Based on certain list of activities developed for software security
Has an active community that enables the organizations to understand the security features followed in other organizations	Do not have any active community activities

BSIMM vs SAMM

BSIMM	OpenSAMM
Descriptive model	Prescriptive model
Has 12 security practices consisting of 112 activities	Has 12 security practices consisting of 72 activities
Based on things actually followed in an organization	Based on certain list of activities developed for software security
Has an active community that enables the organizations to understand the security features followed in other organizations	Do not have any active community activities

Security Requeriment

Have you ever heard the old saying “You get what you get and you don’t get upset”? While that may apply to after-school snacks and birthday presents, it shouldn’t be the case for software security. Software owners don’t just accept any new software features that are deployed; features must go through a strategic process of critique, justification, and analysis before being deployed. Your teams should treat security with the same attention to detail. After all, secure software doesn’t just happen out of nowhere—it has to be a requirement of the strategic development process. To deploy secure software effectively, you need clear, consistent, testable, and measurable software security requirements.

https://www.synopsys.com/blogs/software-security/software-security-requirements/

Good Requeriment Security

Specific	Wording of the software requirement should be clear and precise. It should not be vague and all-encompassing
Measurable/Testable	There should be a dear way to test whether the specific requirement was met or not
Actionable	Developer should get clear understanding of what they exactly need to do to satisfy the requirements of the client
Realistic	It should be implementable in real time considering all the constraints
Timely	It should be of high priority when it is decided to implement

Types Security Requeriment

If you’re entrenched in the requirements or contracting world, you’re already aware of the basic kinds of requirements: functional, nonfunctional, and derived. Software security requirements fall into the same categories. Just like performance requirements define what a system has to do and be to perform according to specifications, security requirements define what a system has to do and be to perform securely.
When defining functional nonsecurity requirements, you see statements such as “If the scan button is pressed, the lasers shall activate and scan for a barcode.” This is what a barcode scanner needs to do. Likewise, a security requirement describes something a system has to do to enforce security. For example: “The cashier must log in with a magnetic stripe card and PIN before the cash register is ready to process sales.”

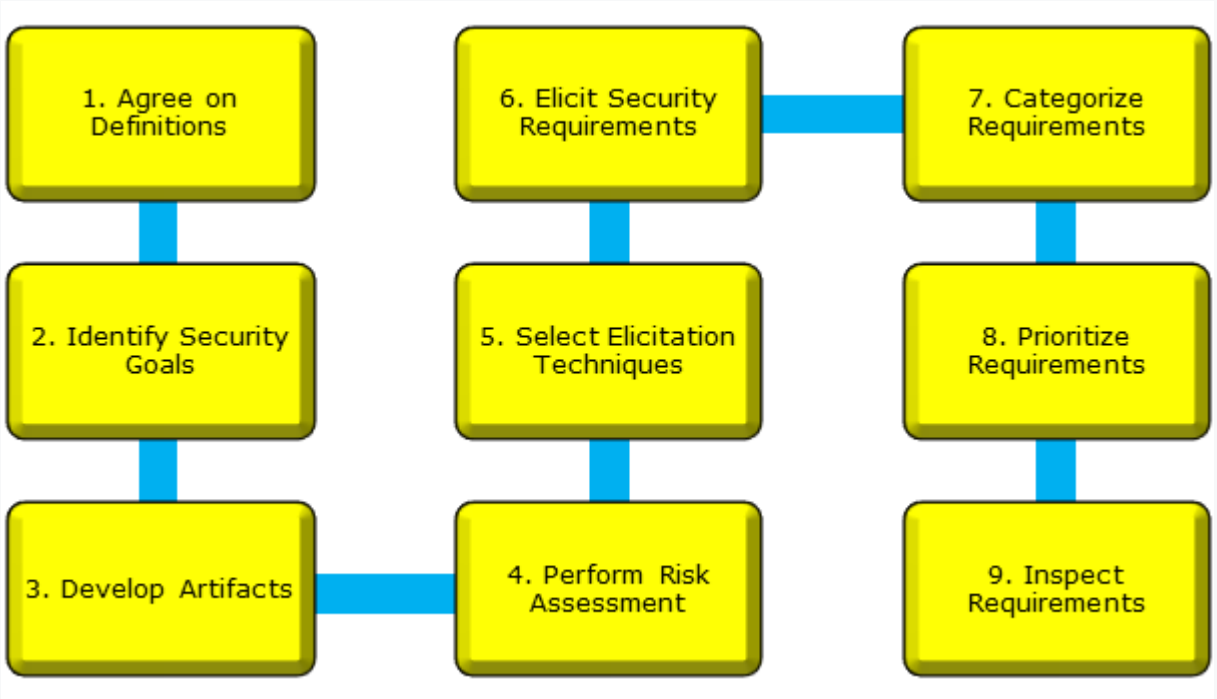
https://www.synopsys.com/blogs/software-security/software-security-requirements/

Types Security Requeriment

Functional requirements describe what a system has to do. So functional security requirements describe functional behavior that enforces security. Functional requirements can be directly tested and observed. Requirements related to access control, data integrity, authentication, and wrong password lockouts fall under functional requirements.
Nonfunctional requirements describe what a system has to be. These are statements that support auditability and uptime. Nonfunctional security requirements are statements such as “Audit logs shall be verbose enough to support forensics.” Supporting auditability is not a direct functionality requirement, but it supports auditability requirements from regulations that might apply.

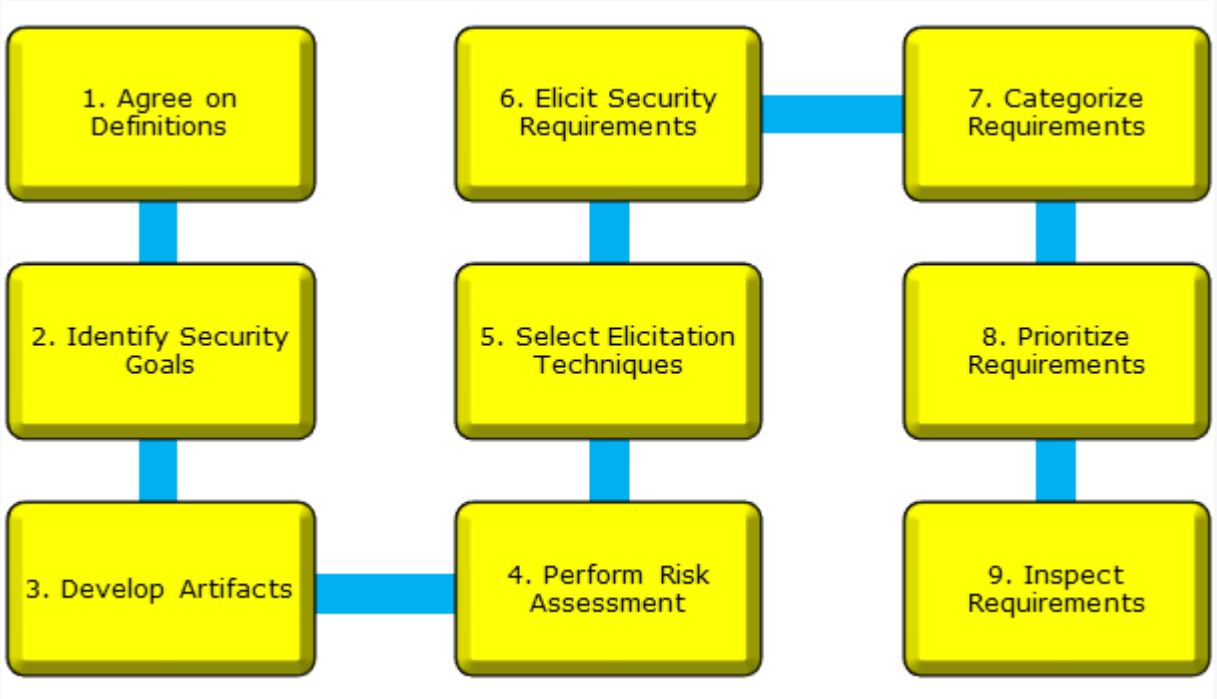
https://www.synopsys.com/blogs/software-security/software-security-requirements/

SRE Phases



https://www.softscheck.com/en/security-consultancy/security-requirements-engineering/

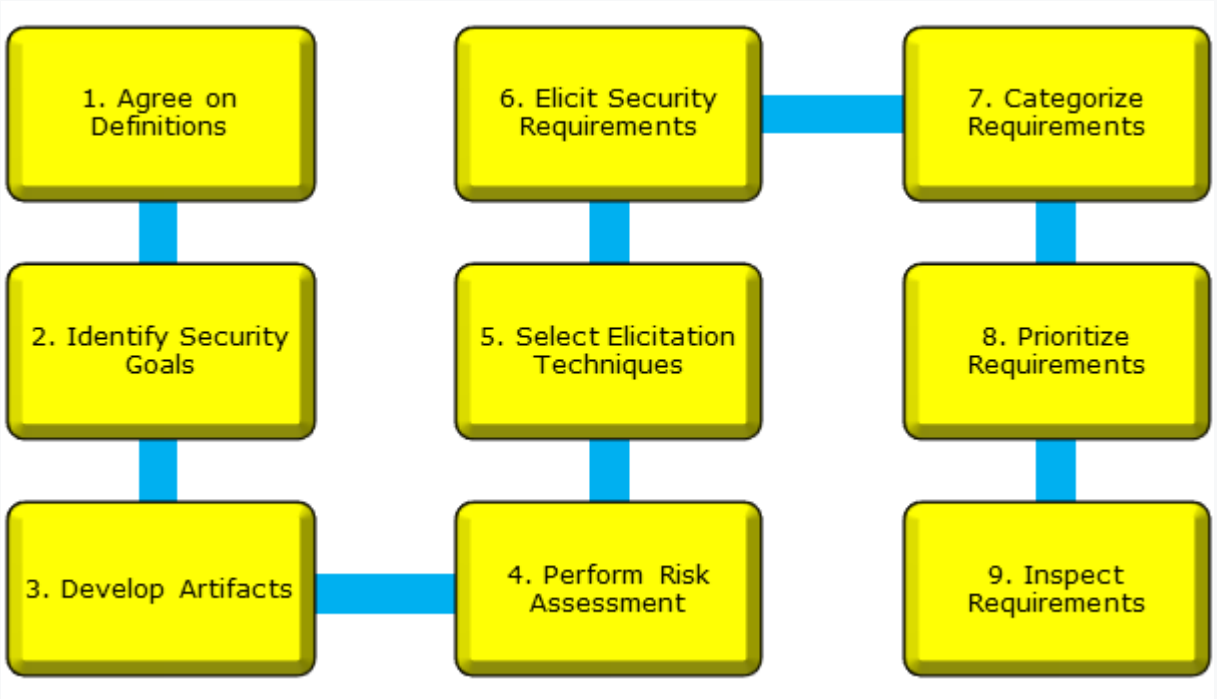
SRE Phases, Analysis and Priorization



https://www.softscheck.com/en/security-consultancy/security-requirements-engineering/

https://www.researchgate.net/publication/276284984_Security_Requirements_Engineering_Analysis_and_Prioritization

SRE Phases, Analysis and Priorization



https://www.softscheck.com/en/security-consultancy/security-requirements-engineering/

https://www.researchgate.net/publication/276284984_Security_Requirements_Engineering_Analysis_and_Prioritization

SRE Phases 2

Requirements Elicitation	Requirement engineer will not be able to identified all security requirements due to lack of knowledge of modern elicitation techniques or does not include all the relevant stakeholder during requirement phase
Requirements Analysis	The identified requirements are directly specified without any analysis or modeling
Requirements Specification	The identified requirements is not SMART
Requirements Management	The identified requirements is not prioritized or scheduled properly

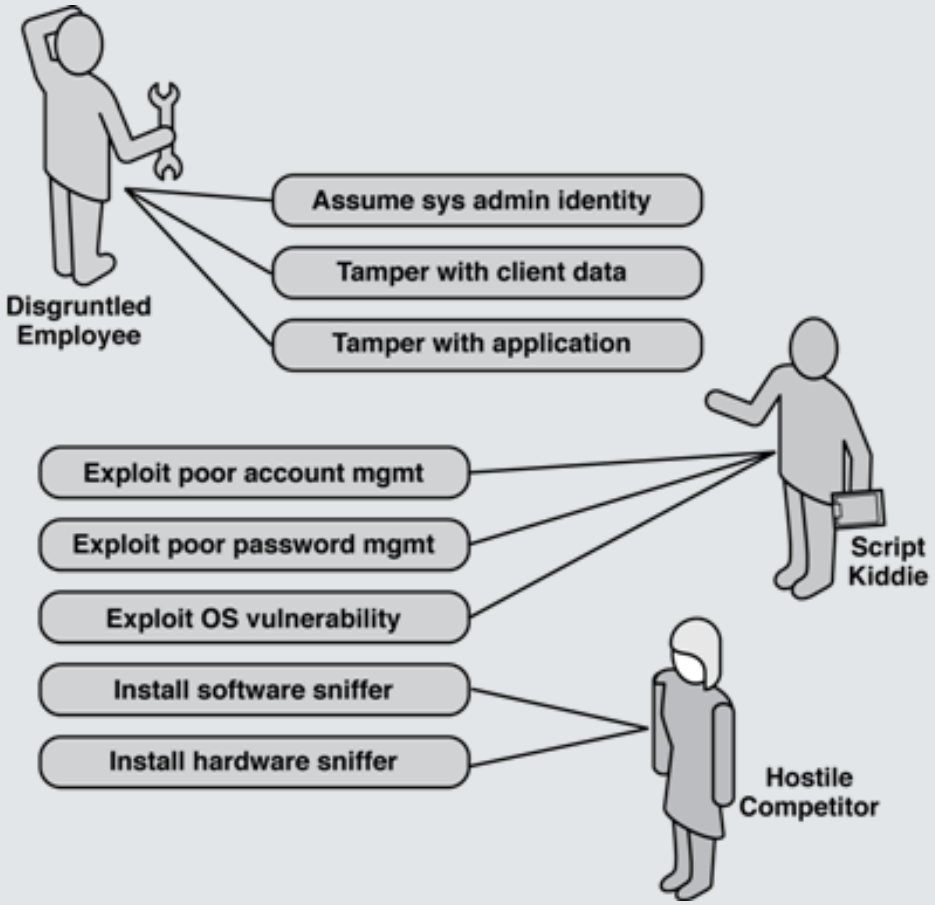
https://www.softscheck.com/en/security-consultancy/security-requirements-engineering/

https://www.researchgate.net/publication/276284984_Security_Requirements_Engineering_Analysis_and_Prioritization

Abuse Cases Application Security

https://cheatsheetseries.owasp.org/cheatsheets/Abuse_Case_Cheat_Sheet.html

https://www.synopsys.com/blogs/software-security/abuse-cases-can-drive-security-requirements/



SQUARE (System Quality Requeriments Enginering)

Requirements problems are the primary reason that projects are significantly over budget and past schedule, have significantly reduced scope, and deliver poor-quality applications that are little used once delivered, or are cancelled altogether.
One source of these problems is poorly expressed or analyzed quality requirements, such as security and privacy. Requirements engineering defects cost 10 to 200 times more to correct during implementation than if they are detected during requirements development. Moreover, it is difficult and expensive to significantly improve the security of an application after it is in its operational environment.
Security Quality Requirements Engineering (SQUARE) is a nine-step process that helps organizations build security, including privacy, into the early stages of the production lifecycle. Instructional materials are available for download that can be used to teach the SQUARE method.

https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484884

SQUARE (System Quality Requeriments Enginering) - Process

Number	Step	Input	Techniques	Participants	Output
1	Agree on definitions	Candidate definitions from IEEE and other standards	Structured interviews, focus group	Stakeholders, requirements engineer	Agreed to definitions
2	Identify assets and security goals	Definitions, Candidate goals, Business drivers, Policies and procedures, Examples	Facilitated work session, surveys, interviews	Stakeholders, requirement engineer	Assets and goals
3	Develop artifacts to support security requirements definition	Potential artifacts (e.g.: scenarios, misuse cases, templates, forms)	Work session	Requirements engineer	Needed artifacts: scenarios, misuse cases, models, templates, forms
4	Perform risk assessment	Misuse cases, scenarios, goals	Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including risk analysis	Requirements engineer, stakeholder, risk expert	Risk assessment results

OCTAVE

OCTAVE is a flexible and self-directed risk assessment methodology. A small team of people from the operational (or business) units and the IT department work together to address the security needs of the organization. The team draws on the knowledge of many employees to define the current state of security, identify risks to critical assets, and set a security strategy. It can be tailored for most organizations.

Unlike most other risk assessment methods the OCTAVE approach is driven by operational risk and security practices and not technology. It is designed to allow an organization to:

- Direct and manage information security risk assessments for themselves
- Make the best decisions based on their unique risks
- Focus on protecting key information assets
- Effectively communicate key security information

https://technology.ku.edu/octave-method-security-assessment

OCTAVE

The OCTAVE method is based on eight processes that are broken into three phases. In the higher education organizations, it is usually preceded by an exploratory phase (known as *Phase Zero*) to determine the criteria that will be used during the application of the Octave method.

The three phases of OCTAVE are:

- *Phase 1:* Develop initial security strategies
- *Phase 2:* Technological view — Identify infrastructure vulnerabilities
- *Phase 3*: Risk analysis — Develop security strategy and plans

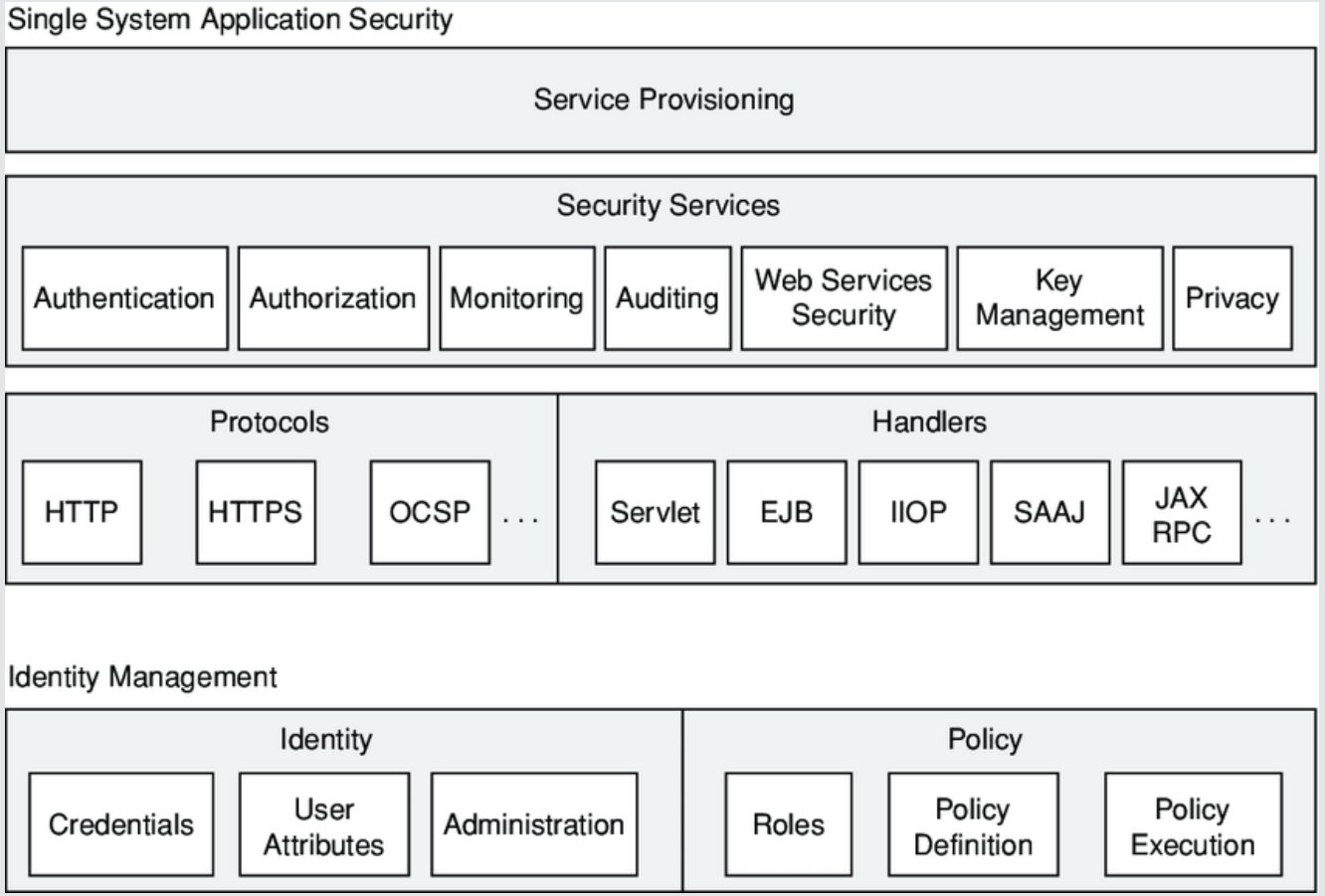
https://technology.ku.edu/octave-method-security-assessment

APPLICATION SECURITY - DESIGN

https://www.linkedin.com/in/joas-antonio-dos-santos

Security Design

https://www.researchgate.net/figure/10-Logical-security-framework-of-an-application-security-provider_fig2_284509993



Security Design - OWASP

https://patchstack.com/security-design-principles-owasp/

The OWASP Security Design Principles have been created to help developers build highly secure web applications.

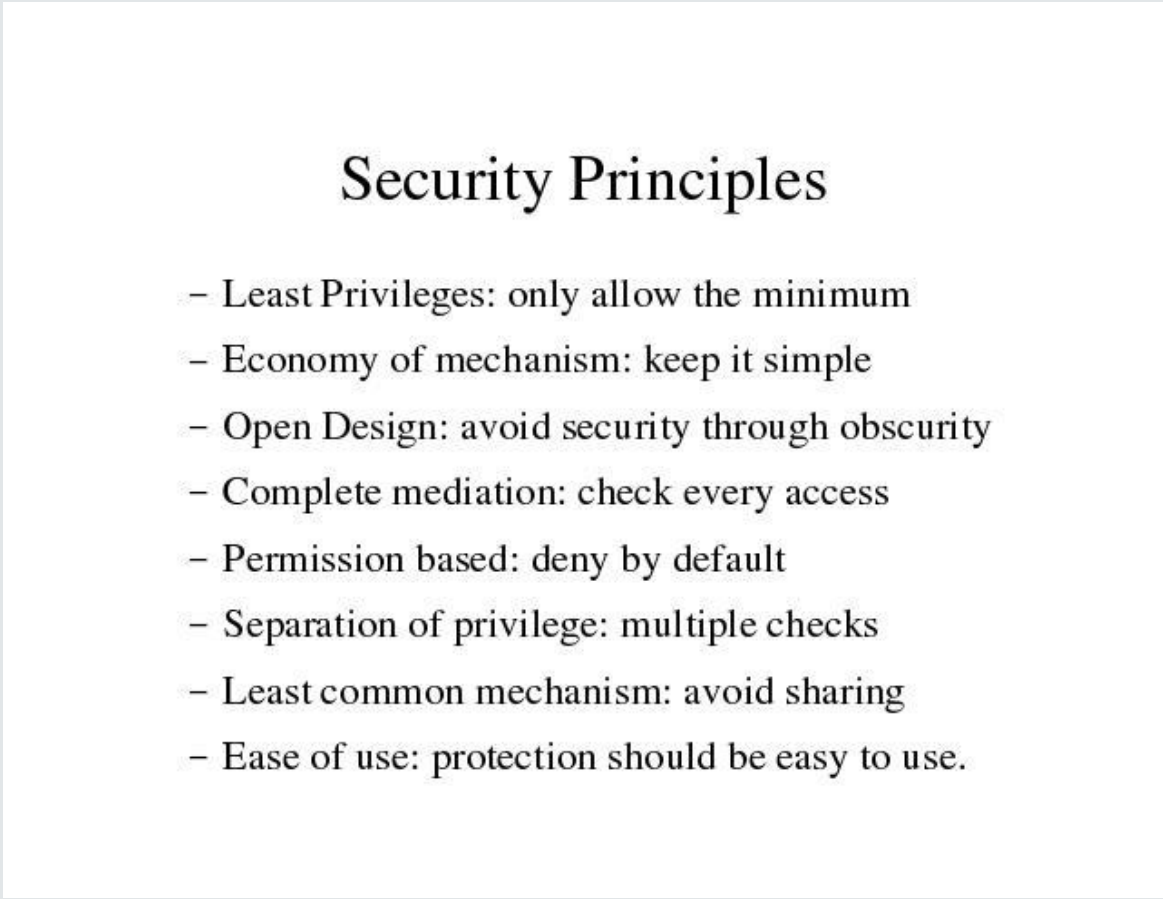
The OWASP security design principles are as follows:

Asset clarification
Before developing any security strategies, it is essential to identify and classify the data that the application will handle. OWASP suggests that programmers create security controls that are appropriate for the value of the data being managed. For example, an application processing financial information must have much tighter restrictions than a blog or web forum.

Understanding attackers
OWASP recommends that all security controls should be designed with the core pillars of information security in mind:
•Confidentiality – only allow access to data for which the user is permitted
•Integrity – ensure data is not tampered or altered by unauthorised users
•Availability – ensure systems and data are available to authorised users when they need it

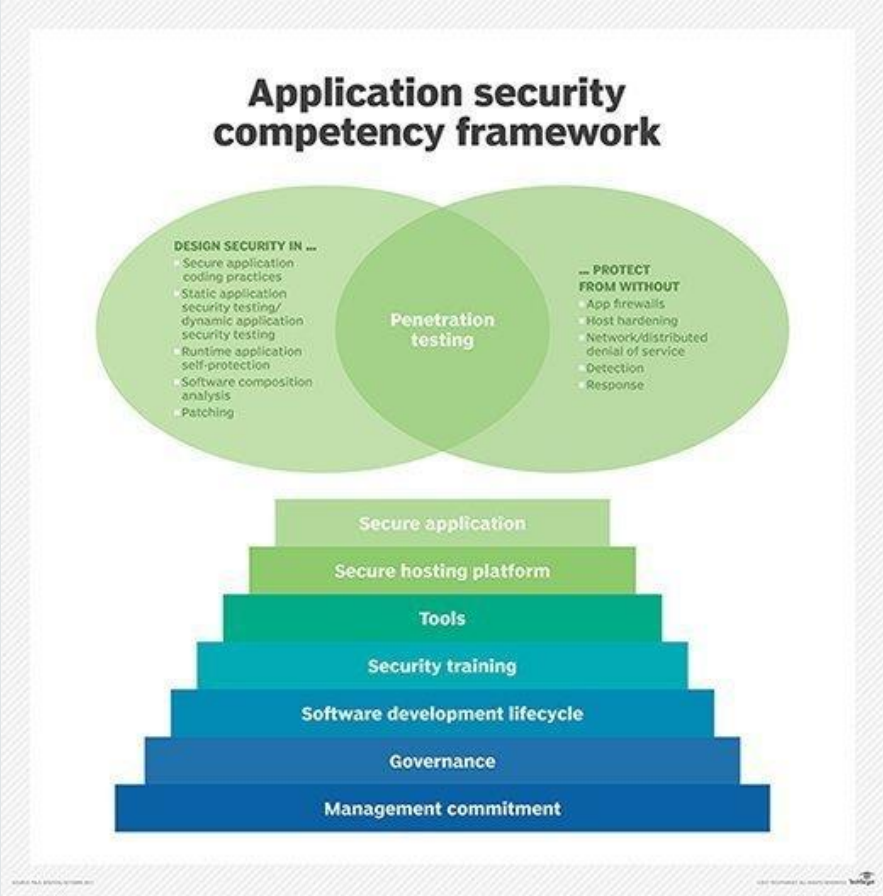
Security Principles

http://www.csun.edu/~jeffw/Courses/COMP424/Lectures/Lecture11/HTML/img39.html



Security Principles

https://searchsecurity.techtarget.com/feature/Security-for-applications-What-tools-and-principles-work



Fundamental Security Design Principles

https://binaryterms.com/fundamental-security-design-principles.html

The security design principles are considered while designing any security mechanism for a system. These principles are review to develop a secure system which prevents the security flaws and also prevents unwanted access to the system.

Below is the list of fundamental security design principles provided by the National Centres of Academic Excellence in Information Assurance/Cyber Defence, along with the U.S. National Security Agency and the U.S. Department of Homeland Security.

Fundamental Security Design Principles

https://binaryterms.com/fundamental-security-design-principles.html

1. Economy of Mechanism
2. Fail-safe Defaults
3. Complete Mediation
4. Open Design
5. Separation of Privilege
6. Least Privilege
7. Least Common Mechanism
8. Psychological Acceptability
9. Isolation
10. Encapsulation
11. Modularity
12. Layering
13. Least Astonishment

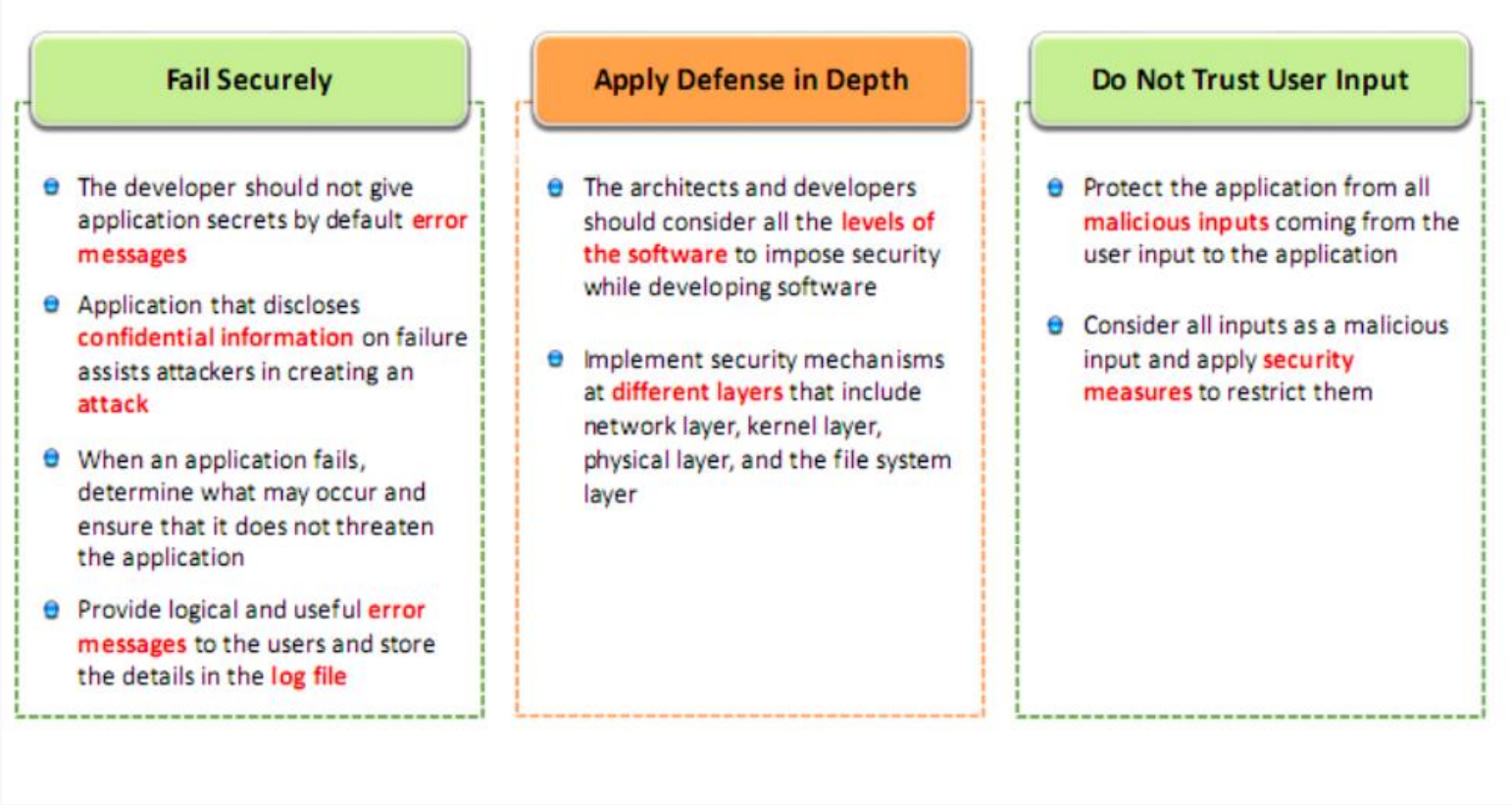
Security Design Principles

Fundamental Security Design Principles

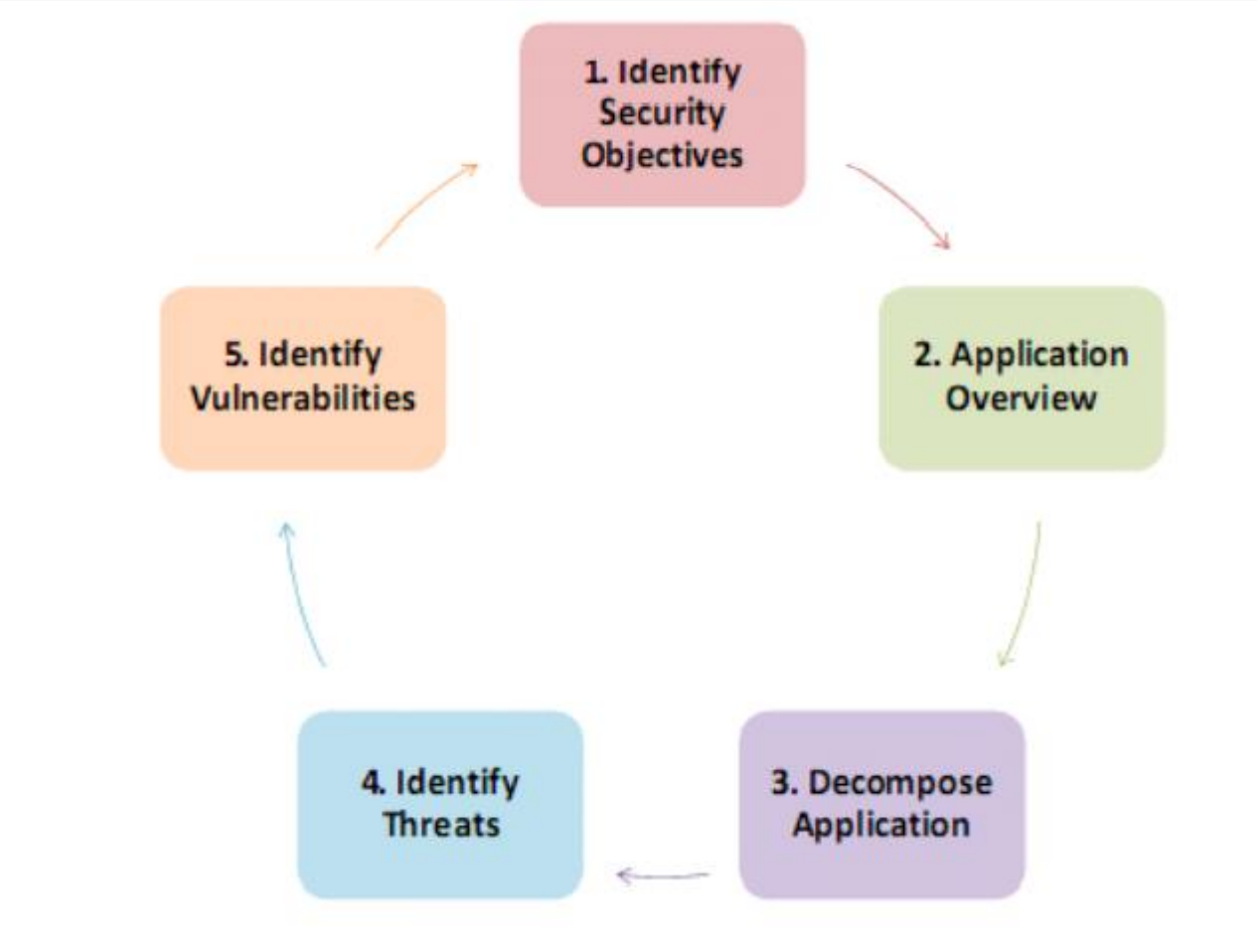
https://binaryterms.com/fundamental-security-design-principles.html

1. Economy of Mechanism
2. Fail-safe Defaults
3. Complete Mediation
4. Open Design
5. Separation of Privilege
6. Least Privilege
7. Least Common Mechanism
8. Psychological Acceptability
9. Isolation
10. Encapsulation
11. Modularity
12. Layering
13. Least Astonishment

Security Design Principles



Threat Model



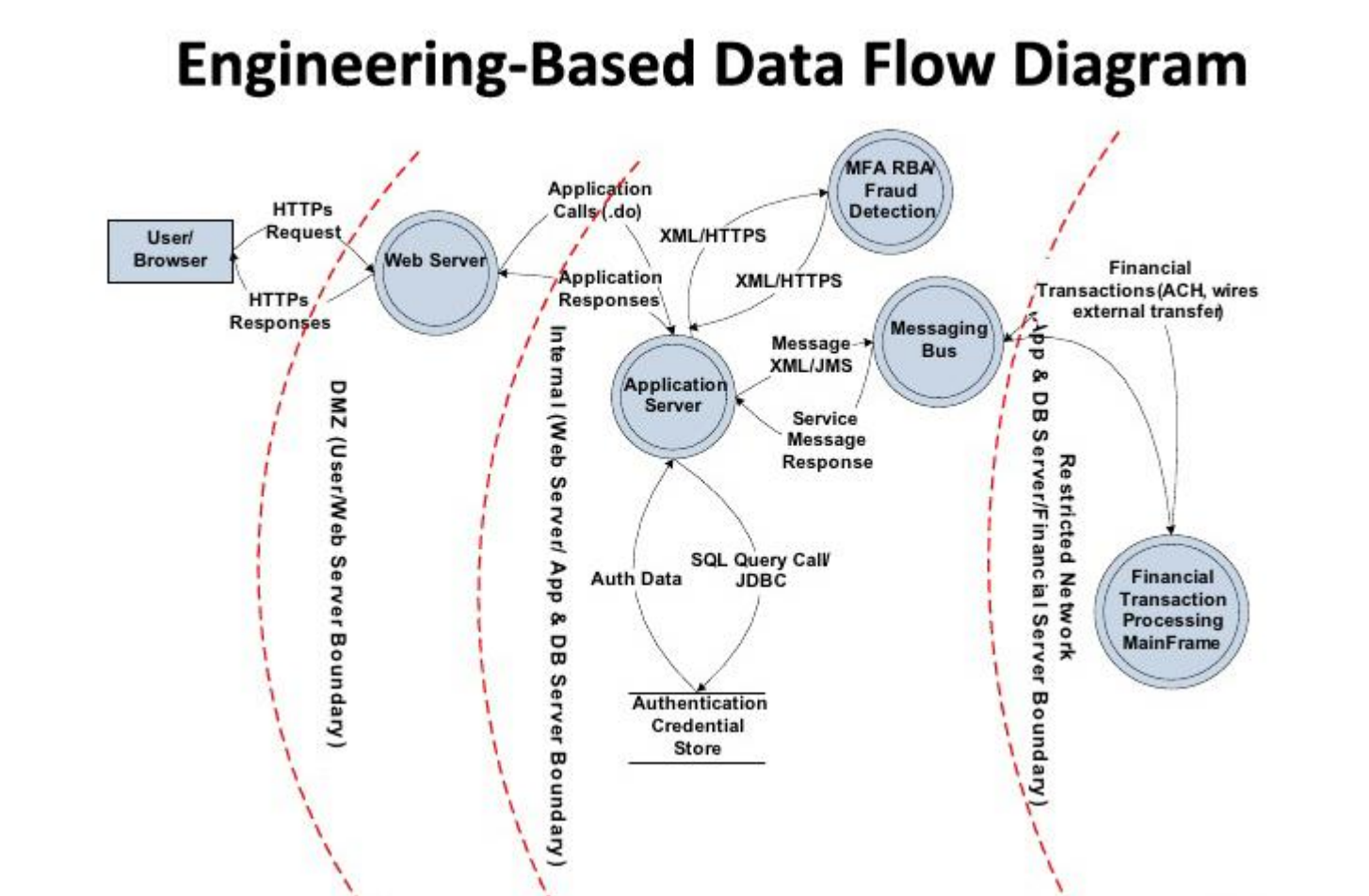
Application Security Mechanism

https://www.researchgate.net/figure/Applicable-Security-Mechanisms_tbl4_221095013

Mechanisms	*Types*	*Description*	*Related Threats*
Anti-Virus Solution	Add-on Application, System Add-on	Anti-virus solutions scan files, memory, SMS, MMS, emails and URLs Anti-virus solutions can prevent malwares and also prevent access to phishing site	T1, T6, (T8)
Firewall	System Modification	Firewall blocks and/or audit un-allowed connections from/to device Firewall can prevent network attacks by denying access to untrusted wireless network	T3, (T8)
Secure API	System Add-on	Secure API provide cryptographic functionalities for application developer Application developer can implement secure functionality using secure APIs	T1, T2, (T8)
Access Control	System Modification	Access control limits access of processes and user to resources and/or services Access control can limit risk from malicious/exploited application	T1, T7
Authentication	System Modification	User should be authenticated to use device Authentication process can prevent unauthorized use of device	T7
Spam Filter	System Add-on, Application Add-on	SPAM filtering applications blocks MMS, SMS, emails and calls from unwanted origin SPAM filtering applications can prevent SPAM	T1
Pre-Testing	System Modification	Pre-Testing guarantee applications and authorizes developer Pre-Testing can prevent malware and ensure security of applications	T1, T4, T5
Regular Update	System Modification	Regular update for platform and smartphone application	T5
Remote Access Control	System Modification	Remote access control includes remote configuration and management of smartphone(remote blocking, remote reset) When user lose his/her smartphone, remote access control can reduce damage by lost smartphone	T7

Application Security DFD

https://threatmodeler.com/data-flow-diagrams-process-flow-diagrams/



Application Security DFD

https://threatmodeler.com/data-flow-diagrams-process-flow-diagrams/

System engineers developed data flow diagrams to provide a high-level visualization of how an application works within a system to move, store and manipulate data. The intended use of DFDs was to provide engineers a way of efficiently communicating their structured system analysis. Security professionals added the concept of trust boundaries to DFDs in the early 2000s to make them more applicable for threat modeling.
Since then many attempts have been put forward by various groups to create a more mature DFD-based process, especially for development environments employing an Agile methodology. Despite the valiant and prolonged effort, DFDs fundamentally remain a means of communicating analysis of a structured system. Hence they have limited capacity to adequately address applications which are created for platform independence and deployed in a highly interconnected environment.

Application Security DFD

https://threatmodeler.com/data-flow-diagrams-process-flow-diagrams/

Furthermore, with DFDs, high volumes of documentation were the expected norm. This, of course, makes them unwieldy for Agile sprinting developers who minimize documentation and any other activity they deem non-productive. Without developer acceptance, organizations will find significant challenge scaling threat modeling processes enterprise-wide.

DFD-based threat modeling fundamentally looks at how data is designed to move through a system. The approach cannot, therefore, provide a means to inherently analyze how an application appears to a potential attacker. Since a DFD cannot analyze an application from the perspective of an attacker, any predictive capacity regarding possible attack vectors, entry points, or exfiltration points, requires significant speculation on the part of the user.

As applied to threat modeling, DFDs are typically used to identify broad categories – usually based on the STRIDE threat classification scheme – of potential threats such as elevation of privilege or Distributed Denial of Service. The list of threats identifiable through such methods is rather limited and provides a poor starting point for producing actionable outputs.

Application Security DFD

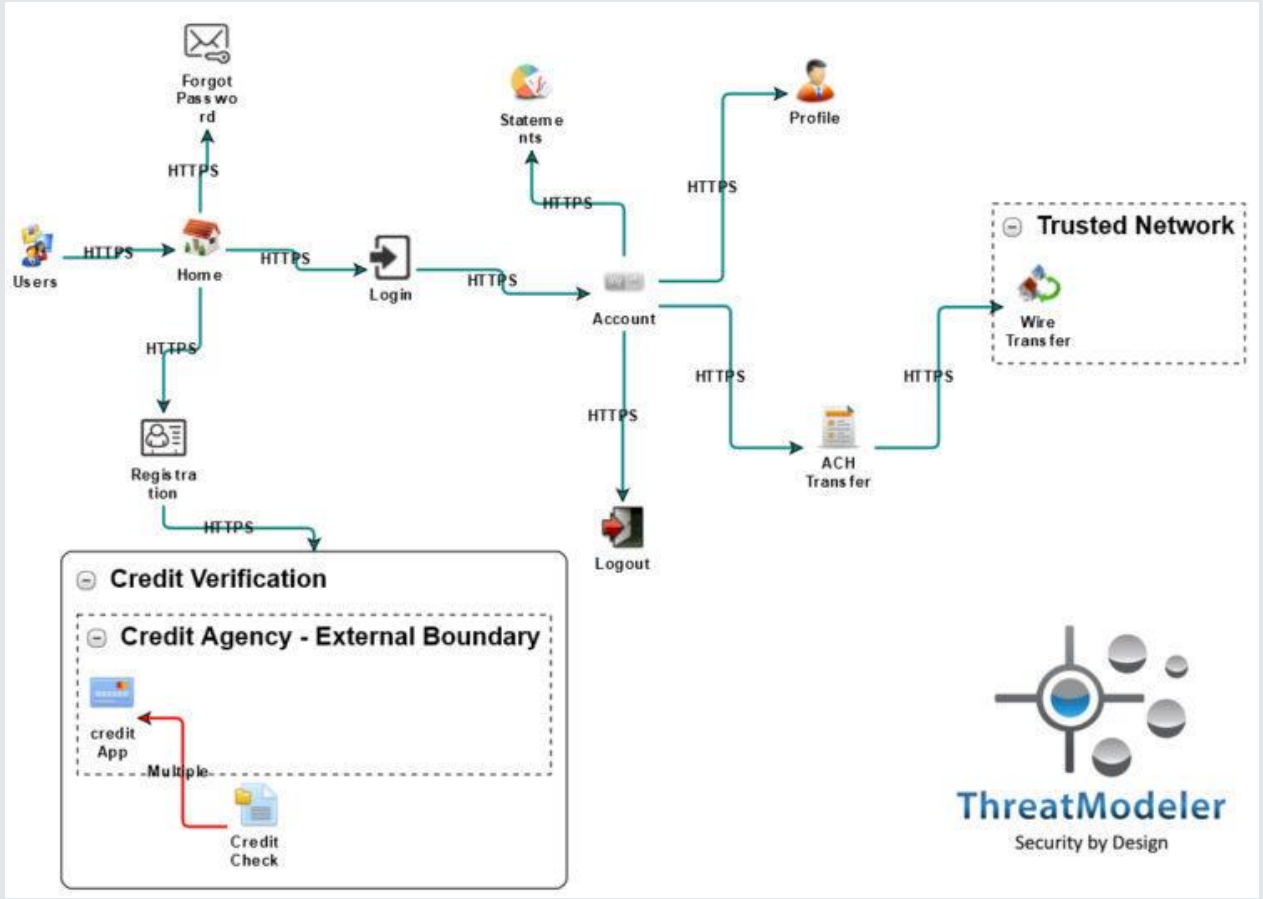
https://threatmodeler.com/data-flow-diagrams-process-flow-diagrams/

DFD-based threat modeling leaves a threat modeling practice with fundamental weaknesses:

- DFDs do not accurately represent the design and flow of an application
- They analyze the operational component and how the data is flowing, rather than on how users interact and move through the application features;
- Data flow diagrams are hard to understand because they require security expertise. The developer community does not embrace DVD-based threat models because they are vague, and complex
- DFD-based threat modeling has no standard approach – different people tend to create different threat models with entirely different outputs
- The DFD process is fundamentally focused on very high-level system issues. It cannot, therefore, to help developers understand the relevant threats and their mitigating controls

Application Security DFD – Process Flow

https://threatmodeler.com/data-flow-diagrams-process-flow-diagrams/



The advantages of utilizing process, or application flow diagrams

•Creating threat models with developer-level application details of communication protocols and employed coding elements intrinsically included allowing more efficiency identifying potential threats;
•Creation of a “process map,” showing how individuals move through an application. Security professionals and developers can then view the application from the attacker’s vantage, resulting in more efficiently prioritizing potential threats;
•An easy to understand threat model that promotes collaboration across all organizational stakeholders, regardless of an individual’s level of security expertise;
•Standardization of the threat modeling process resulting in consistent, actionable output regardless of who created the threat model.
Process flow diagrams are the result of a maturing threat modeling discipline. They genuinely allow incorporation of developers in the threat modeling process during the application design phase. This helps developers working within an Agile development methodology initially write secure code. The threat modeling initiative then becomes a means of enhancing the developer’s ability to sprint to production. This will significantly help the organization in scaling threat modeling processes.

CREATE A SECURITY PROFILE

- To create **security profile** for an application, focus on:
  - Various **design** and **implementation approaches** of an application which can be most susceptible to vulnerabilities
  - Various **areas of applications** which can be most susceptible to vulnerabilities
- Document the **security profile** for application depending on the area of vulnerabilities

STRIDE MODEL

STRIDE Threat Model

Spoofing identity

- Illegally accessing and then using another user's authentication information

Tampering with data

- Malicious modification
- Unauthorized changes

Repudiation

- Deny performing an malicious action
- Non-repudiation refers to the ability of a system to counter repudiation threats

Elevation of privilege

- Unprivileged user gains privileged access to compromise the system
- Effectively penetrated and become part of the trusted system

Denial of service

- Deny service to valid users
- Threats to system availability and reliability

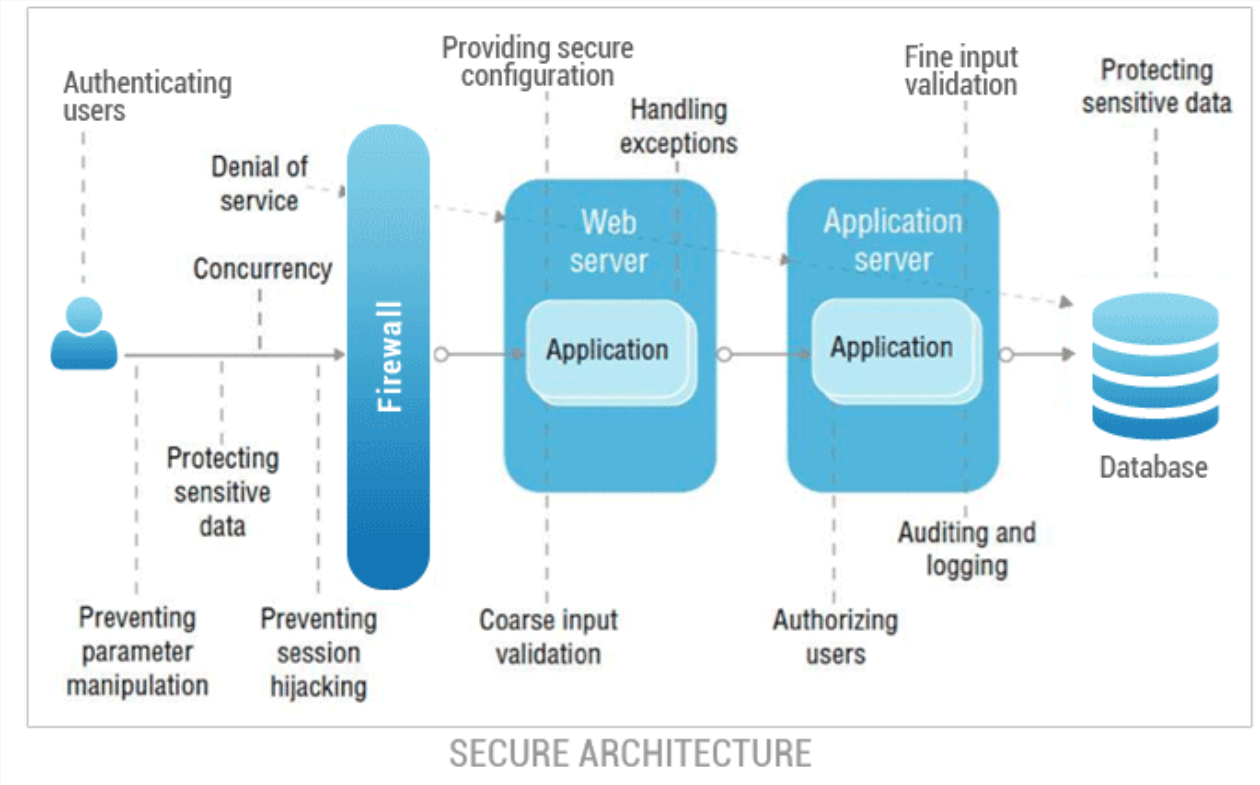
Information disclosure

- Exposure of information to individuals not supposed to access

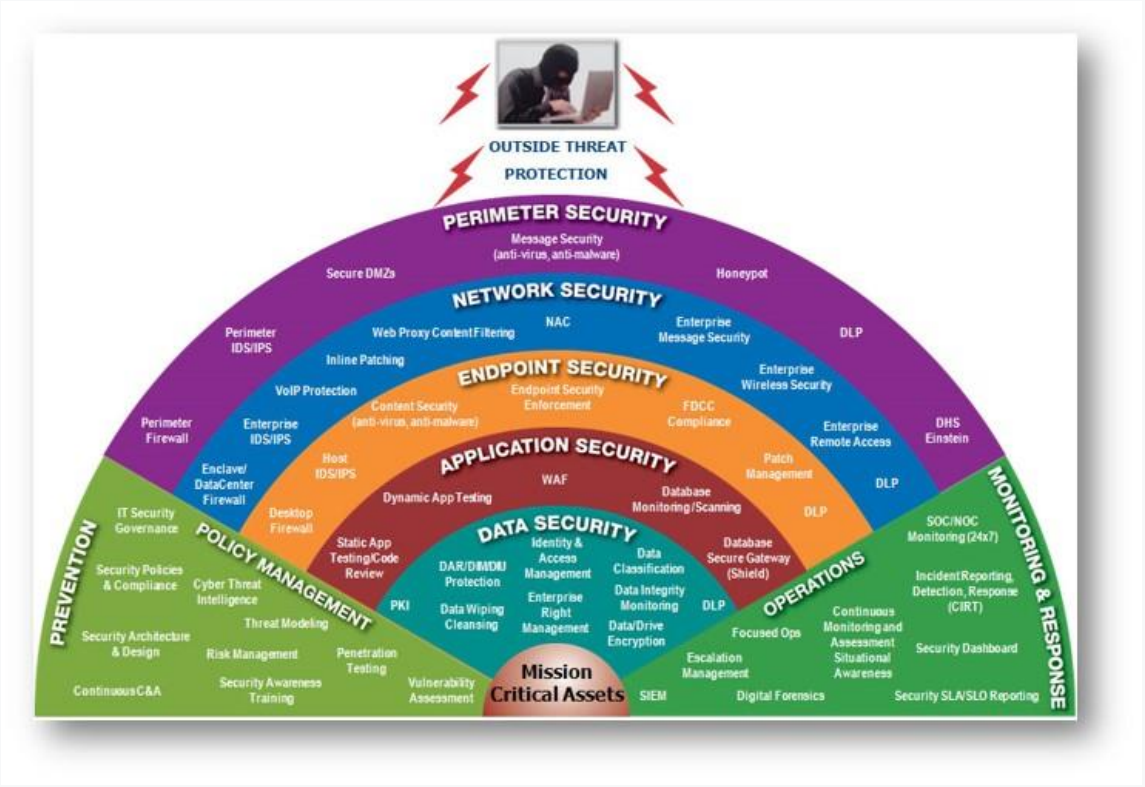
DREAD MODEL

Elements	*Description*	*Rationale*	*Evaluated*
Damage	*Aircraft computer systems, flight information screens, computer monitors for check-in at check-in counters, radio systems to stop working. More than 100 flights delayed. Over 400 thousand the airlines' customer database was stolen and made public on the internet*	*Infringement on availability requirement. Confidentiality of information is compromised*	*6 to 8 point*
Affected users	*More than 100 flights were affected, of which dozens of flights were delayed by 15 to 60 minutes. Staff and passengers checked in the flight in manually, using the portable speaker to announce the flight to the passengers*	*Admin, power users, group, user, and public. All five groups were affected*	*7 to 8 point*
Reproducibility	*The website of Vietnam Airlines has been re-activated, but according to security experts have not yet patched the vulnerabilities and the risk of recurrence*	*Moderate or simple*	*4 to 5*
Exploitability	*This is a sophisticated attack. Attackers used malicious code that not detected by antivirus software.*	*Expert Or Journeyman*	*8 to 9 point*
Discoverability	*According to Vietnam security experts, the vulnerability on the VNA eCommerce website have warned long before the incident*	*Easy or moderate*	*3 to 4 point*
Range of Risk Exposure score = 5.6 to 6.8 point means ***Important***			

DESIGN SECURITY APPLICATION ARCHITECTURE

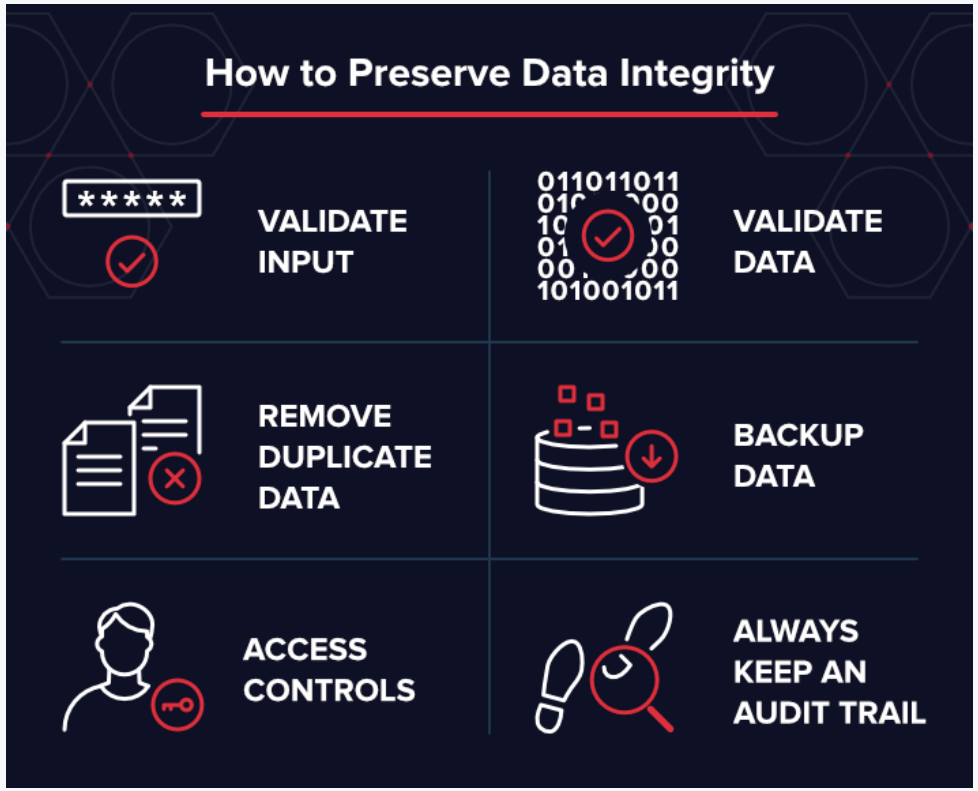


SECURE ARCHITECTURE



SECURITY DESIGN & TESTING

DATA INTEGRITY



https://www.varonis.com/blog/data-integrity/

Certifications and Courses

https://www.eccouncil.org/programs/application-security-training/

https://shehackspurple.ca/

https://www.pluralsight.com/

https://application.security/

https://www.securityinnovation.com/training/software-application-security-courses/

https://www.isc2.org/Certifications/CSSLP

http://elearnsecurity.com/

https://www.offensive-security.com/awae-oswe/