Application Security Introduction – Overview PT 2

JOAS ANTONIO

HTTPS://WWW.LINKEDIN.COM/IN/JOAS-ANTONIO-DOS-SANTOS

Reason for Existe of Application Vulnerabilities

- Most of the software development related curriculum often do not address security issues
- Failure to Gather Application Security requeriments in the Inception Phase
- Insecure Coding Techniques give space to various vulnerabilities
- Security Negligence in the Deployment Phase

Common Flaws due to Insecure Coding Techniques

- Improper Input Validation
- Insecure Cryptographic Storage
- Insufficient Transport Layer Protection
- Insecure Direct Object References
- Broken Authetication and Session Management
- Improper Handling
- Unvalidated Redirects and Forwards
- Failure to Restrict URL Access

Improper Input Validation

- The product receives input or data, but it does not validate or incorrectly validates that the input has the properties that are required to process the data safely and correctly.
- Input validation is a frequently-used technique for checking potentially dangerous inputs in order to ensure that the inputs are safe for processing within the code, or when communicating with other components. When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.

- https://cwe.mitre.org/data/definitions/20.html

Insecure Cryptographic Storage

- Insecure Cryptographic Storage is a common vulnerability that occurs when sensitive data is not stored securely. Insecure Cryptographic Storage isn't a single vulnerability, but a collection of vulnerabilities. The vulnerabilities in the collection all have to do with making sure your most important data is encrypted when it needs to be. This includes:
- Making sure you are encrypting the correct data
- Making sure you have proper key storage and management
- Making sure that you are not using known bad algorithms
- Making sure you are not implementing your own cryptography, which may or may not be secure
- Developers should identify all sensitive data and encrypt that data, even when it's stored on a hard drive. Ensure that sensitive data cannot be easily overwritten and overwrite sensitive memory locations immediately. Additionally, identify the people who should and shouldn't have knowledge of secrets such as proprietary algorithms, encryption keys and DRM. In most cases, it's recommended to hide these secrets from even the administrator. Additionally, identify all sensitive data read into the memory and overwrite it with random data.
- https://www.veracode.com/security/insecure-crypto

Insufficient Transport Layer Protection

- When designing a mobile application, data is commonly exchanged in a client-server fashion. When the solution transmits its data, it must traverse the mobile device's carrier network and the internet. Threat agents might exploit vulnerabilities to intercept sensitive data while it's traveling across the wire. The following threat agents exist:
- An adversary that shares your local network (compromised or monitored Wi-Fi);
- Carrier or network devices (routers, cell towers, proxy's, etc); or
- Malware on your mobile device.
- https://owasp.org/www-project-mobile-top-10/2014-risks/m3-insufficient-transport-layer-protection

Insecure Direct Object References

- Insecure direct object references (IDOR) are a cybersecurity issue that occurs when a web application developer uses an identifier for direct access to an internal implementation object but provides no additional access control and/or authorization checks. For example, an IDOR vulnerability would happen if the URL of a transaction could be changed through client-side user input to show unauthorized data of another transaction.
- Most web applications use simple IDs to reference objects. For example, a user in a database will usually be referred to via the user ID. The same user ID is the primary key to the database column containing user information and is generated automatically. The database key generation algorithm is very simple: it usually uses the next available integer. The same database ID generation mechanisms are used for all other types of database records.
- https://www.acunetix.com/blog/web-security-zone/what-are-insecure-direct-object-references/

Broken Authetication and Session Management

- These types of weaknesses can allow an attacker to either capture or bypass the authentication methods that are used by a web application.
- User authentication credentials are not protected when stored.
- Predictable login credentials.
- Session IDs are exposed in the URL (e.g., URL rewriting).
- Session IDs are vulnerable to session fixation attacks.
- Session value does not timeout or does not get invalidated after logout.
- Session IDs are not rotated after successful login.
- Passwords, session IDs, and other credentials are sent over unencrypted connections.
- The goal of an attack is to take over one or more accounts and for the attacker to get the same privileges as the attacked user.
- https://hdivsecurity.com/owasp-broken-authentication-and-session-management

Broken Authetication and Session Management

SESSION MANAGEMENT

- Credentials should be protected: User authentication credentials should be protected when stored using hashing or encryption.
- Do not expose session ID in the URL: Session IDs should not be exposed in the URL (e.g., URL rewriting).
- Session IDs should timeout: User sessions or authentication tokens should be properly invalidated during logout.
- Recreate session IDs: Session IDs should be recreated after successful login.
- Do not send credentials over unencrypted connections: Passwords, session IDs, and other credentials should not be sent over unencrypted connections.

BROKEN AUTHENTICATION

- Password length: Minimum password length should be at least eight (8) characters long. Combining this length with complexity makes a password difficult to guess using a brute force attack.
- Password complexity: Passwords should be a combination of alphanumeric characters. Alphanumeric characters consist of letters, numbers, punctuation marks, mathematical and other conventional symbols.
- Username/Password Enumeration: Authentication failure responses should not indicate which part of the authentication data was incorrect. For example, instead of "Invalid username" or "Invalid password", just use "Invalid username and/or password" for both. Error responses must be truly identical in both display and source code.
- Protection against brute force login: Enforce account disabling after an established number of invalid login attempts (e.g., five attempts is common). The account must be disabled for a period of time sufficient to discourage brute force guessing of credentials, but not so long as to allow for a denial-of-service attack to be performed.

Improper Handling

- The software does not handle or incorrectly handles an exceptional condition.
- Improper Check or Handling of Exceptional Conditions
- Generation of Error Message Containing Sensitive Information
- Improper Handling of Insufficient Permissions or Privileges
- Detection of Error Condition Without Action
- Use of NullPointerException Catch to Detect NULL Pointer Dereference
- Declaration of Catch for Generic Exception
- Improper Cleanup on Thrown Exception
- Missing Standardized Error Handling Mechanism
- Not Failing Securely ('Failing Open')
- Missing Custom Error Page
- Improper Handling of Single Event Upsets
- https://cwe.mitre.org/data/definitions/755.html

Unvalidated Redirects and Forwards

- Unvalidated redirects and forwards are possible when a web application accepts untrusted input that could cause the web application to redirect the request to a URL contained within untrusted input. By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts may have a more trustworthy appearance. Unvalidated redirect and forward attacks can also be used to maliciously craft a URL that would pass the application's access control check and then forward the attacker to privileged functions that they would normally not be able to access.
- https://hdivsecurity.com/owasp-unvalidated-redirects-and-forwards

Failure to Restrict URL Access

- 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.

Failure to Restrict URL Access 2

- To prevent forced browsing, you can use appropriate permissions or ACLs to disallow anonymous reading. Also, do not allow anonymous web visitors user read permissions to any sensitive data files.
- Secondly, define the list of file types available for remote reading on the server. Many servers allow you to define which file extensions can be served remotely. For example, .log, .dat and database files are not files that all users should have access to - except through secure channels.
- Also, remove all unnecessary files from web-accessible directories. That is, if files are unneeded within the directory, remove them, even though they may be secure. ACLs may need to change in the future, and this could pose a security risk.
- Finally, use virtual directories for web access, separate secure directories data. They allow you to specify granular ACLs and can help in directory traversal bugs.

Why we need different Approach for Security Requeriments Gathering

- Functional requeriments are positive requeriments specifying what the software should do
- Security requeriments are negative requeriments specifying what the software should not do
- It is against natural tendency of people that they are clear about what they want but quite find it difficult to understand what they don’t what
- A software need to be viewd in a more negative, critical and destructive way to reveal its non-intended use and its associated security requeriments

Stakeholders involvement in Security Requeriments Gathering

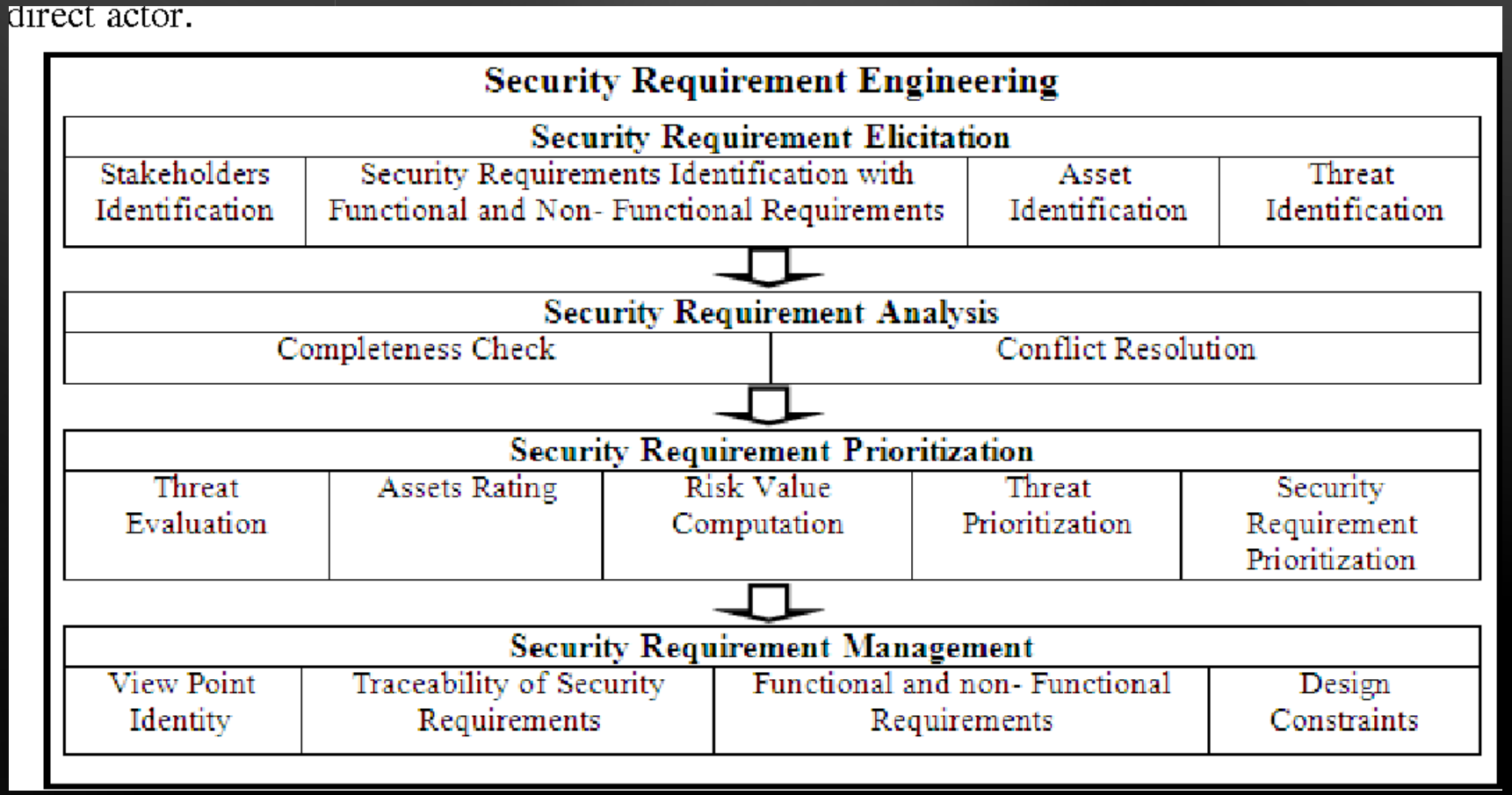
- User, Costumer or Business Sponsor
- Developers
- Requeriment Engineer
- Security Expert
- Project Manager/Experts

Security Requeriments Engineering

- Security Requeriment Engineering (SRE) empowers security requeriment gathering in the Application development process
- SRE is the systemat approach to identify, analyze, elict and specify security requeriments for software system
- SRE plays a major role in integrated security software development lifecycle process

Security Requeriments Engineering 2

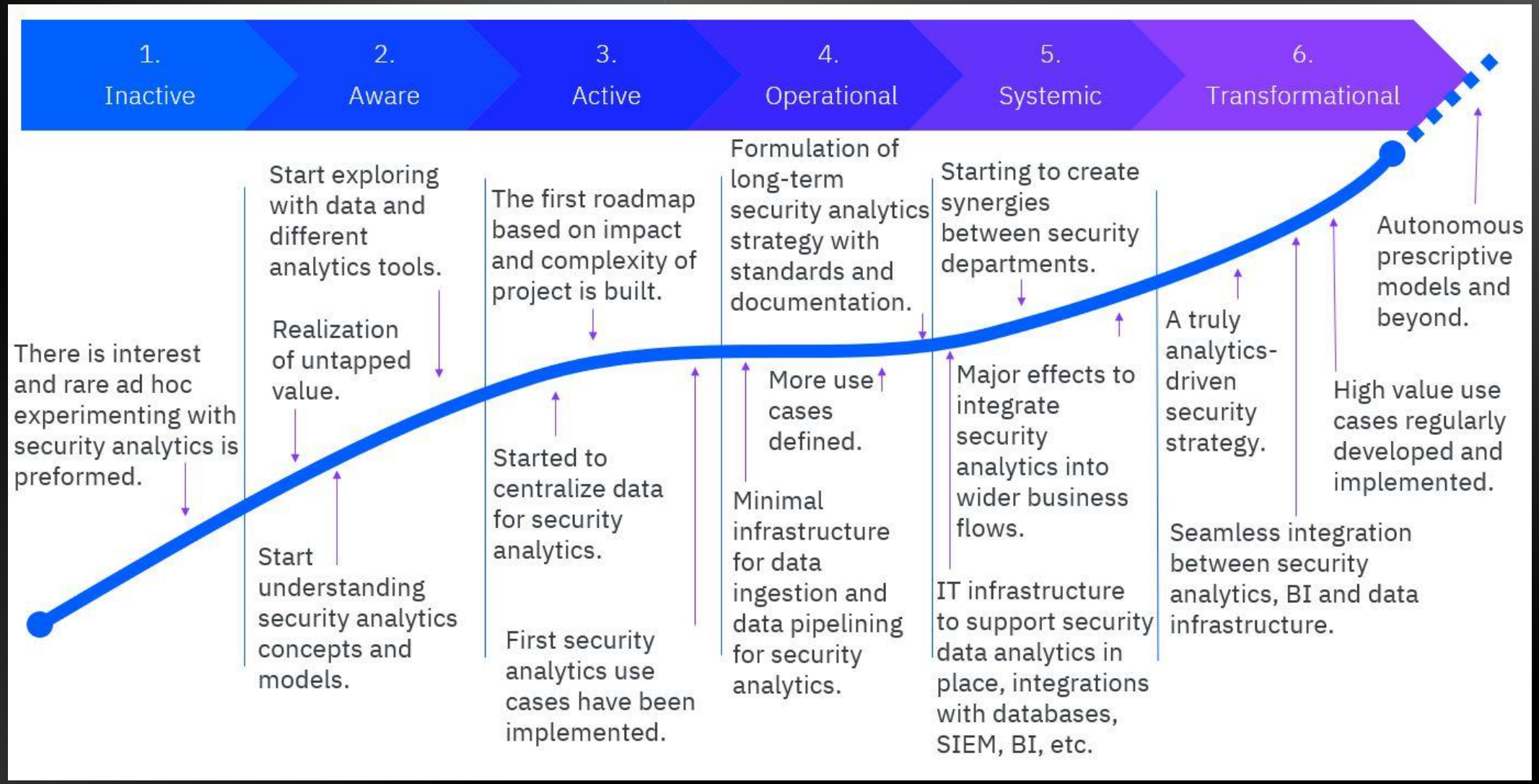
https://www.semanticscholar.org/paper/Security-Requirements-Engineering-%3A-Analysis-and-Gupta-Jaiswal/6eaea3a10127db1dde65c86e2b65decc0a055b57/figure/1



Threat Modeling

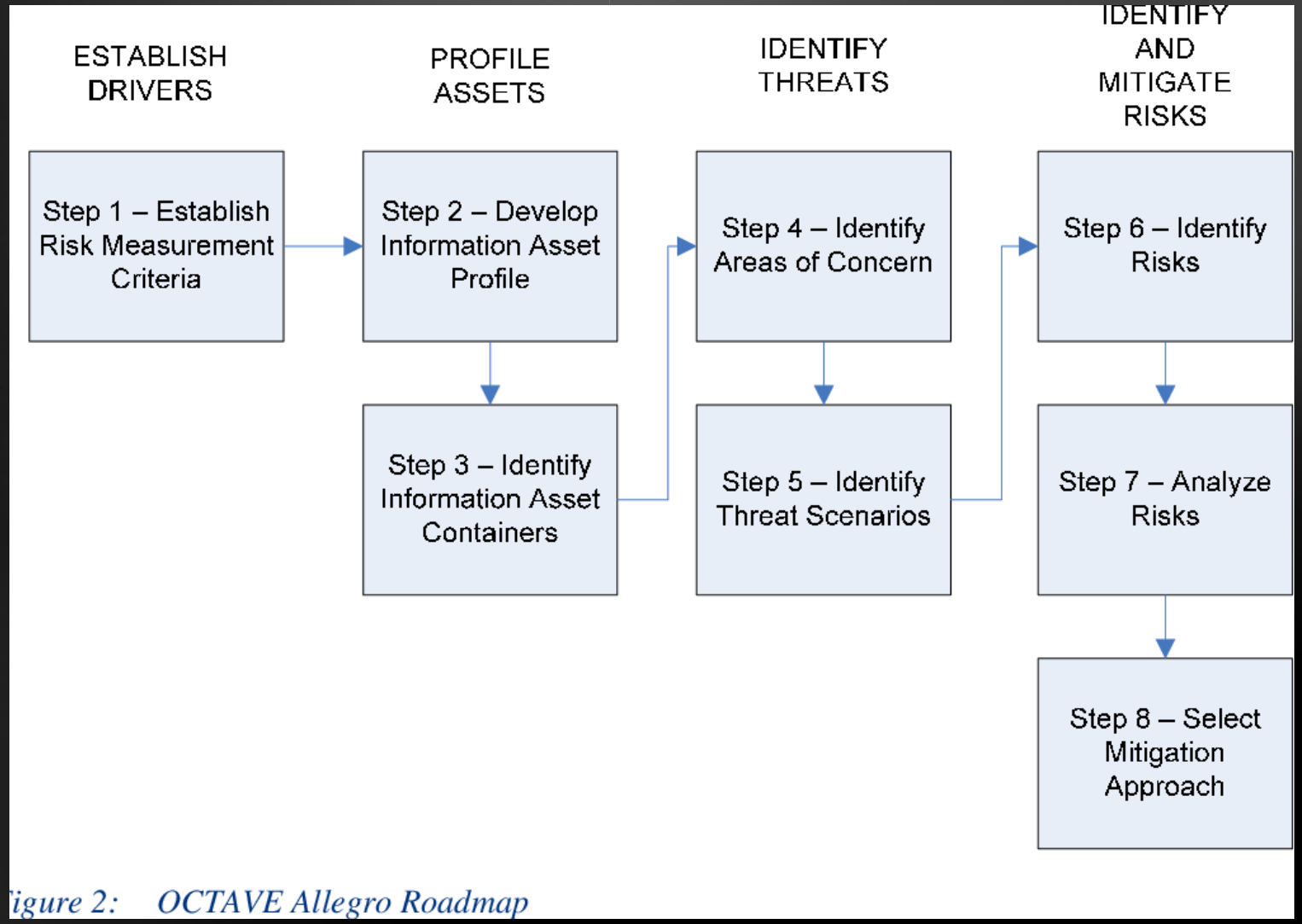
- https://towardsdatascience.com/how-to-perform-threat-modeling-security-analysis-in-5-steps-c42639efc952
- Step 1: Identify the Use Case, Assets to Protect, and External Entities
- Step 2: Identify Trust Zones, Potential Adversaries, and Threats
- Step 3: Determine High-Level Security Objectives to Address Potential Threats
- Step 4: Define Security Requirements for Each Security Objective Clearly
- Step 5: Create a Document to Store All Relevant Information

Threat Modeling 2



https://securityintelligence.com/posts/how-security-maturity-model-transforms-analytics/

OCTAVE



igure 2: OCTAVE Allegro Roadmap

https://www.semanticscholar.org/paper/Introducing-OCTAVE-Allegro%3A-Improving-the-Security-Caralli-Stevens/490fb4c8fad85a2ea6975504b4984d713cfb0361

Input Validators

- Input validation is performed to ensure only properly formed data is entering the workflow in an information system, preventing malformed data from persisting in the database and triggering malfunction of various downstream components. Input validation should happen as early as possible in the data flow, preferably as soon as the data is received from the external party.
- Data from all potentially untrusted sources should be subject to input validation, including not only Internet-facing web clients but also backend feeds over extranets, from suppliers, partners, vendors or regulators, each of which may be compromised on their own and start sending malformed data.
- Input Validation should not be used as the *primary* method of preventing XSS, SQL Injection and other attacks which are covered in respective cheat sheets but can significantly contribute to reducing their impact if implemented properly.

Input Validators Strategies

- Input validation should be applied on both syntactical and Semantic level.
- Syntactic validation should enforce correct syntax of structured fields (e.g. SSN, date, currency symbol).
- Semantic validation should enforce correctness of their values in the specific business context (e.g. start date is before end date, price is within expected range).
- It is always recommended to prevent attacks as early as possible in the processing of the user's (attacker's) request. Input validation can be used to detect unauthorized input before it is processed by the application.
- https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

Injection Prevetion

- Application accessibility is a very important factor in protection and prevention of injection flaws. Only the minority of all applications within a company/enterprise are developed in house, where as most applications are from external sources. Open source applications give at least the opportunity to fix problems, but closed source applications need a different approach to injection flaws.
- Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code, often found in SQL queries, LDAP queries, XPath queries, OS commands, program arguments, etc. Injection flaws are easy to discover when examining code, but more difficult via testing. Scanners and fuzzers can help attackers find them.
- Depending on the accessibility different actions must be taken in order to fix them. It is always the best way to fix the problem in source code itself, or even redesign some parts of the applications. But if the source code is not available or it is simply uneconomical to fix legacy software only virtual patching makes sense.

Injection Prevetion

- Three classes of applications can usually be seen within a company. Those 3 types are needed to identify the actions which need to take place in order to prevent/fix injection flaws.

A1: New Application

- A new web application in the design phase, or in early stage development.

A2: Productive Open Source Application

- An already productive application, which can be easily adapted. A Model-View-Controller (MVC) type application is just one example of having a easily accessible application architecture.

A3: Productive Closed Source Application

- A productive application which cannot or only with difficulty be modified.

Injection Prevetion

- Three classes of applications can usually be seen within a company. Those 3 types are needed to identify the actions which need to take place in order to prevent/fix injection flaws.

A1: New Application

- A new web application in the design phase, or in early stage development.

A2: Productive Open Source Application

- An already productive application, which can be easily adapted. A Model-View-Controller (MVC) type application is just one example of having a easily accessible application architecture.

A3: Productive Closed Source Application

- A productive application which cannot or only with difficulty be modified.

Use Privacy Protection

Strong Cryptography

- Any online platform that handles user identities, private information or communications must be secured with the use of strong cryptography. User communications must be encrypted in transit and storage. User secrets such as passwords must also be protected using strong, collision-resistant hashing algorithms with increasing work factors, in order to greatly mitigate the risks of exposed credentials as well as proper integrity control.
- To protect data in transit, developers must use and adhere to TLS/SSL best practices such as verified certificates, adequately protected private keys, usage of strong ciphers only, informative and clear warnings to users, as well as sufficient key lengths. Private data must be encrypted in storage using keys with sufficient lengths and under strict access conditions, both technical and procedural. User credentials must be hashed regardless of whether or not they are encrypted in storage.

Use Privacy Protection 2

Support HTTP Strict Transport Security

- HTTP Strict Transport Security (HSTS) is an HTTP header set by the server indicating to the user agent that only secure (HTTPS) connections are accepted, prompting the user agent to change all insecure HTTP links to HTTPS, and forcing the compliant user agent to fail-safe by refusing any TLS/SSL connection that is not trusted by the user.
- HSTS has average support on popular user agents, such as Mozilla Firefox and Google Chrome. Nevertheless, it remains very useful for users who are in consistent fear of spying and Man in the Middle Attacks.

Use Privacy Protection 3

Digital Certificate Pinning

- Certificate Pinning is the practice of hardcoding or storing a predefined set of information (usually hashes) for digital certificates/public keys in the user agent (be it web browser, mobile app or browser plugin) such that only the predefined certificates/public keys are used for secure communication, and all others will fail, even if the user trusted (implicitly or explicitly) the other certificates/public keys.

Some advantages for pinning are:

- In the event of a CA compromise, in which a compromised CA trusted by a user can issue certificates for any domain, allowing evil perpetrators to eavesdrop on users.
- In environments where users are forced to accept a potentially-malicious root CA, such as corporate environments or national PKI schemes.
- In applications where the target demographic may not understand certificate warnings, and is likely to just allow any invalid certificate.
- https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html

Digital Certificate

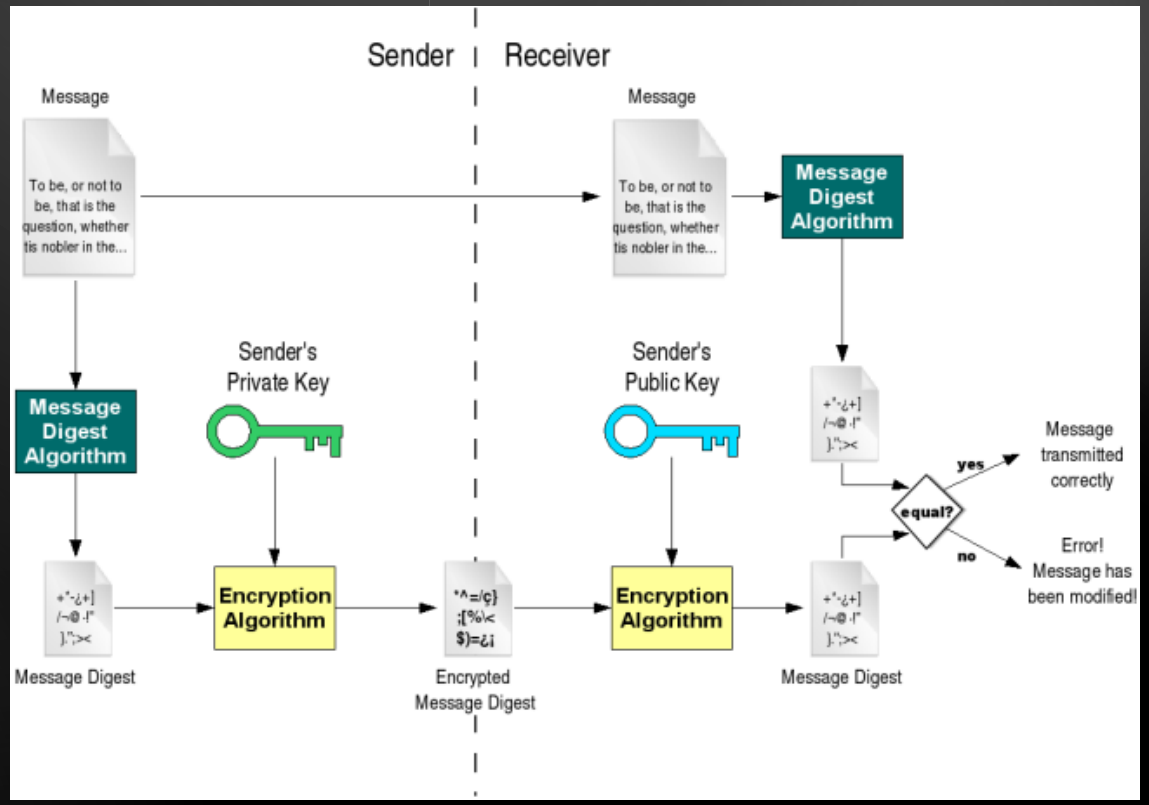
- A digital certificate, also known as a public key certificate, is used to cryptographically link ownership of a public key with the entity that owns it. Digital certificates are for sharing public keys to be used for encryption and authentication.
- Digital certificates include the public key being certified, identifying information about the entity that owns the public key, metadata relating to the digital certificate and a digital signature of the public key the certificate issuer created.
- The distribution, authentication and revocation of digital certificates are the primary functions of the public key infrastructure (PKI), the system that distributes and authenticates public keys.

Digital Certificate

Digital certificates are used in the following ways:

- Credit and debit cards use chip-embedded digital certificates that connect with merchants and banks to ensure that the transactions performed are secure and authentic.
- Digital payment companies use digital certificates to authenticate their automated teller machines, kiosks and point-of-sale equipment in the field with a central server in their data center.
- Websites use digital certificates for domain validation to show they are trusted and authentic.
- Digital certificates are used in secure email to identify one user to another and may also be used for electronic document signing. The sender digitally signs the email, and the recipient verifies the signature.
- Computer hardware manufacturers embed digital certificates into cable modems to help prevent the theft of broadband service through device cloning.
- As cyberthreats increase, more companies are considering attaching digital certificates to all of the IoT devices that operate at the edge and within their enterprises. The goals are to prevent cyberthreats and protect intellectual property.
- https://www.techtarget.com/searchsecurity/definition/digital-certificate

Digital Certificate 2



https://docs.huihoo.com/globus/gt4-tutorial/ch09s03.html

SSL and TLS

- This cheat sheet provides guidance on how to implement transport layer protection for an application using Transport Layer Security (TLS). When correctly implemented, TLS can provides a number of security benefits:
- Confidentiality - protection against an attacker from reading the contents of traffic.
- Integrity - protection against an attacker modifying traffic.
- Replay prevention - protection against an attacker replaying requests against the server.
- Authentication - allowing the client to verify that they are connected to the real server (note that the identity of the client is not verified unless client certificates are used).
- TLS is used by many other protocols to provide encryption and integrity, and can be used in a number of different ways. This cheatsheet is primarily focused on how to use TLS to protect clients connecting to a web application over HTTPS; although much of the guidance is also applicable to other uses of TLS.

SSL vs TLS

- Secure Socket Layer (SSL) was the original protocol that was used to provide encryption for HTTP traffic, in the form of HTTPS. There were two publicly released versions of SSL - versions 2 and 3. Both of these have serious cryptographic weaknesses and should no longer be used.
- For various reasons the next version of the protocol (effectively SSL 3.1) was named Transport Layer Security (TLS) version 1.0. Subsequently TLS versions 1.1, 1.2 and 1.3 have been released.
- The terms "SSL", "SSL/TLS" and "TLS" are frequently used interchangeably, and in many cases "SSL" is used when referring to the more modern TLS protocol. This cheatsheet will use the term "TLS" except where referring to the legacy protocols.

Test the Server Configuration

- There are a number of online tools that can be used to quickly validate the configuration of a server, including:
- SSL Labs Server Test
- CryptCheck
- CypherCraft
- Hardenize
- ImmuniWeb
- Observatory by Mozilla
- Scanigma
- Additionally, there are a number of offline tools that can be used:
- O-Saft - OWASP SSL advanced forensic tool
- CipherScan
- CryptoLyzer
- SSLScan - Fast SSL Scanner
- SSLyze
- testssl.sh - Testing any TLS/SSL encryption
- tls-scan

Certificates

Use Strong Keys and Protect Them

- The private key used to generate the cipher key must be sufficiently strong for the anticipated lifetime of the private key and corresponding certificate. The current best practice is to select a key size of at least 2048 bits. Additional information on key lifetimes and comparable key strengths can be found here and in NIST SP 800-57.
- The private key should also be protected from unauthorized access using filesystem permissions and other technical and administrative controls.

Use Strong Cryptographic Hashing Algorithms

- Certificates should use SHA-256 for the hashing algorithm, rather than the older MD5 and SHA-1 algorithms. These have a number of cryptographic weaknesses, and are not trusted by modern browsers.

Use Correct Domain Names

- The domain name (or subject) of the certificate must match the fully qualified name of the server that presents the certificate. Historically this was stored in the commonName (CN) attribute of the certificate. However, modern versions of Chrome ignore the CN attribute, and require that the FQDN is in the subjectAlternativeName (SAN) attribute. For compatibility reasons, certificates should have the primary FQDN in the CN, and the full list of FQDNs in the SAN.
- Additionally, when creating the certificate, the following should be taken into account:
- Consider whether the "www" subdomain should also be included.
- Do not include non-qualified hostnames.
- Do not include IP addresses.
- Do not include internal domain names on externally facing certificates.
    - If a server is accessible using both internal and external FQDNs, configure it with multiple certificates.

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

Code Review

- Code Review, also known as Peer Code Review, is the act of consciously and systematically convening with one's fellow programmers to check each other's code for mistakes and has been repeatedly shown to accelerate and streamline the process of software development like few other practices can. There are peer code review tools and software, but the concept itself is important to understand. Software is written by human beings. Software is therefore often riddled with mistakes. To err is, of course, human, so this is an obvious correlation. But what isn't so obvious is why software developers often rely on manual or automated testing to vet their code to the neglect of that other great gift of human nature: the ability to see and correct mistakes ourselves.
- https://smartbear.com/learn/code-review/what-is-code-review

Code Review 2 - Tools

https://pt.slideshare.net/alimenkou/code-review-tool-for-personal-effectiveness-and-waste-analysis

Code Review

- Code Review, also known as Peer Code Review, is the act of consciously and systematically convening with one's fellow programmers to check each other's code for mistakes and has been repeatedly shown to accelerate and streamline the process of software development like few other practices can. There are peer code review tools and software, but the concept itself is important to understand. Software is written by human beings. Software is therefore often riddled with mistakes. To err is, of course, human, so this is an obvious correlation. But what isn't so obvious is why software developers often rely on manual or automated testing to vet their code to the neglect of that other great gift of human nature: the ability to see and correct mistakes ourselves.
- https://smartbear.com/learn/code-review/what-is-code-review

Application Security Labs

- https://appsec-labs.com/
- https://www.immersivelabs.com/labs-and-content/app-security/
- https://application.security/
- https://www.veracode.com/products/security-labs

Programming Language Security