TOPIC 6
INTRODUCTION TO APPLICATION SECURITY

Creating secure code

JOSÉ MANUEL REDONDO LÓPEZ *"S-81 ISAAC PERAL"* PROJECT V4.0

Source: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Logo: @creative_vanesa (*Instagram*)

INDEX

< Go to Index

APPLICATION SECURITY FUNDAMENTALS

Basics before you start

*Icons by Bing Chat AI

Source: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Secure software design principles (for ALL software)

- **Least privilege**
  - Software or users must have only the necessary privileges their your work
- **Fail-safe defaults**
  - Unless the software receives explicit access to an object, access must always be denied by default
- **Full Mediation**
  - Each object access must be validated to ensure it is allowed
- **Separation**
  - Software must not grant access to a resource, or take security-relevant actions, based on a single condition
- **Minimum trust**
  - Software must check all inputs and the results of all security-relevant actions
- **Economy of mechanisms**
  - Security functions should be as simple as possible

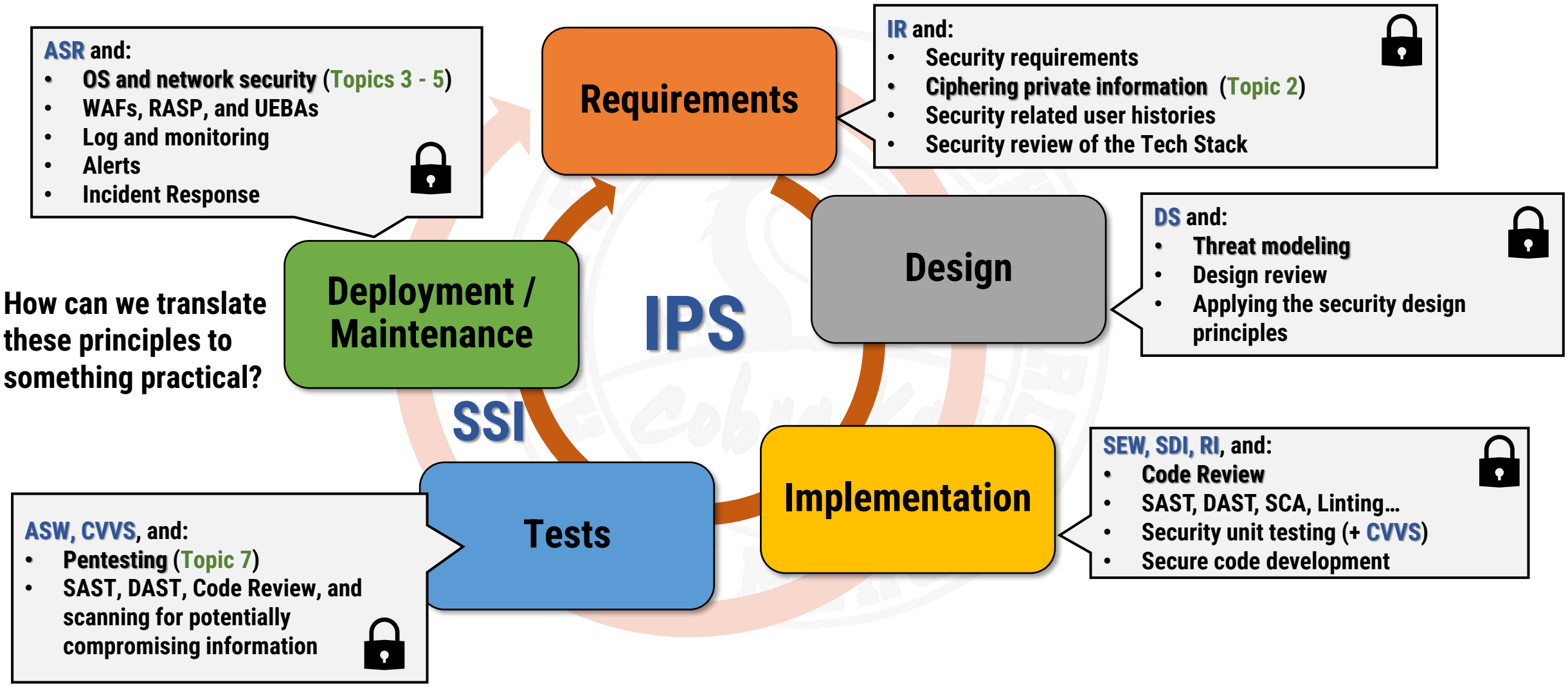
Secure software design principles (for ALL software)

- **Minimize common mechanisms**
  - Minimize the use of shared resources
- **Less amazement/surprise**
  - Software functions and security mechanisms should be designed so handling them is logical and simple
- **Open design**
  - Software security and its functions should not depend on the secrecy of its design or implementation
- **Layers (separation principle)**
  - Organize the software into layers: modules can only interact with those of layers immediately below and above
  - Allows layered testing, using top-down or bottom-up techniques, and reduces access points

Secure software design principles (for ALL software)

- **Abstraction**
  - Hide the implementation of each layer, leaving only its public interfaces
  - Allows to change the implementation of a layer without affecting the components of other layers
- **Modularity**
  - Design and implement software as a collection of cooperative components (**modules**)
  - Each module interface is an abstraction
- **Full linking**
  - Link the design and implementation of a software's security to its security requirements
- **Iterative Design**
  - Plan the design so that it can change if needed
  - Minimizes the effects of design changes on security

From SDLC to S-SDLC (3rd year, integrated and "secured")



The "Pushing Left" philosophy

- The later a security issue is discovered in the S-SDLC, the more expensive it will be to fix it (in time and resources)
- We must ensure that all the security features of the application are resolved the earliest possible on the SDLC chain
  - This is often referred to as "**pushing left**" philosophy
- This topic also concentrates its explanations on the initial elements of the chain
- Rest of the elements are also covered by other courses (SEW, SDI, CVVS, IPS, ASR), or by other course parts

< Go to Index

Source: Bing Chat AI

Data handling >

Authentication >

Other >

REQUIREMENT-LEVEL SECURITY

Security is planned from minute zero

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

*Icons by Bing Chat AI

What are you going to find in this block?

- **In the first section of this block, you'll learn**
  - How important it is to **treat any input that the application receives as insecure**, and why it is essential to validate it
    - And the different points where **you can have data inputs**
  - The importance (and difference) between **encoding and character "escaping“**
  - What you need to do to create **secure cookies** in your web app
- **In the second, you'll learn**
  - The keys to **strong authentication**: MFA and password management requirements
  - The difference between **service and user accounts** and why the former is better for applications
  - How to plan the **recover a forgotten password** requirement securely
- **And in the third you'll learn**
  - How hugely important it is to **minimize the use of third-party components**
  - **Other important security requirements**: HTTP headers, backups, file uploads, logs, authentication and authorization forms, least privilege, etc.

SECURITY QUESTIONS IN THE REQUIREMENTS

- **In 4th year there is a course called Requirements Engineering (IR)**
  - You should use its contents to model your application requirements **and add the security ones**
  - Your methodology (IPS, ASW), language (TPP) or framework (SDI, SEW) does not matter
- **Have a person or team assigned to develop application security requirements**
- **They need to ask "unpleasant" questions to elaborate them**
  - *Does it contain or handle confidential, sensitive, or Personal Identifiable Information (PII)?*
  - *Where and how is the data stored?*
  - *Is the application publicly available?* (Internet)
    - Or not? (Intranet-only)
  - *Is the application intended to perform sensitive or essential tasks?*
    - Money transfers, handling medical data...
  - *Does it need to allow potentially dangerous operations?*
    - Ex.: Allowing file uploads, handling payment data, having multiple user roles…
  - ... (do not be shy!, *"Take what you can, and give nothin' back"*)

ENCRYPTION REQUIREMENTS

- **We saw encryption types in Topic 2, and we must remember...**
  - Passwords and sensitive information are **NEVER saved in plain text**
  - **Hashing techniques** are used for passwords (Key Derivation Functions)
    - And any information whose original value is not needed
    - Identity, authentication, data integrity...
  - For all other cases, we will use **strong encryption/decryption** algorithms
    - Always encrypt sensitive information **in transit** (user <-> frontend, frontend<->backend, backend <-> DBMS, API <-> API, etc.)
    - And **when it is saved** (in the DBMS) (also called "**encryption at rest**")
  - This would **ensure the confidentiality and integrity** of the information (not availability)
  - Encrypting data in RAM is possible, but only with extremely sensitive data
    - It is better to "clean" the memory once we use them

Data handling requirements

Different techniques to treat your data in a secure way

*Icons by Bing Chat AI

Data handling requirements

- **Never trust user input**
  - **Any input** that reaches the system can be spoofed or maliciously manipulated
  - This can cause an application to fail in unforeseen ways and become unstable
    - This situation is when an attacker can use our app to trigger a very dangerous attack
  - Therefore, **ALL INPUTS must be validated** and handled properly
    - Do not allow your application to fall into unstable states
  - Always **rely on allowlists** of supported entries if possible
    - In addition, the rules for validating each type of entry are known and classified
    - https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
  - **Validation ONLY in JavaScript is useless from a security point of view**: it can be disabled
    - Web proxies allows you to change any content before it is transmitted to the server, even if validated
    - E.g. Burp, https://portswigger.net/burp/documentation/desktop/tools/proxy, see **Seminar 3**
- **But if you know how and where to do it, what's the problem?**
  - That many times we do not know the **HUGE AMOUNT** of input data sources that we handle
  - Let's look at a non-exhaustive list of manipulable things

POSSIBLE DATA ENTRY POINTS

- **Any input from the users (the obvious one) even HTTP headers, "hidden" fields....**
- **Information received from**
  - **DBMS** (whether it is yours or not)
  - **API** (even if you have made it yourself)
  - Code included in the application and compiled with it (**third-party dependencies**)
    - Libraries, frameworks, GitHub code...
  - **Another application**
    - Especially if ours will be integrated with it, or accepts inputs from it
    - It includes serverless applications, scripts, external frameworks, data to log (tell it to Log4j!)...
- **Values in the URL (parameters), in cookies, in configuration files...**
  - Especially if the cookies do not have the flags "secure" and "HTTPS only"
  - Includes values that are used from online storage systems (S3, Firebase...)
- **Data or commands that come from cloud applications**
- **Images you receive from other places**

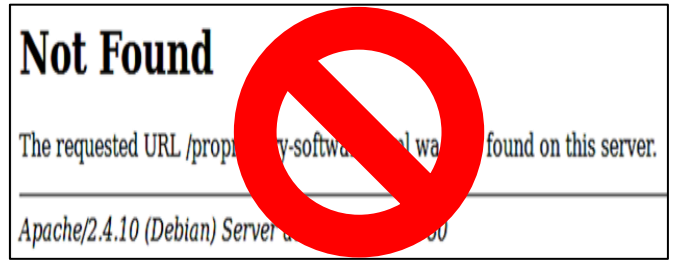
DATA HANDLING REQUIREMENTS

- An input is **NOT reliable until it has passed your validation procedures**
  - **NEVER before that**
  - Validation comprises all tests ensuring that **the input is adequate and has the correct type**
  - Any discrepancies result in the **application rejecting the received information**
- Examples
  - If you're expecting a **date of birth**…
    - Check that it's a date, and that it's realistic in the context of the application
  - If you expect a **name**…
    - Check that it has the characteristics of an expected name in your case (O'Reilly, Japanese characters...)
  - If you expect an **email**…
    - There are regular expressions that can validate them without error
  - If you receive the **result of a search**…
    - Validate that it has the expected type and format (XML, JSON, its values ...)
  - If you call any API and it returns **a known type**…
    - You MUST check that what you receive has this type, and it is **not excessively long**
    - In general, **it is advisable to set a maximum expected size for any handled data**, if possible

DATA HANDLING REQUIREMENTS

- If your app must accept **URLs** and handle them…
  - **You should consider that you are in a potentially dangerous situation (open redirect)**
  - Try to **avoid** that scenario as much as possible
  - Validate that it is a **real** URL (regular expressions)
  - Make sure it arrives through an **encrypted channel**
- If your application (or part of it) must be made in a **non memory-safe** language (ex. C/C++)…
  - This is usually done when performance is the most important factor, but incorporates **new threats**…
  - Use **Rust**: similar performance and memory safety; eliminates a great threat: **buffer overflows**
- ... (think and adapt it to your case)

- **If after all this you detect an input error, always reject the data and report the error**
  - **DO NOT try to correct the data**
  - Never give too much information to the user
    - It could be used against you
  - Try **not to reproduce the data** that caused the error in the message
    - If you must, then resort to **encoding and "escaping"** it



Data handling requirements

- **Data privacy**
  - There must be a policy specifying cookie collected data
  - Adapted to current legislation (see ASLEPI)
- **Data classification**
  - All data used by the application must be **classified and labeled** depending on its privacy
    - Anyone will know what kind of data they are working with
    - And, therefore, what things must be especially protected so that the privacy requirements are met
  - The type of classification depends on the **applicable country/legislation**
    - E.g.: Classified, Secret, Top Secret (see ASLEPI)
    - It will depend on whether knowing a piece of data can harm a person, a company or even an entire country
  - If there are no applicable rules, you can follow the **NIST** ones
    - "Guide for Mapping Types of Information and Information Systems to Security Categories"
      - https://www.nist.gov/publications/guide-mapping-types-information-and-information-systems-security-categories-2-vols

Requirements: Application of encoding and "escaping"

- "Escaping" a character or value means removing "the power" it has in a context
  - Typically, the ability **to be interpreted as code**
  - It is usually done by adding a `\` to the character
- Encoding a value means changing it to an equivalent in a different format
  - Typically, the target format is **harmless** in the context is going to be used
    - And never interpreted as code!
    - E.g.: in HTML "`>`" becomes "`>`"
  - The **encoding type** depends on the context in which it is going to be used
    - URL, base64, HTML encoding...
  - Encoding **IS** reversible and **IT IS NOT** encryption
- Both are done to avoid code injection attacks
  - E. g.: the well-known XSS (which we will talk about later)
- Requirements determine the situations these techniques should be applied

Cookie security requirements

- Cookies are the mechanism that was integrated into the HTTP protocol to be able to have state (SEW)
  - **A Cookie is not the same as the "Local Storage" of the browser**
  - They have **different** security rules
    - https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html
- If cookies are used, the requirements are
  - Everything written in a cookie **must be validated** before it is used, and rejected if incorrect
  - The session ID must ALWAYS be passed **in a session cookie**
    - Never in a persistent cookie (tracking cookie)
    - A session cookie is destroyed at the end of a session
  - Cookies must be **sent under encrypted connections** (Secure Flag, `Set-Cookie: Secure`)
  - Cookies must not be accessible from JavaScript (HTTPOnly Flag, `Set-Cookie: HttpOnly`)
    - This prevents some XSS attack vectors
  - Persistent cookies should have an **expiration time** (`Expires` flag)

Cookie security requirements

- **Domain**
  - By default, **the cookies of a domain can only be accessed by websites of that domain**
  - It is **secure by default**
  - If we want other domains to read our cookies, we can do so with `Set-Cookie: Domain: ads.redondo.com`)
    - If we put `redondo.com` instead, any page on that domain could read our cookie
  - It is recommended to leave the default option, or restrict domains to the maximum
- **Path**
  - **Many applications are composed by different sub-applications that form a much larger one**
  - All under a single base URL
  - Each sub-application owns a particular path of the base URL
  - In this scenario each cookie should be read only by its "piece" of application
  - Each “piece” is specified with its path within the URL (`Set-Cookie: path=/accounting`)

Cookie security requirements

- **Same-Site**
  - **Measure against CSRF attacks** (seen later)
  - The `Strict` value (recommended) means that cookies can only come from the same site that uses them, not from a link or another page
    - `Set-Cookie: same-site=Strict`
  - The `Lax` value blocks cookies from other pages that come via `POST`, but allows links
- **Cookie Prefixes**
  - New defense-in-depth cookie security measure
  - Ensures that only certain subdomains can access a cookie
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie

Think you've understood it all? Evaluate yourself!

?

- **You can do the following**
  - *Understand that thinking about security requirements requires asking the right questions*
  - *Distinguish between encryption in transit and encryption at rest*
  - ***About data handling requirements***
    - *Be very aware that no user input is trustworthy, under any circumstances*
    - *Understand that user input is not only **textfield** components, but many other things related to the HTTP protocol, third-party software with which we interact...*
    - *Understand that the type of data expected is key when validating it*
    - *Be aware that data must be labeled according to your privacy and applicable law*
    - *Distinguish between encoding and escaping a piece of data, when and what it is used for*
    - *Understand the function of all cookie security tools: **Secure**, **HttpOnly**, **Expires**, **Domain**, **Path**, and **Same-Site***

Authentication requirements

Dealing with passwords and user identities

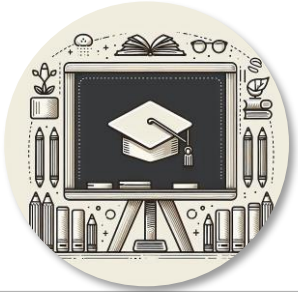
*Icons by Bing Chat AI

User password & Application secrets requirements

- **Users should use a password manager (e.g.: KeePass, https://keepass.info/)**
  - This avoids remembering many passwords and ensures all are complex
  - They usually have **extra features**
    - Automatically look if a registered account is part of a data breach
    - Report if a site has MFA available (seen later)
  - This way **the password management is optimized and secure passwords are enforced**
    - In case of data theft, the impact is minimized (the reuse of keys and credential stuffing is avoided)

  Password manager + unique password by web/service + MFA = personal defense in depth

- **Secret stores are like password managers but for software, not people**
  - Software file ("vault") that encrypts secrets of an application
    - Keys, hashes, certificates, cloud service tokens, anything that requires protection...
  - **Programmatic access**: from the application code or the build process of the CI/CD pipeline (ASW)
    - The development framework to be used must implement these stores
    - At the requirements level, it must be determined **WHAT** should go in these stores
  - **This MUST be done instead of "hardcoding" application secrets**
    - This problem also occurs with applications made **with low-level languages**
    - We will review it in **Seminar 4** (reversing)



Requirements for user passwords

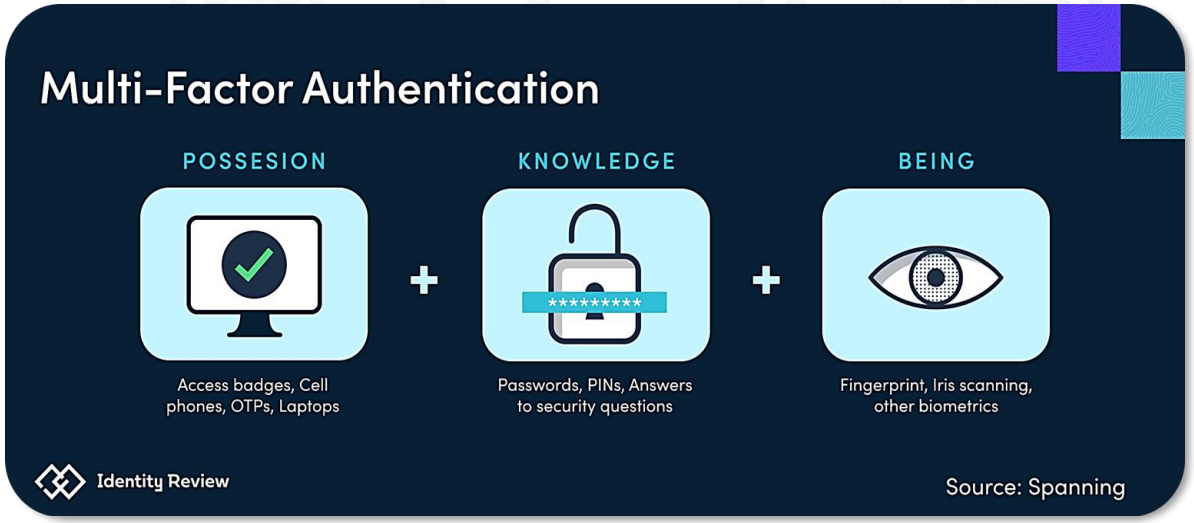
- Try to **NOT** store passwords in your application
  - Use a third-party authentication service (Google, Outlook…) if possible
- If this is not possible, store passwords following Topic 2
  - **KDF algorithm**, long, complex, **salt** (28+ characters), **pepper**, **strong hashing**...
  - The `haveibeenpwned` API can be used to verify if a new password is already compromised
    - https://haveibeenpwned.com/API/v2
  - **Never inform the user** about what it is wrong exactly (username or password)
    - When trying to login unsuccessfully, use a generic message: “Account information is wrong”
  - Never allow the **reuse of keys** or to use very similar derived keys ("Pass1", "Pass2", etc.)
  - **Use MFA for all important accounts**
    - This applies to both our personal and application accounts
    - Preferably use an application or device rather than an SMS (prevents SIM-Swapping)
  - **Do not force periodic password changes**, only when a breach is discovered

Second / Multi-Factor Authentication (2FA / MFA)

- **The process of requiring users to verify their identity in two or more unique and independent ways before accessing a system**
- **Extends the classic username/password with an extra authentication step**
  - Typically requiring the user to enter a dynamically generated one-time token or one-time password (OTP) displayed by a method that can only be accessed by the user
  - This token is harder to obtain than a password for an attacker
- **MFAs are becoming more common, given the weakness of passwords**
  - The complexity of passwords, if mandatory, is counteracted by using the same password in different services (with different levels of security), phishing, social engineering...
- **With MFA, even with a compromised password account access is blocked**
  - It is necessary to know the MFA method and the OTP or token
  - A repeated failure in an MFA input will generate a compromised password alert
- **Supported by the FIDO Alliance: https://fidoalliance.org/**

Types of MFA. Basic principles

- **MFA schemes are based on users giving 2 out of 3 "somethings"**
  - Something **they Know**, such as an account password or PIN
  - Something **they Have**, such as a physical (mobile) device or an app that generates OTPs
  - Something **they Are**, a unique biological feature such as a fingerprint, voice, or retina
- **There are many ways to implement MFAs**
  - All have their disadvantages, but all increase the security of accounts
  - All methods force the user to enter a 2nd password after the 1st to access the system
  - This second password is dynamically generated and constantly changing



Types of MFA

- **SMS Token**: Sends a unique token via SMS to a registered and validated phone
  - **Drawbacks**: signal loss, SIM cloning or theft (SIM Swapping, user spoofing)…
    - Or SMS interception! (SMS are not encrypted)
  - PlayStation Network and Amazon can use this method (recommend to use another)
- **Email Token**: Identical to SMS, but the code is received via email
  - **Drawbacks** : It is even easier to impersonate the user if the email credentials are stolen
  - A popular implementation is Steam Guard (recommend to use another)
- **Hardware Token**: Users have a physical device (such as a USB stick) that contains a dynamically generated access token that must be connected to the computer
  - Google Titan is a similar approach, but using a PKI infrastructure
    - https://cloud.google.com/titan-security-key/?hl=es
  - Its popularity is rising due to cheaper devices complying the FIDO protocols
    - https://www.amazon.es/YUBIKEY4-YubiKey-4/dp/B018Y1Q71M
  - Typical method to prevent private data exfiltration from stolen PCs

Types of MFA

- **Software Token**: Users install an application that generates the dynamic tokens
  - **The currently recommended cost/effective method**
  - **Drawbacks**: Applications can be expensive to deploy or maintain, but also…
    - Requires installing third-party software (which may be restricted in certain environments)
    - If compromised, then our authentication system will be too.
  - **Examples**: Google/Microsoft Authenticator, Blizzard Authenticator or Steam Guard
- **Phone call**: After valid username/password, the token is communicated in a call
  - **Drawbacks**: Intercepted/redirected call, tapped voice messages, phone signal is required
  - Used to validate users connecting for the 1st time, re-connect after a crash, or use VPNs
- **Biometric Verification**: the user is the token
  - Saving biometric data (e.g.: Fingerprints) raises **concerns about privacy**
    - Users can be impersonated with stolen data
  - Requires special devices to verify them (cameras, scanners…) that impose an **additional cost**
  - Hollywood movies use this to impress viewers, but it's also used in real life!

The Consumer Authentication Strength Maturity Model (CASMM) V6

8 PASSLESS

Passwordless *Examples*: WebAuthN, FIDO2

In addition to managed passwords, your 2FA comes from a physical token or your mobile/desktop's built-in trust center.

VULNERABLE TO: HARDWARE COMPROMISE, FORCE

7 CODELESS

App-based Codeless 2FA *Examples*: Some Microsoft Auth

In addition to managed passwords, you have a 2FA app that prompts you to accept or deny an authentication attempt.

VULNERABLE TO: MALWARE, FORCE

6 APP2FA

App-based 2FA Codes *Examples*: Authenticator, Authy

In addition to managed passwords, you get MFA codes generated for you by an application that only you can access.

VULNERABLE TO: PHISHING, MALWARE

5 SMS2FA

SMS-based 2FA Codes *Examples*: Any SMS-based auth

In addition to managed passwords, you get MFA codes sent to you by text message (SMS).

VULNERABLE TO: PHISHING, SIM-SWAPPING

4 PASSMAN

Password Manager *Examples*: 1Password, LastPass

In addition to having unique passwords, you also store them securely in an encrypted archive.

VULNERABLE TO: ACCOUNT RESET / TAKEOVER

3 QUALPASS

Quality Passwords *Examples*: A012-2vOs-11xM

Your passwords are not just unique, but they're long, random, and they include special characters.

VULNERABLE TO: PASSWORD DUMPS / CRACKING

2 UNIQPASS

Unique Passwords *Examples*: DOBs, Teams, Schools

Your passwords are unique, but they're too short, simple, or contain personal information.

VULNERABLE TO: LIVE PASSWORD GUESSING

1 SHARPASS

Shared Passwords *Examples*: Gmail, Wells Fargo, Netflix

You use the same password in multiple places across the internet.

VULNERABLE TO: CREDENTIAL STUFFING

DANIEL MIESSLER 2022

Source:
https://danielmiessler.com/p/casmm-consumer-authentication-security-maturity-model

Service accounts for the application as a Requirement

- **Service accounts are user accounts that are intended to be used by computers**
  - **Never "human" users**
  - That means that a service account must not have access to an interactive shell or GUI
- **If an application needs to access a database…**
  - It must do so through a service account,
  - NEVER the account of a real user with the ability to log in
  - This way, if an employee leaves the company, the application is not compromised because they cannot access through their account
- **Each application must have its own service account for this type of access**
  - This facilitates its monitoring with respect to the others
  - In addition, this perfectly distinguishes actions performed by applications and users
  - Another advantage is that this way the permissions can be restricted to the maximum
  - **Something that is ALWAYS advisable!**

Requirements: Forgotten password functionality

Do not reveal if the username is correct or not

Specific page for forgotten passwords. Users ask to reset their passwords

Verify user identity

Send users a link to the email they registered with, NEVER ask for an email

Use an MFA if possible

The system registers the password request

If this happens 10+ times in less than 24 hours, block the user account and IP and generate an alert

The users click on the link to change their password

The system registers the password change

If a password is changed 10+ times in less than 24 hours, block the account and generate an alert

The system sends a password change confirmation email

Process ends successfully

Never include a link, so users do not think it is phishing

Think you've understood it all? Evaluate yourself!

?

- **You are able to do the following**
  - *Distinguish between a password manager and a secret store, what they are used for, and who uses them*
  - *Understand which policy is best for managing passwords and what to do if you must create it yourself*
  - *Understand what MFA is, its types (with their advantages and disadvantages), and which one is best from an implementation/security cost standpoint*
  - *Understand what a service account is, how and what we should use it for in our applications*
  - *Know how to raise the requirements for a forgotten password functionality service*

Other security requirements

Additional things we must consider

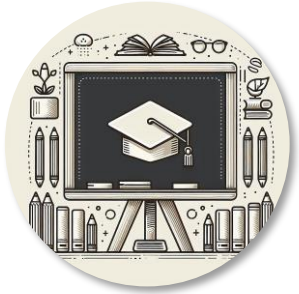
*Icons by Bing Chat AI

Other important security requirements: Minimize 3rd-party components

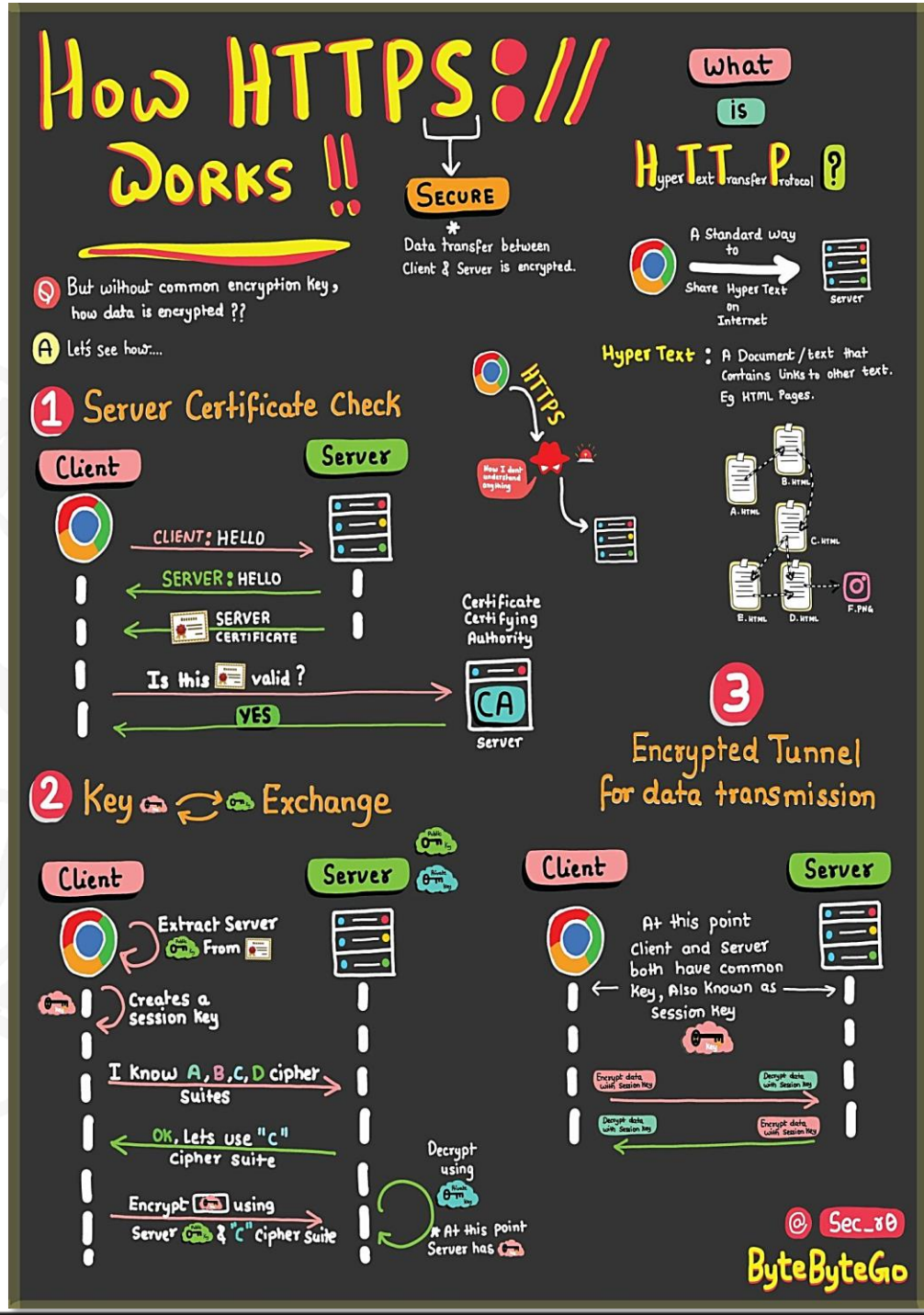
- **EVERY** code from a library, framework, plugin, 3rd party component, etc. **IT IS A RISK** incorporated into your application
  - **They are attacks on your software supply chain "supply chain attacks"**
    - **Isn't this too extreme? NO, it is the reality** (they are becoming more frequent!)
  - Any vulnerability in any third-party code you include in your app is now **YOUR vulnerability**
  - It is an application requirement to **periodically check ALL** third-party code you use
    - To verify that it has no known vulnerabilities (CVEs, see **Topic 1**)
- Luckily, there are tools that **automate** the process (**SCA**)
  - **Use them in your old app code repositories** regularly
  - **Integrate them into your CI/CD pipeline** (ASW) and stop the deployment if vulnerabilities are found
  - Another measure is to **MINIMIZE** the use of third-party components
    - Try to use only **those that are strictly necessary**
    - This is called **minimizing your attack surface**, which is another basic principle to fulfill from the design
    - You minimize the risk of being affected by **unknown (zero-day) vulnerabilities**!

Other important security requirements

Computer Security



- Use HTTP security headers (see Seminar 6)
- **ALWAYS** use HTTPS
  - We do not know what type of network users have
    - Cable, Wi-Fi, data, their own router, a shared one...
  - Using HTTPS on every application page has **advantages**
    - To avoid **network sniffing** problems and **data interception**
      - Using specialized applications (e.g., Wireshark)
    - Unencrypted traffic can be **modified "en route"**
      - By injecting malware, malicious elements, disinformation...
- The HTTPS configuration should be robust
  - And follow the latest standards, recommendations and practices
  - Which are all freely available and public here: https://ssl-config.mozilla.org/



Other important security requirements

- **Remove all comments from the web front-end**
- **Backups**
  - All the data that the page handles must be in a periodic backup
  - Also, with a weekly backup in another geographical location
  - **The process of restoring such backups should also be tested and practiced**
- **Framework security features**
  - You must use **ALL** applicable security features of your framework **ALWAYS**
  - Instead of developing your own (encoding, session management...)
  - They will always be more tested than anything you can develop
- **At requirements level, we must deal with the possible company "technical debt“**
  - Hard coding, obsolete / insecure / unmaintained or unpatched software…
  - Backdoors introduced "for agility“ ☹
  - It makes organizations slow to react to incidents, something that today cannot be allowed
  - As software engineers, we must make the company see the risks of these practices

Other important security requirements

Uploading files

- Try to avoid it and, if not possible, implement it with a tested and verified 3rd party component
- It's **probably the most dangerous functionality** (especially if you let any user do it)
- A series of recommendations must be followed if this functionality is necessary:
  - https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html

Errors and log

- Errors can **NEVER** have stack traces or database errors
- They must be captured so they do not give technical information to the user
- If your organization has a SIEM (see **Topic 5**)…
  - The logs must be written using the SIEM format and in the expected location to enable its analysis
- **Sensitive information should never be saved** in logs
  - Passwords, card numbers, etc., nothing that identifies a person
- If any suspicious activity is detected, the application must register (**log**) it
  - And **send an alert** (to the security team email, not to an individual person...)
  - **Ex.**: access attempt a restricted feature, excessive login attempts, application failures…

Other important security requirements

- **Authorization and Authentication**
  - If the application has different access types for different users (Role-Based Access Control, RBAC), these must be defined during the requirements phase (AuthZ)
    - That is, to set the roles of the users clearly from the beginning
  - The user authentication method (AuthN) must also be defined
    - That is, how their identities will be treated
- **Use parameterized queries**
  - No user or application should be able to directly interact with the database
  - Using parametrized queries prevents this
    - User input are only the parameters of pre-constructed statements
    - This prevents SQL Injection attacks (seen later)
  - Therefore, it should be a requirement to establish how these types of queries are built
    - Using the chosen framework features
    - **And enforcing its usage on every database access**

Other important security requirements

- **URL parameters**
  - Never put important parameters or private data in the URL
  - For example, usernames, IDs...
  - It should be restricted only to irrelevant data (for example, the page language)
  - In addition to giving facilities to malicious users, they are written to the log!
- **Principle of least privilege**
  - **It must be forced in all application parts**, especially in database or APIs accesses
  - When we interact with a DBMS, we must create **two user accounts**
    - One with **read-only permissions** to run `SELECT` statements
    - Another with permissions to implement a CRUD of the entities
      - NEVER database owner (DBO)
    - The first **should always be used in all possible operations**
  - A service account must be created **for each API** that the application uses (isolation)
    - And each API must have only the strictly necessary permissions for its functionality
  - Give read/write permissions to the **code repository ONLY to members of the development team**

Think you've understood it all? Evaluate yourself!

?

- **You can do the following**
  - *Understand that any third-party component included in an application is a risk, and that its use must be minimized*
  - *Know the importance of configuring the HTTP protocol in the most robust way possible (headers, encryption)*
  - *Understand that there are **several other important security requirements** that we need to consider, such as*
    - *Back up all application code and data*
    - *Use all the security features at our disposal (libraries, frameworks)*
    - *That the situation is not going to be ideal because of the concept of technical debt, and that we must consider its presence*
    - *That we should at least aspire to use roles to manage users as a requirement*
    - *The importance of parametrized queries when making requests to DBMS*
    - *That URL parameters are a frequent entry point for attacks*
    - *And that we should never grant a user and/or part of the application more privileges than are strictly necessary*

< Go to Index

Source: Bing Chat AI

Threat modeling >

SECURE DESIGN

Security aspects in application design

*Icons by Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

What are you going to find in this block?

- **In this block you will learn**
  - The main concepts that model **secure design**
    - Seeing them as a way of translating requirements into concrete elements of action
  - What threat modeling is and how is it done
    - Including the **actors** involved and the importance of the output and the end of this process

Design flaw vs Bugs & Secure design concepts

- **All the stated security requirements must be put into practice in the design**
  - So, we could avoid design flaws as much as possible (not to be confused with security bugs)
    - A design flaw allows users to do things they shouldn't
    - A bug is an error in the implementation code of a feature
  - **Remember**: the earlier a security-related design flaw is discovered, the cheaper to fix it is
  - This is achieved with **Threat modeling**
- **Protection of sensitive data**
  - **Implement all that was said in requirements regarding data protection**
  - If there are company policies regarding how to treat sensitive data, they must be followed
    - If they don't exist, it may be a good idea to create them at this point
    - E.g.: Decide how long sensitive data is saved and automatically delete it once it has passed
  - If you are going to put data in the cloud, be careful with which ones
    - Beware of current legislation, ASLEPI
  - Try to hire professional services in case leaks of our sensitive data appear online

*Icons by Bing Chat AI

SECURE DESIGN CONCEPTS

- **Never Trust, Always Verify/Zero trust/Assume breach**
  - **Only use data previously validated on the server to make decisions**
  - **Assume that any data could be compromised** when designing the application functionalities

- **Deny by default**
  - **All functions must be designed to verify that users are authorized to use them**
  - **Always identify** (AuthN) **before checking if you have permission to do anything** (AuthZ)
  - Recheck if you have access to each page or feature of the app, even if it's only reloaded
  - Always close the transaction correctly in case of failure
    - Never fall into unknown states
  - Verify identity (AuthN) and whether a feature (AuthZ) is accessed in APIs
    - Both ways (calls from an API to the application and from the application to the API)
  - Block any protocol, port, verb, or HTTP method that the application does not use

- **Isolation by design**
  - Design the system to try to deploy **only one application per server**, PaaS, or container

*Icons by Bing Chat AI

Secure design concepts

Code separation by design

- **Separate security-related from functional code**
- Different classes, another application, using an external authentication provider (E. g. Google), etc.
- Even in a separate servers or containers
  - With a **firewall** that restricts access to authorized applications only
- Also, with logs and monitoring, IDS/IPS, WAFs... (see **Topic 5**)

Separating functionality types by design

- **Separate administrators' code from the "standard" functionality**
- The same way as the previous one

Design of the application's secret management

- **Define where application secrets are kept** (DB connection strings, certificates, API Keys,...)
- Define a **secret store**, incorporate it into the CI/CD chain, and define programmatic access

*Icons by Bing Chat AI

SECURE DESIGN CONCEPTS

- **CSRF**
  - Identify the points where users perform sensitive operations
  - Purchases, password change requests...
  - To ask for his password, a captcha, or a personal token
- **NEVER use production data for testing**
  - Design separate data sources
- **Design how to protect the source code of the application at all costs**
  - **Repository access permissions and policies**, obfuscation, backup, monitoring and logging…
  - Supply chain attacks are so serious and prevalent that a specific strategy to protect your code repositories are required
  - Open source allows anyone to view the code and request changes, but changes must be properly monitored to avoid introducing malicious code
  - To do it correctly you can follow the OWASP Software Component Verification Standard
    - https://owasp.org/www-project-software-component-verification-standard/

Threat modeling

Identifying potential threats before building the application

*Icons by Bing Chat AI

Threat modeling ("Evil brainstorming")

- **Identifying potential threats** to a business, system, product or software
- **Brainstorming** where the possible dangers that we can face are thought, reviewed, and defined. E. g.:
  - Leak of sensitive data and its subsequent sale, and the impact it can have
  - How to protect against potential different attacks we identify
  - ... (any danger, any problem you imagine…think bad, think dangerous, embrace the dark side like if it was your father)

- **If we identify any we must**
  - Change the app design to counteract it
  - That a team member responsible for this accept the risk in a documented way (see DPPI)
- **It can be very complex, so here we will give an introduction**
  - To identify risks, we can rely on the elements of MITRE ATT&CK (**Topics 5 and 7**)

Threat modeling ("Evil brainstorming"): What do we need?

Computer Security

- A representative of each of the application **stakeholders**
  - **Client**
  - **Offensive security** specialists
  - **System architects**
  - **Deployment platform specialists**
  - …
- Each group must think and document the risks they see
  - *"If you were to attack the system, how would you do it?"*
  - *"Who do you think would attack the system?"*
  - *"How would you prepare for an attack?"*
  - *"What could go wrong?"*
  - *"How do we protect the user?"*
  - *"What's the worst thing that can happen?"*
  - …

*Images by Bing Chat AI

Threat modeling ("Evil brainstorming")

- **They must see it from an "evil" mentality, as if they were adversaries**
  - Turn "user stories" into "abuse stories" or "negative use cases"
    - E.g.: If an application is supposed to do X, what if it does Y instead?
  - Requires practice and attention to detail

- **There are methodologies that facilitate its implementation**
  - Attack trees: https://www.mytechiebits.com/AttackTrees
  - STRIDE: https://www.koenig-solutions.com/blog/stride-methodology-in-threat-modelling
  - PASTA: https://owasp.org/www-pdf-archive/AppSecEU2012_PASTA.pdf
- **All these methodologies allows the team to ask the necessary questions**
  - To cover most of the things that can go wrong

Threat modeling ("Evil brainstorming")

- **Threat modeling also allows thinking about…**
  - What are the valuable application assets (money, data...)
  - How they can be stolen, manipulated (reputation, DoS...), broken or misused
- **The answers are a "list of problems"**
  - It is necessary to evaluate which ones are actual **risks**, assigning them a **score** (DPPI)
  - And choose whether to **mitigate** (change the application requirements, design, code…) or **accept** them
    - Only by project management staff
  - This whole process must be **documented**
  - The method changes depending on how the design is done
    - **Cascade design** uses a session at the beginning
    - An **iterative design** works best with several shorter sessions in each iteration...

Non-invasive attack	Remote	Local	Potential damage	Probability of attack	Mitigation
Communication protocol	✓		Access to all data on network; escalation of privilege on device	High	Secure communication protocol (TLS) with a True Random Number Generator
Authentication	✓	✓	Impersonation of device on network, access to all assets	High	Follow proper authentication guidelines, fuzz testing
Code vulnerabilities	✓	✓	Control of device	High	Isolate application memory in CPU
JTAG (debug) access		✓	Full control of device; access to all assets	High	Disable JTAG; re-enable only with secure protocol
GPIO control		✓	Change device state	Medium	Monitor GPIO for unexpected behavior, fuzz testing
Side channel (DPA/TA)		✓	Access to keys	Medium	Side channel resistant cryptographic algorithm implementations
External bus monitoring		✓	Access to keys and application data	Low	Encrypt external bus interfaces

Source: https://www.synopsys.com/designware-ip/technical-bulletin/using-threat-models-2017q4.html

Think you've understood it all? Evaluate yourself!

?

- **You can do the following**
  - *Understand that the design must be a true reflection of the requirements*
  - *Understand that, when protecting data, an existing policy applicable in the context of the application must always be followed*
  - *Understand why it's best to design assuming that any data can be compromised*
  - *Understand that, when in doubt about what to do, the safest thing to do is to deny a request*
    - *Including that before authorizing something, you must first make sure to verify the identity and then the permissions over that something of that identity*
  - *Understand that it's best to design an application in isolated parts as much as possible*
    - *Using different criteria, including the phases into which development is divided (testing data other than production data)*
    - *And, in addition, protect access to the source code in those cases where it is not public*
  - *Know that in the design phase all possibility of "hardcode" information must be eliminated*
  - *Understand what threat modeling is, who performs it, and what it is for*
    - *Including understanding that, at the end of the process, we will have a list of identified risks that we need to study how to deal with*

< Go to Index

Source: Bing Chat AI

AuthN & AuthZ >

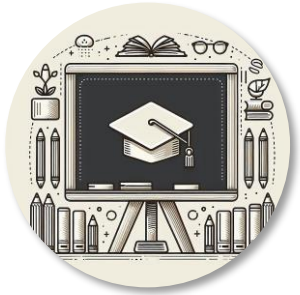
Log and errors >

Usability >

SECURE CODE CREATION

Aspects to consider in the implementation

(see Seminar 5 for concrete examples)



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

*Icons by Bing Chat AI

What are you going to find in this block?

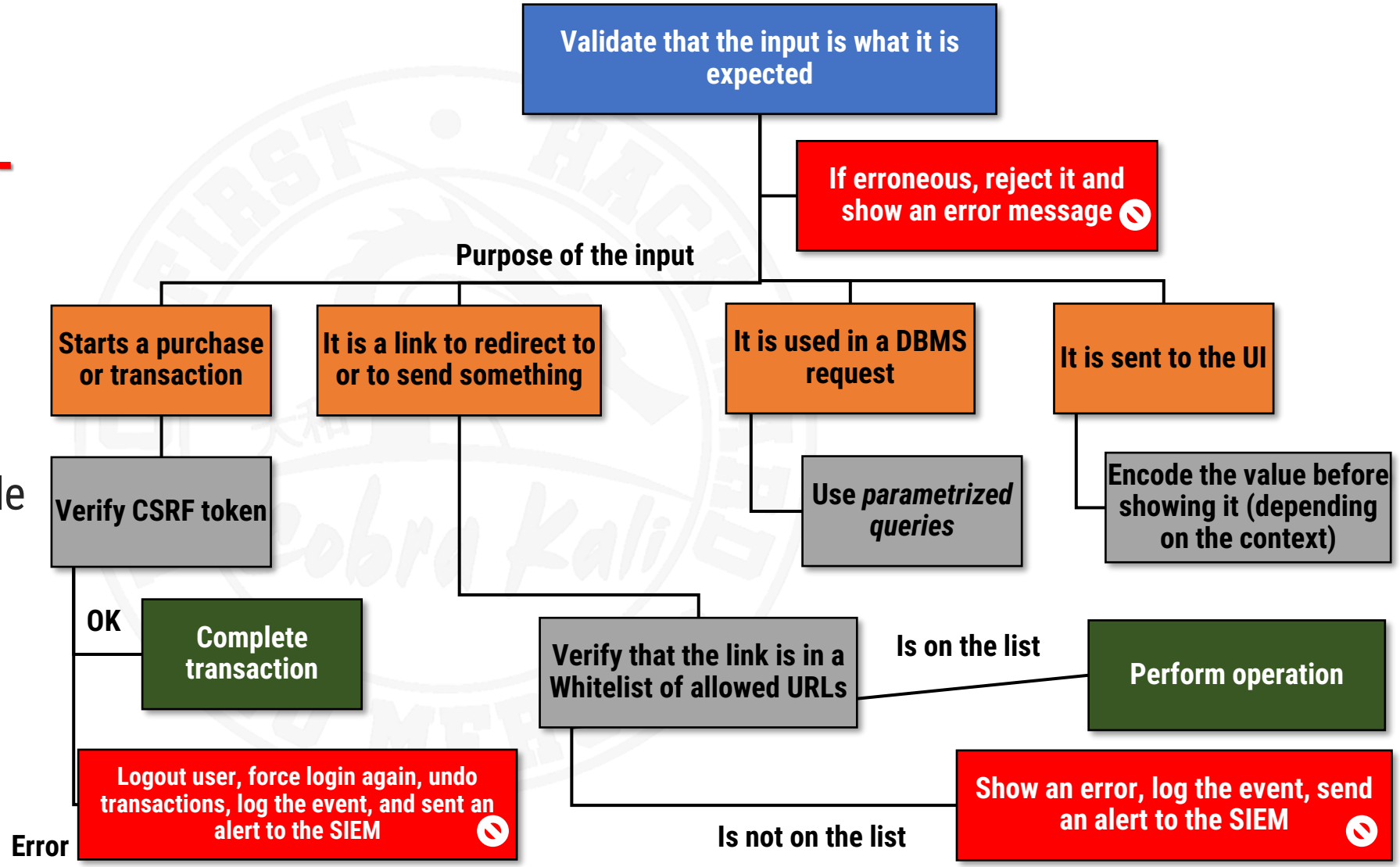
- In this block you will learn
  - Criteria for **selecting development frameworks** and languages
  - A concrete **algorithm to validate data**
  - Important elements for a **secure web implementation**
  - Criteria for implementing **secure identity management and authentication**
    - Including the importance of outsourcing the service and the best authorization policy
  - Criteria for **implementing an application's log**
    - Including its protection
  - **Security & usability** basics

Framework & Programming languages

- **In many projects this is not a real choice**
  - It is already given by the development environment
  - E. g.: If we are in a Java ecosystem, we will not have much freedom of choice…
- **But if we have the option to choose it, we must follow some rules**
  - See how long the chosen version **will be supported**
    - The more, the better (LTS (Long-Term Support) versions)
    - Never use anything near the end of its maintenance cycle
    - In general, use the latest or penultimate long support version available
  - View the **history of vulnerabilities** in a **CVE** repository (**Topic 1**)
  - **Do not** use "the latest framework or fashionable language" (for "serious" projects)
    - Until it is established, and it is known that it will have a constant and lasting support
  - **Minimize** the number of libraries, components, frameworks, etc. used (**attack surface**)
  - Scan the used version with an SCA tool to see if it has vulnerabilities if possible
    - More on this later
- **Don't use non memory-safe language such as C/C++, as they introduce problems**
  - A memory-safe alternative is Rust

Untrusted data: Validation

- Data validation is a must
- **Always validate server-side**
  - Whether or not we implement client-side validations with JavaScript
  - Good for usability, terrible for security
- Before any possible usage of user input
- Use the following schema as a guide



HTTP Verbs

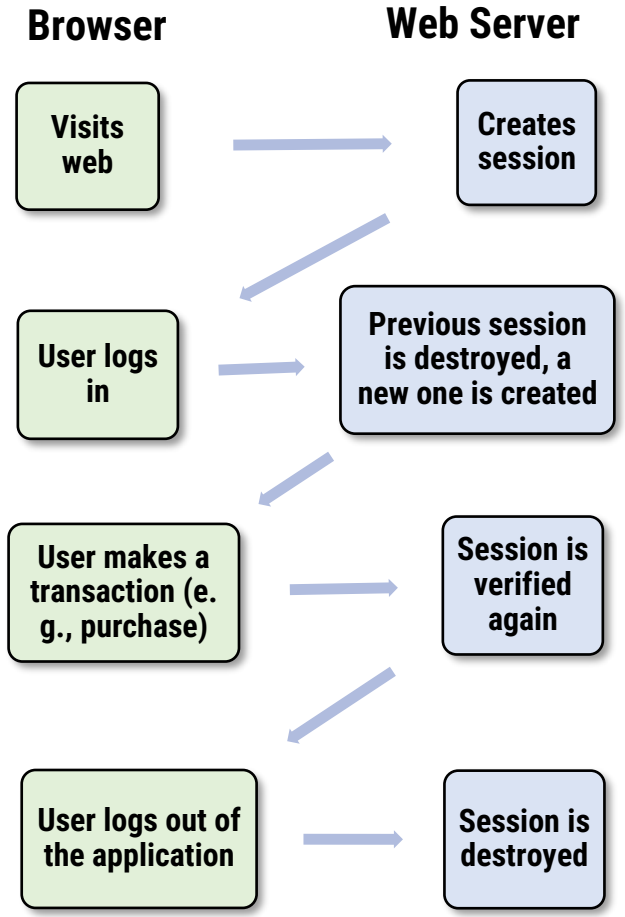
- **If an application uses the HTTP protocol…**
  - There are several actions that it can perform that are modeled through HTTP verbs (`GET`, `POST`, `PUT`, `DELETE`, … see SEW)
  - **Remove support for any verb not used in the application**
    - To avoid giving too much information and possible attacks
  - Some HTTP protocol extensions use additional verbs
    - And we need to make sure we don't remove legitimate ones
  - A typical web application normally only uses `GET`, `POST`, `OPTIONS`, and `HEAD`
- **To disable all unused verbs, it is recommended to follow the CIS Benchmarks**
  - We can do it at the web server or at the framework level
  - Remember **Topic 4**!

HTTP Method	Request has body	Response has body	Safe	Idempotent	Cachable
GET	NO	YES	YES	YES	YES
POST	YES	YES	NO	NO	YES
PUT	YES	YES	NO	YES	NO
DELETE	YES	YES	NO	YES	NO
TRACE	NO	YES	YES	YES	NO
OPTIONS	NO	YES	YES	YES	NO
CONNECT	NO	YES	NO	NO	NO
PATCH	YES	YES	NO	NO	NO

Be careful specially on REST applications.
Source: https://intmain.co/http-request-methods-rest-api-verbs

SESSION MANAGEMENT

- **Sessions are tokens transmitted between client and server**
  - They uniquely identify each user
- **Correct management of session IDs requires this**
  - Must be at least **128 characters long**
  - Must be **unpredictable** (random) to **avoid ID precalculation**
    - A validated and non-proprietary random ID generator must be used
  - Users must receive a new ID **each time they authenticate**
  - **Use, if it exists, the framework session manager**
  - Must have an **expiration date** even if the user is still in the app
  - Should only be **transmitted over encrypted channels**
  - **Never accept** one that has not been generated by the application
    - This should generate a log/SIEM alert and block the IP (security incident)
  - Must be **regenerated** at each authentication and re-authentication
    - Or any other event that changes the user's privilege level
  - **Many frameworks have this**: learn to use it and **don't program it yourself**



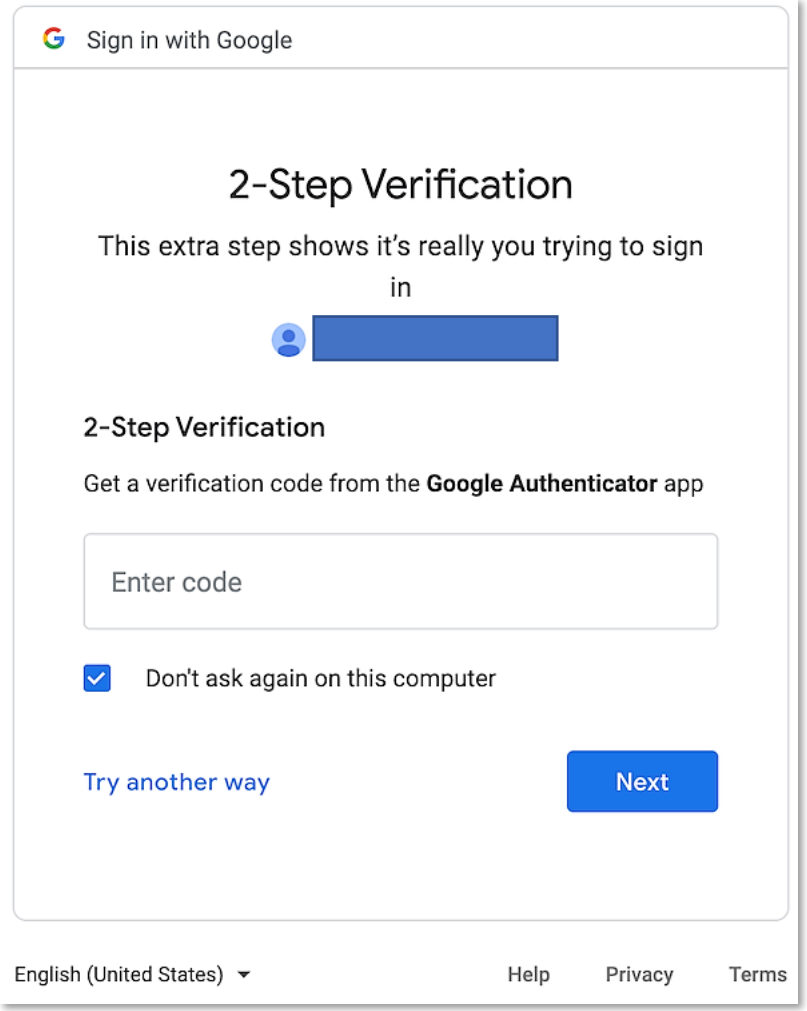
Identity and authentication (AUTHN)

How to implement identity and permission checks for our users

*Icons by Bing Chat AI

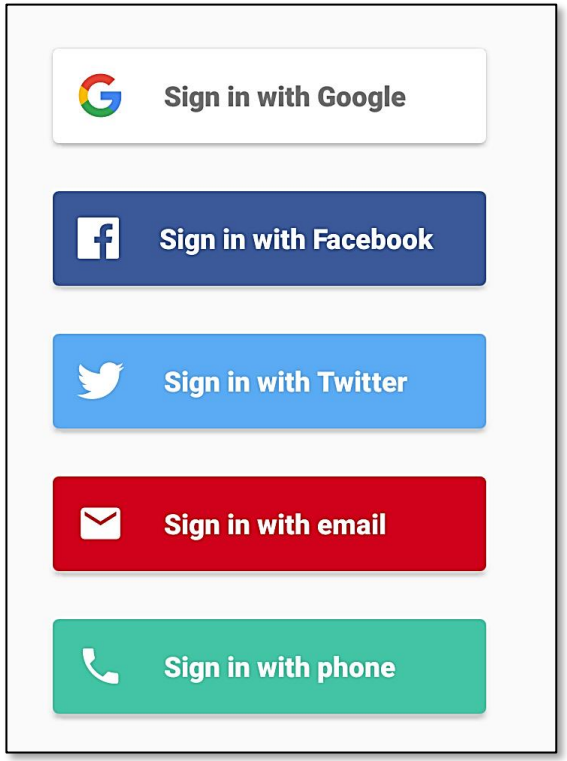
AVOID WRITING YOUR OWN AUTHENTICATION SYSTEM

- It is common to have a **Microsoft Active Directory**-based identity system to identify out users
- Other typical cloud providers have their own authentication systems. E.g.:
  - **Google**: https://cloud.google.com/docs/authentication
  - **Amazon**: https://aws.amazon.com/es/cognito/
- We must choose and use one that is compatible with our requirements
  - And the business where the application is going to be deployed
- If this is not an option because you have very particular requirements…
  - It is best to use an established protocol such as OAUTH: https://oauth.net/getting-started/



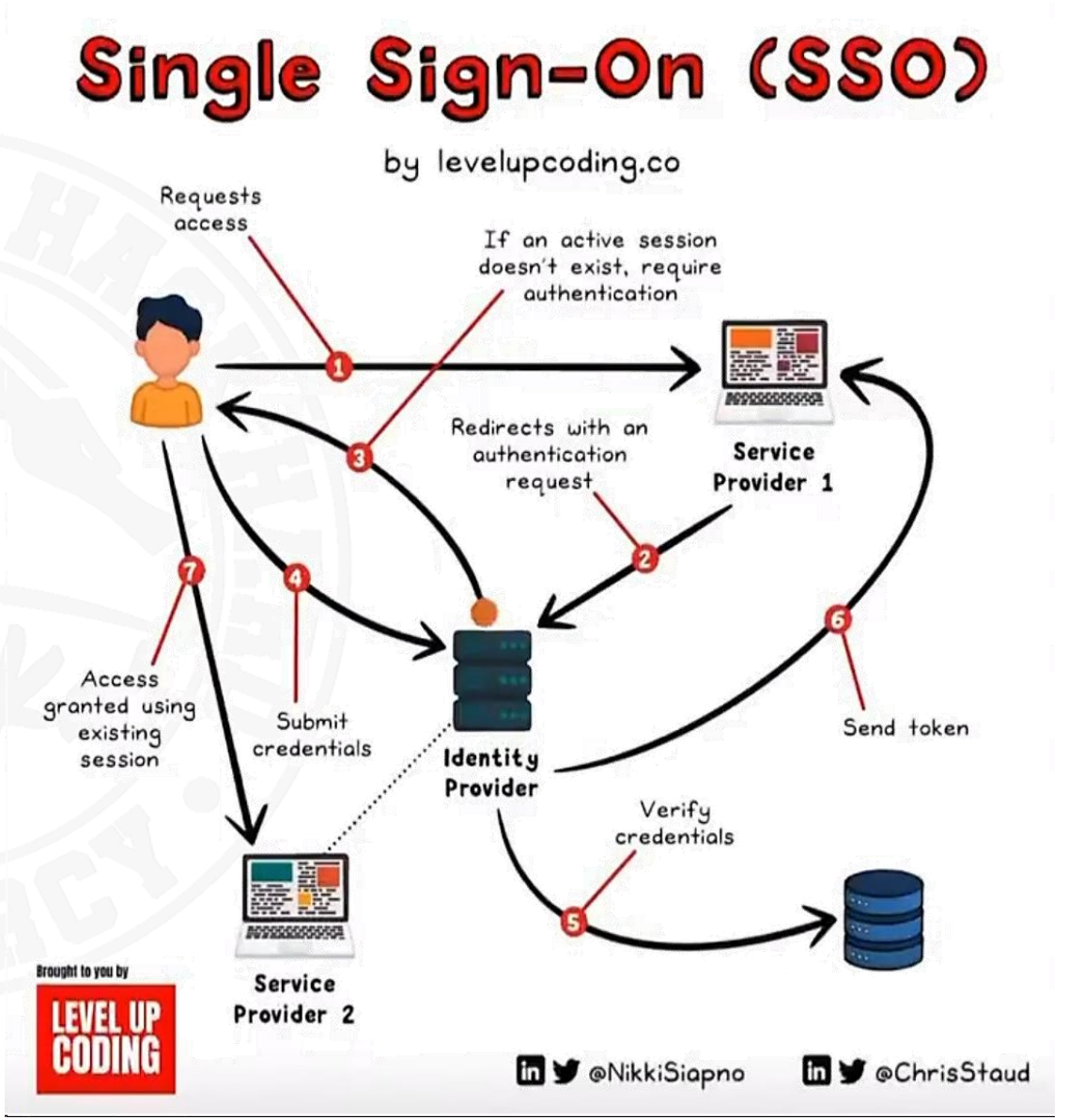
Always use a 3rd-party authentication provider

- **Google, Microsoft, LinkedIn, Twitter, Facebook...**
- **Advantages**
  - Deploys **quickly** (there are deployment guides)
  - We **get rid of the responsibility** of maintaining or testing that code
  - Authentication code is more **secure**, it's more tested
  - Users **can trust** these systems more
- **Disadvantages**
  - Some users don't want to share data between different companies
    - And DO want to have separate logins for each site
  - Some users may not use a particular provider
    - **Consider supporting multiple ones** to counteract this
  - Its usage could add a cost
  - These services **may collect personal information** from users
    - Unacceptable in certain contexts
  - A breach in the third-party system is automatically one **we MUST assume!**



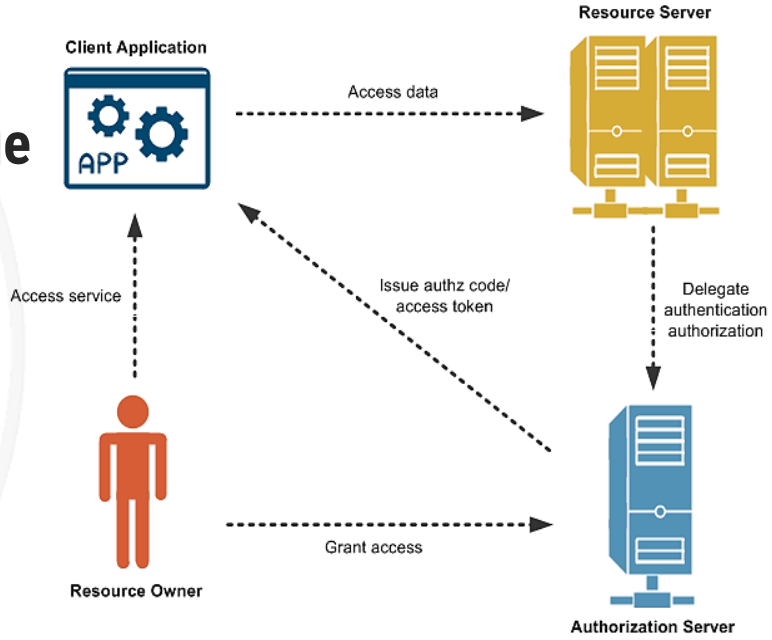
ALWAYS USE A 3RD-PARTY AUTHENTICATION PROVIDER

- **Using a third-party provider also makes it easier to use Single Sign-On (SSO)**
  - The user correctly enters their username and password (and 2FA) into an authentication service
  - By doing so, **you have access to one or more applications** linked to that authentication service
- **Makes account management easy**
  - It is no longer necessary to have several, one for each application
  - Improves user experience



AUTHENTICATION LIBRARIES

- Purchase or use a library or software system to implement authentication
- Advantages
  - **Faster and much more suitable** than creating your own one
  - The library only leaves us the responsibility of **implementing its usage**
    - Not designing the authentication system
  - If the library has adequate support, it can be very secure
  - **User data is in our possession**
    - Which can be convenient in some scenarios
- Disadvantages
  - Could have an **associated cost**
  - A **vulnerability** in the library is automatically ours until we update/mitigate it
  - **We are responsible for user data**
    - With the legal issues that it entails and that we must consider (see ASLEPI)



Source: docs.oracle.com

Authorization (AuthZ)

- **Determines what an authenticated user can or cannot do in the system**
  - Once we know who he is (**AuthN**)
- **RBAC (Role-Based Access Control)**
  - **A series of user roles are created in the design**
  - Each are assigned the minimum permissions to do a certain job
    - e.g.: Tester, developer, database administrator...
  - Users are assigned one or more roles
    - Only these roles determine what users can or cannot do
  - When users leave or get promoted, their roles are removed or changed
    - And, with it, automatically change their permissions (like Windows groups, ASR)

- **DAC (Discretionary Access Control)**
  - **The resource owner can freely give or remove permissions**
  - Like the Linux file system
  - But it can be applied to more abstract entities, such as "users on the same network", etc.

*Icons by Bing Chat AI

Authorization (AuthZ)

- **MAC (Mandatory Access Control)**
  - **Access to a resource is based it its privacy level**
  - And whether the user has sufficient privilege to access that level or not
  - Identical to what was described in topic 3: OS Security

- **PBAC (Permission-Based Access Control)**
  - **Users are granted certain permissions on the information or systems**
  - Which are checked when accessing it
  - No roles are used
  - Permissions are granted to individual users or multiple users at once

*Icons by Bing Chat AI

Error handling, logging, and monitoring

What to do with the information that we can obtain from our running application

*Icons by Bing Chat AI

Error handling, logging, and Monitoring: Handling rules

- **ALL** application errors must be captured and handled
  - Doing a global `catch` (on the `main`) is a good idea as a last resort
  - To capture unexpected errors and deal with them appropriately (**don't leave it empty**!)
- Internal error information (stack traces...) should **NEVER** be disclosed to users
  - In fact, errors **should show no technical information** to users (never product versions, technologies...)
- As we said, never say if the username or the password is wrong
  - Give an identical error in both cases (**prevents enumeration**)
- Security errors (login, access control, validation failures...) **must create an alert**
  - Ideally, the application logs are sent to an IDS/IPS or a SIEM that analyzes them (next slide)
  - They can be provoked by running an automatic vulnerability scanning tool (**Seminar 3**)
- When the error includes information from other systems…
  - Encode the content, delete all non-printable characters, and limit the size of the logged information

Error handling, logging, and monitoring

- **There are log monitoring software that analyzes logs sent to them looking for problems (e. g.: SIEM, seen in Topic 5)**
  - Many of them analyze them using different ML/AI techniques (e. g.: rules, SI course)
  - We must use a log format compatible with our SIEM (**Topic 5**)
  - **General rules**
    - Log failed/successful logins, and errors
    - Record brute-force login attempts (e.g., more than 10 failed login attempts in less than 1 minute)
    - Generally, log all security-related events
      - Authenticated user, user blocked after several failed attempts, failed validations…
  - **Rules about the information they contain**
    - **Remember**: logs **must not contain private or personally identifiable information**
    - The **type** of event that has occurred (security/other)
    - **When** it happened (**timestamp**)
    - **Where** it has occurred, both at the address and system level (system, IP, URL, subdomain...).
    - The **result** of the event
    - If possible, identify **the user** or **source** associated with the event

Error handling, logging, and monitoring: Log protection

- Logs must be protected from unauthorized access
  - And backed up as part of the company's backup strategy
- Log access must also be recorded and monitored
- They must be stored encrypted and on a secure server
  - See **SNI**, **Topic 5**
- Logs must be accessible by incident response teams
- They must be deleted periodically
  - Following the company policy for sensitive information
  - And, of course, the legal regulations of our country
- This links with the **Computer Forensics and Audit** (**IFA**) optional course

*Images by Bing Chat AI

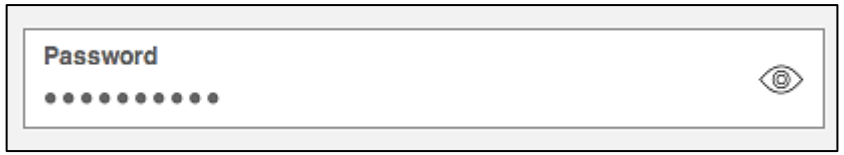
Security and usability

Security must be usable, or our software could fail

*Icons by Bing Chat AI

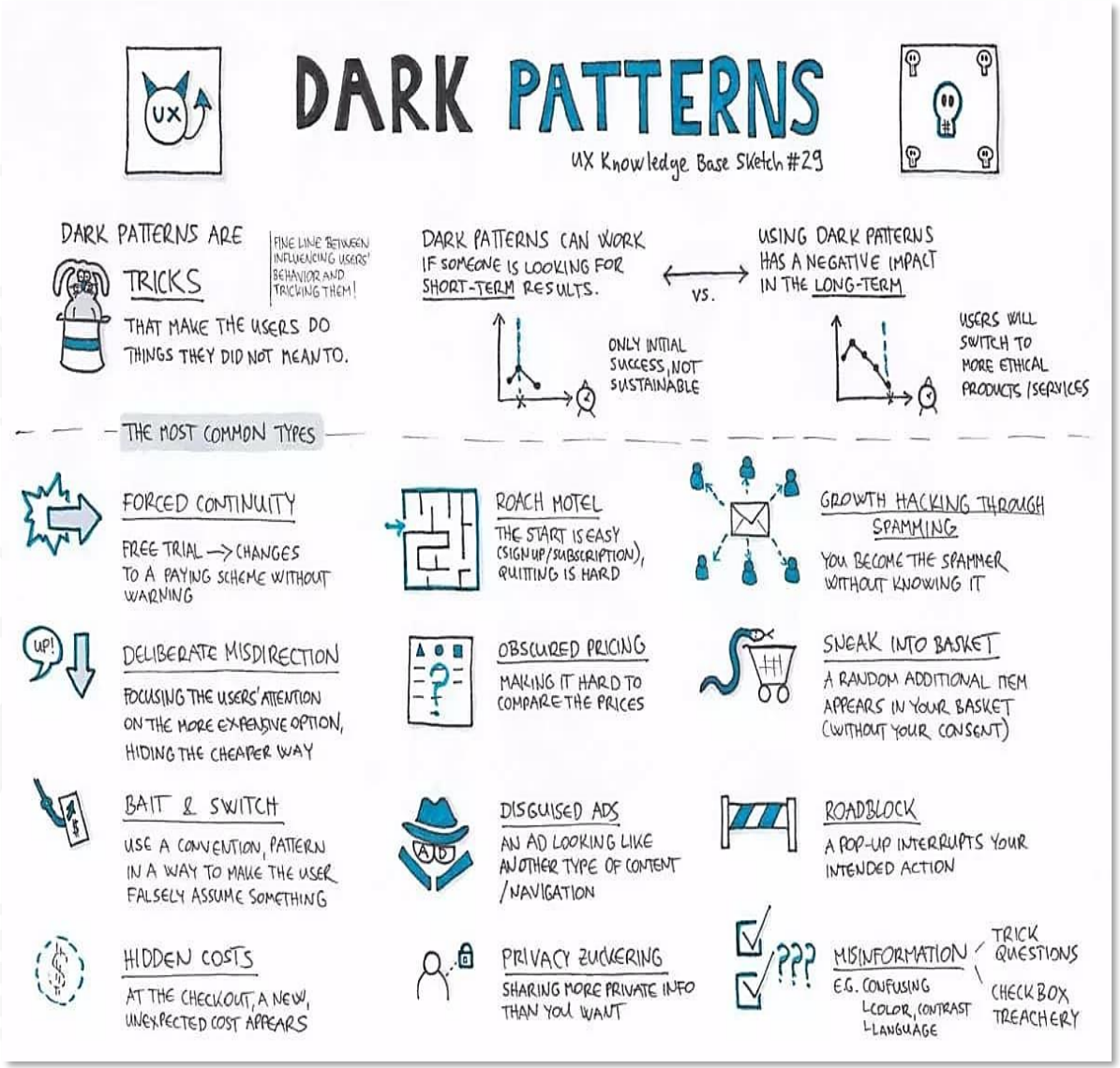
SECURITY AND USABILITY

- **If the security measures of an application are difficult to use, users will try to avoid them**
  - Or go to rival products!
- **It is necessary to try to make security usable. E.g.:**
  - Allow to unlock a device or app using fingerprint or facial recognition, instead of a password
  - Teach users to create passwords from phrases, instead of using typical complexity rules
  - Teach users how to use **Password Managers**
    - To use them, your application **must not disable copy & paste** on password fields
  - ...
- **This is a complex topic that exceeds the competences of this course**
  - And it is related to what was taught in the CPM course
- **Some materials on UI design from a security perspective**
  - https://uxdesign.cc/designing-uis-with-security-in-mind-f4f6f265573a
  - https://www.w3.org/TR/UISecurity/#intro



ANTI-USABILITY: "DARK PATTERNS"

- **Designing programs to "trick" users into doing things they don't want**
  - Buy products, service subscriptions...(e.g.: extra cart items, fake “x persons are viewing this”...)
  - Some can compromise user security!
- **Assume users do not read the whole web**
  - And/or that they will behave in a given way
  - **Pages are designed to deceive**; they seem to say one thing, but the reality is different
- **As users, we must know these practices to defend ourselves from them**
  - Ant to avoid reproducing them in our applications
- **List of "dark patterns"**
  - https://www.darkpatterns.org/types-of-dark-pattern



Source: https://carlosguadian.net/2021/06/07/dark-patterns-o-como-obligarte-a-hacer-lo-que-no-quieres/

Think you've understood it all? Evaluate yourself!

- You can do the following
  - *Understand the criteria for choosing a development framework or language correctly*
  - *Understand the algorithm with which you should implement the validation of all input data*
  - *Understand that you need to minimize the number of HTTP verbs your app supports*
  - *Understand the algorithm to implement when managing sessions*
  - ***About identity and authentication***
    - *Understand why you should never create your own system to authenticate users*
    - *Understanding the concept of Single Sign-On*
    - *Understand the types of authorization you can implement, their advantages and disadvantages*
  - ***About log***
    - *Understand what should never appear in your app's log*
    - *Understand that your application is likely to be in a larger ecosystem, and that the log format needs to be adapted to it*
    - *Be very clear that the manipulation and alteration of logs by unauthorized people leaves us blind to attacks that may occur*
  - ***About security and usability***
    - *Understand how to implement the application to facilitate the use of password managers*
    - *Understand what a dark pattern is and what it's used for*

< Go to Index

Source: Bing Chat AI

Static tools >

Dynamic tools >

AST (Application Security Testing)

Techniques and tools to check software security

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

*Icons by Bing Chat AI

What are you going to find in this block?

- **In this block you will learn**
  - What **SCA** tools are
  - What **SAST** tools are
    - Including how to have both easily if you use GitHub
  - What **DAST** tools are
  - What **IAST** tools are
  - The importance of integrating all these tools into a **SecDevOps** development cycle

TESTING

- **Testing activities are critical to detect security issues**
  - It uses the same method given in CVVS to do functional tests
  - But for the mentioned "abuse stories" or "negative use cases"
- **The security of our software products may be evaluated by third parties**
  - Normally, to be accepted in various countries/contexts
  - The Common Criteria (functional security testing for security requirements) are used for this
    - https://www.commoncriteriaportal.org/index.cfm
    - Each country has a certifying body; the **Centro Criptológico Nacional** (CCN) does it in Spain
- **Testing involves introducing AST tools into the software development process**
  - Which is especially relevant if we follow a DevOps development model with a CI/CD (ASW)
    - These tools should be integrated into our pipeline
    - We will turn it into SecDevOps
  - Some security tasks in each phase can be performed in parallel with others

Static tools: SCA and SAST

Analyzing the code prior to its execution

*Icons by Bing Chat AI

SCA (Software Composition Analysis) tools

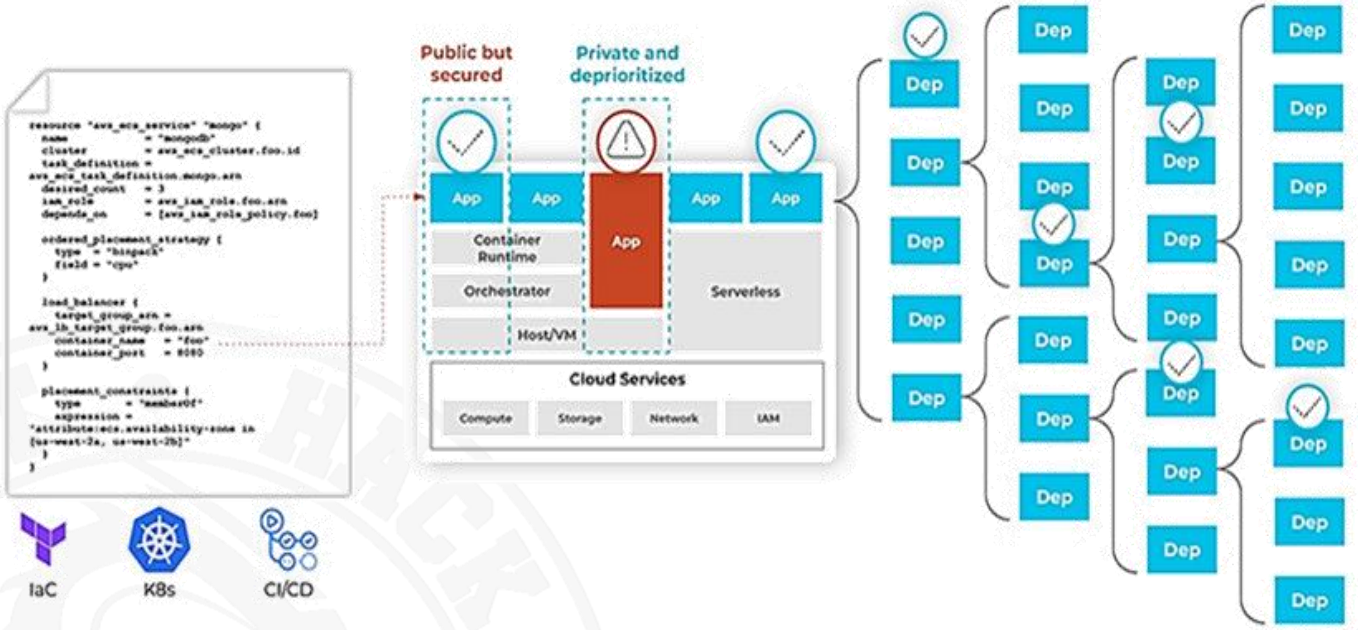
- **Manage the usage of open-source components**
  - Automatically scan open-source code we use, including artifacts such as containers
- **Have these purposes**
  - **Inventory**: Collect all used open-source components, direct, and indirect dependencies
    - Enables knowing every software component in use, which is key to controlling your security
  - **License Compliance**: provide information about each of the identified components
    - Includes if the license type is compatible with the organization policy and authorship attribution
  - **Security Vulnerabilities**: Identifies known vulnerabilities in these components (**CVEs**)
    - **It is their main function** (see next slide)
    - Reports if the application calls vulnerable code, suggest mitigations, and manage updates or patches
  - **Advanced features**: Force all open-source components to comply with company policies
    - With automated responses (request component approval, prevent application construction…)
    - Warns about vulnerabilities in a component before making a pull request and allow it into the system
- **Examples**
  - https://sourceforge.net/software/software-composition-analysis-sca/
  - https://www.paloaltonetworks.com/cyberpedia/what-is-sca

*Icons by Bing Chat AI

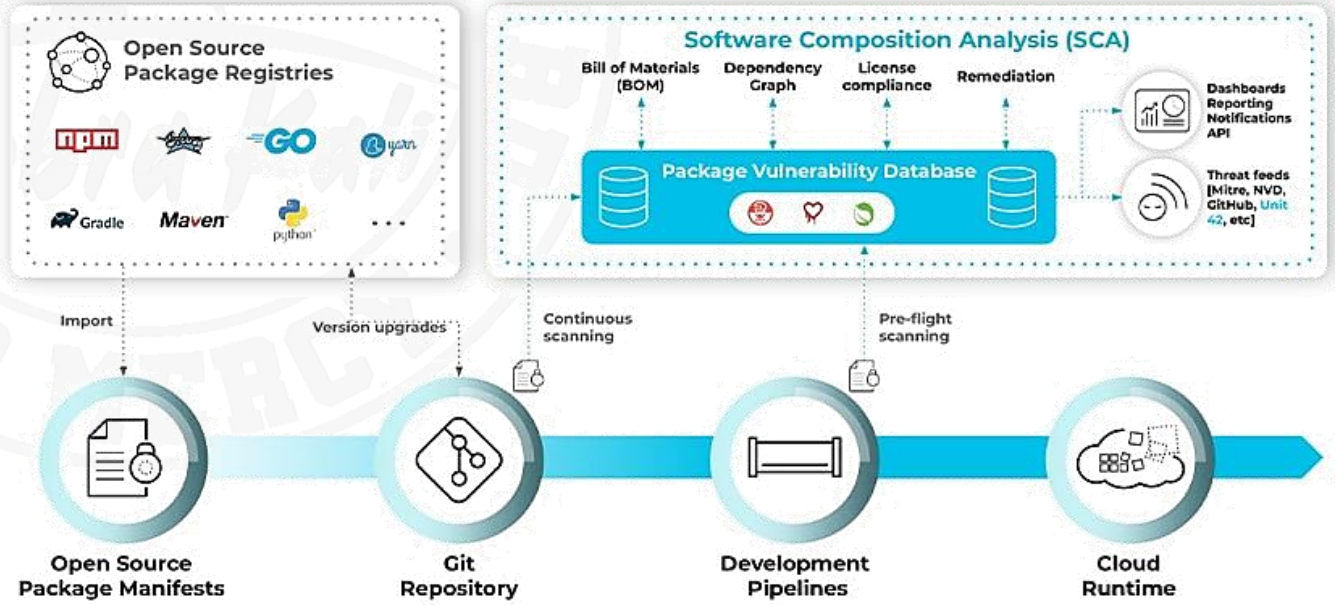
SCA TOOLS

Computer Security

- **A modern application have many dependencies**
  - Several are open source, so we can revise their code
- **SCA tools analyze the code of those open-source dependencies**
  - Avoid introducing their vulnerabilities in our application
  - If no SCA is used, sooner or later a vulnerable dependency will "poison" our software ☹
- **All this analysis process can be automatically integrated in the development pipeline**



An SCA tool analyzes the code of all the open-source dependencies of the application within the application pipeline. Source: https://www.paloaltonetworks.com/cyberpedia/what-is-sca

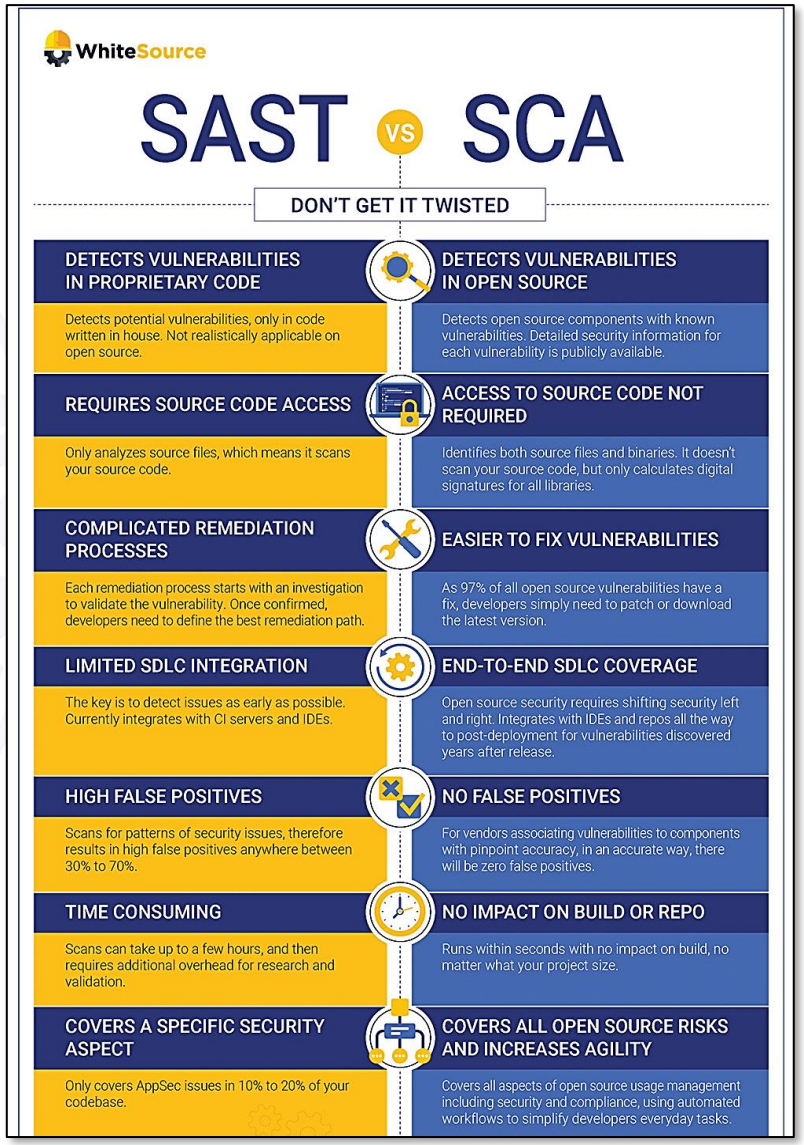


SAST (Static Application Security Testing) tools

Computer Security

- **Test application's internal structures and source code**
  - Not functionality (white box tests)
- **Advantages**
  - **Support many languages**: PHP, Java, .NET, C, C++, C#...
  - Discover complex vulnerabilities **during early development stages**, that can be resolved quickly
  - **Detail the problem**, including the line of code
  - Can be integrated into an existing SDLC environment
- **Disadvantages**
  - Scaling them to major developments can be challenging
  - Tend to be **inaccurate**: many false positives and negatives
    - Even worse with dynamic typing languages
  - Cannot test applications in real environments
    - **Do not detect vulnerabilities in the application logic**
    - Or insecure configurations

Source: https://www.mend.io/resources/blog/sast-vs-sca/

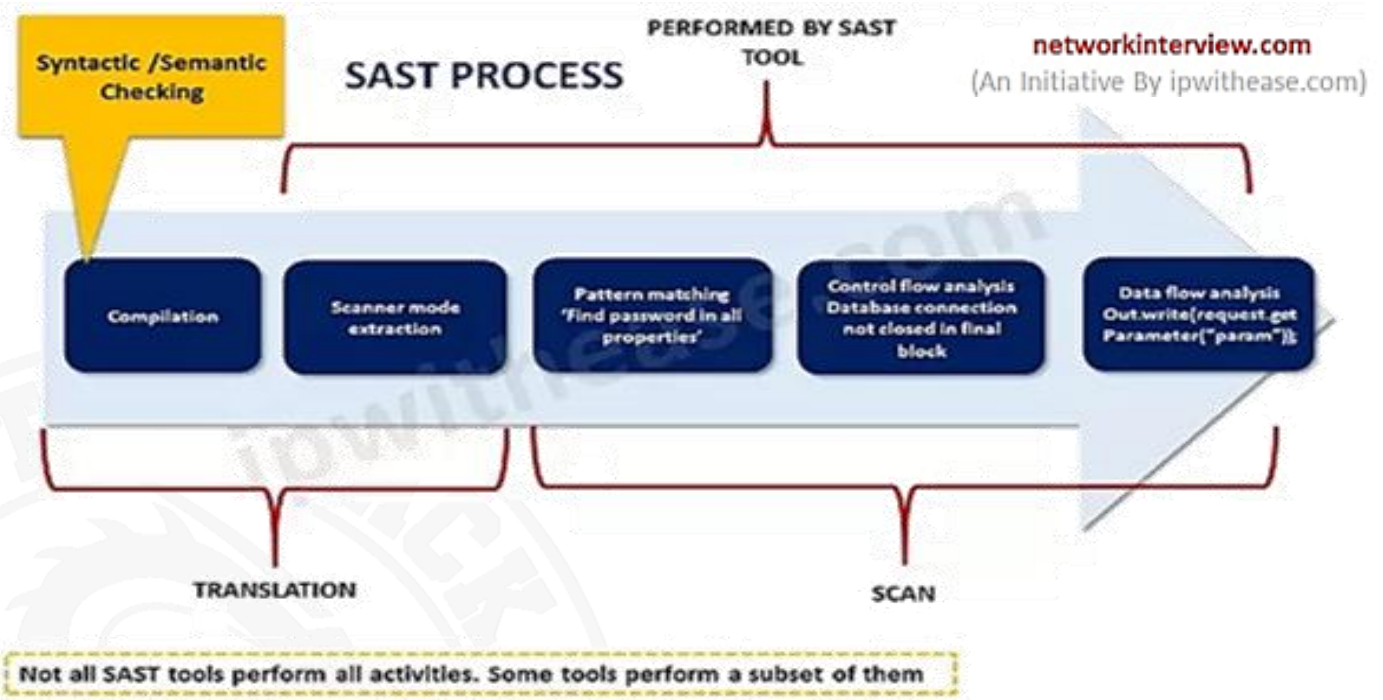


*Icons by Bing Chat AI

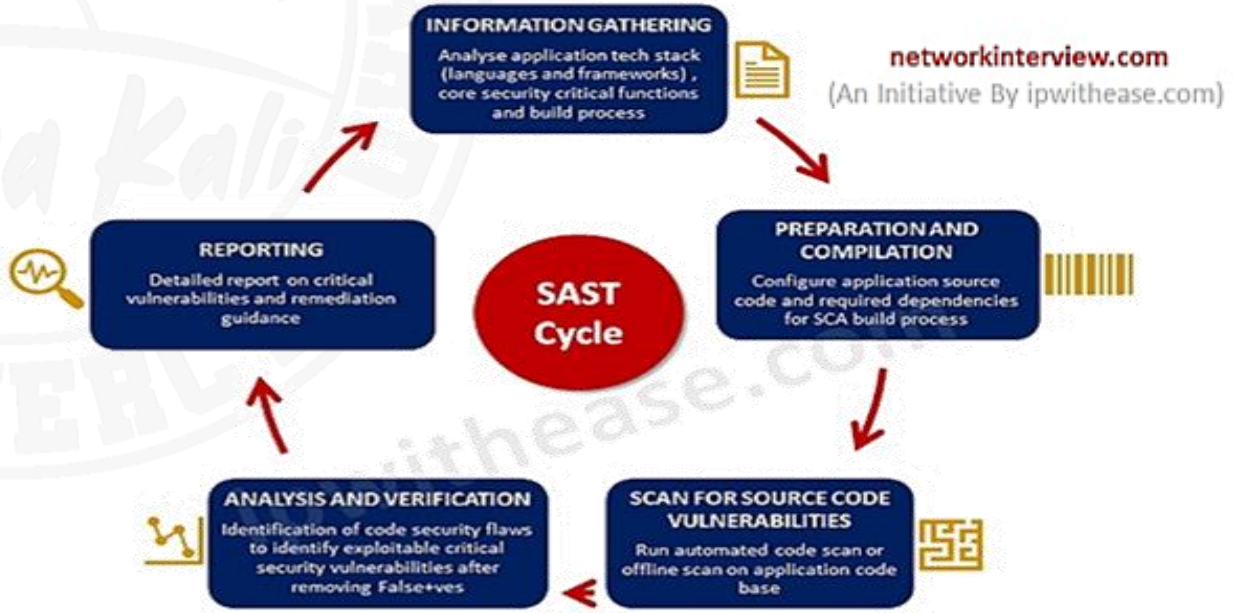
SAST TOOLS

Computer Security

- You can consider a SAST tool as an extension of the software compilation process
- Instead of type errors, they look for known bad coding patterns
  - It is like a language processor, but specialized in “pitfalls”
- After analyzing the code, it outputs a report, independent from the compiler output
  - With all the details about found problems and suggested solutions

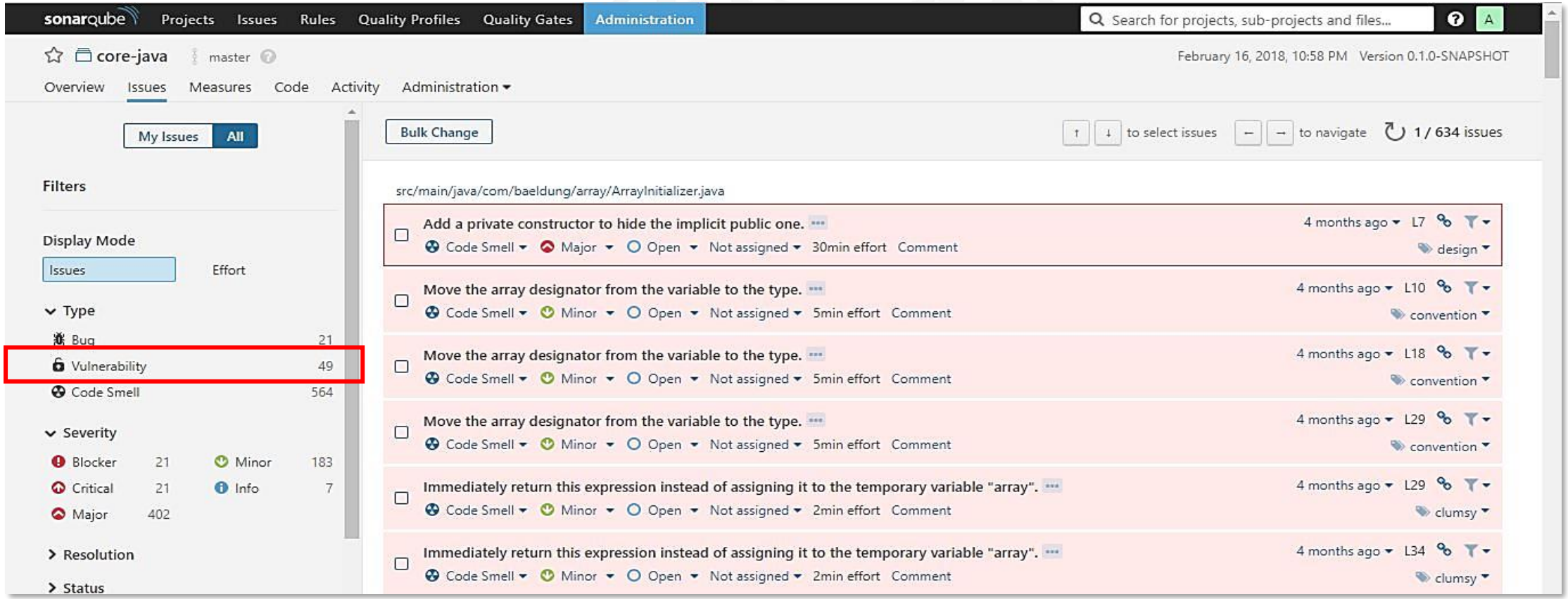


A SAST tool works after syntactic and semantic analysis (DLP). Source: https://networkinterview.com/sast-static-application-security-testing/



SAST TOOLS

- **A very popular SAST tool is SonarQube (seen on Lab 10)**
  - More tools: https://www.g2.com/categories/static-application-security-testing-sast
  - Analyzing your **TFG** with SCA and SAST tools is a good plus (wink, wink ☺)



SonarQube's report revealing a terrible development process (allegedly ☺): https://www.baeldung.com/sonar-qube

➢ Exercise **L10B3_SASTSCA**

Easy and fast SAST and SCA with GitHub

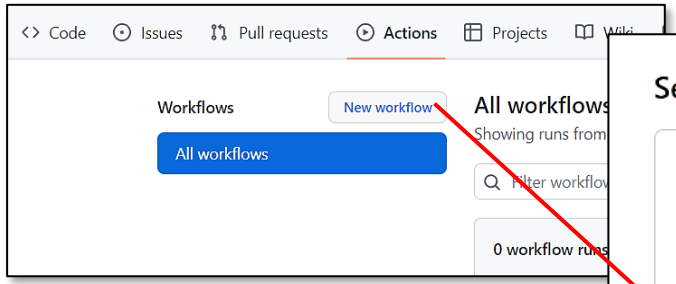
*Logo oficial de GitHub

Easy SCA and SAST with GitHub actions and workflows

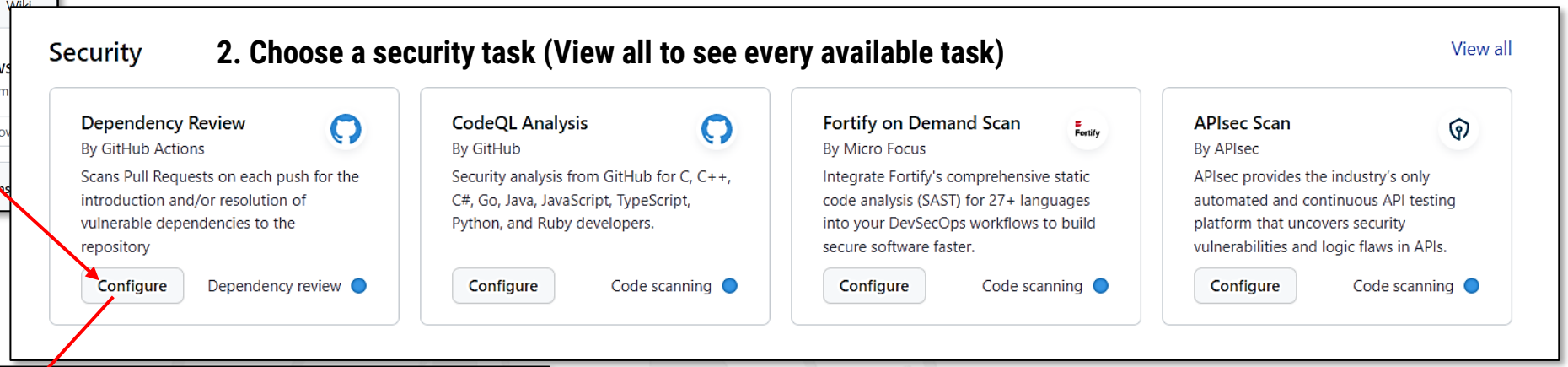
- **Using GitHub gives you the ability to use SCA and SAST features very easily**
  - Only for public repositories at this moment
- **It integrates SCA and SAST scanners as part of their GitHub Actions**
  - Among **many other functionalities**
  - Therefore, you can launch a lot of automated processes to improve our code
  - One of the most popular free ones is **CodeQL**
  - Follow: https://docs.github.com/es/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/setting-up-code-scanning-for-a-repository
  - Check other available GitHub Actions to find more AST tools that may fit your development
    - And improve your security!

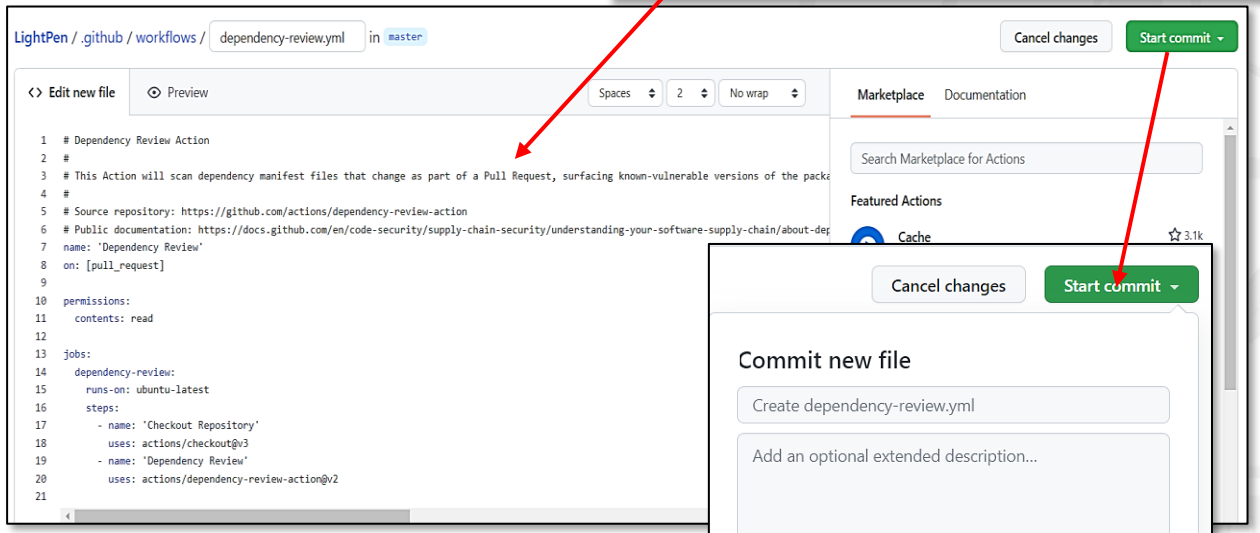
Easy SCA and SAST with GitHub Actions and Workflows

Computer Security



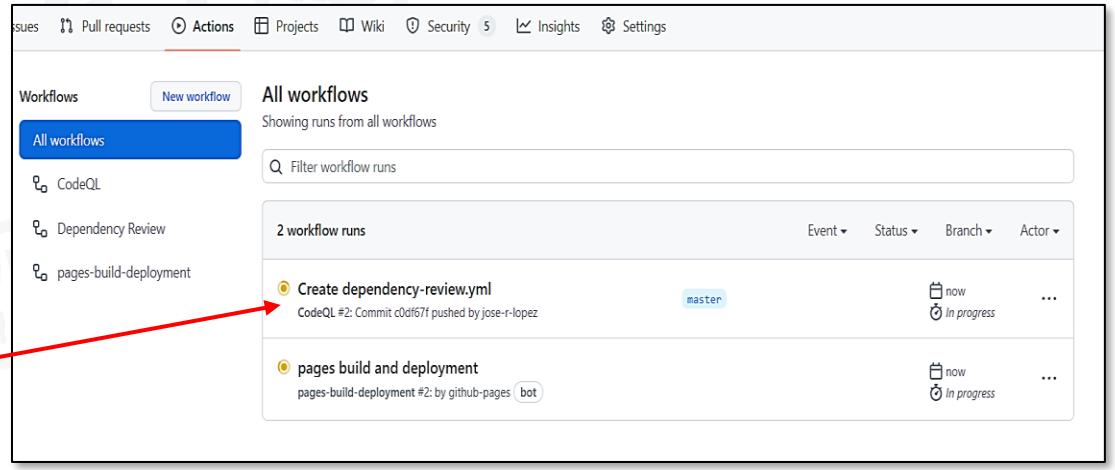
1. Create a new workflow





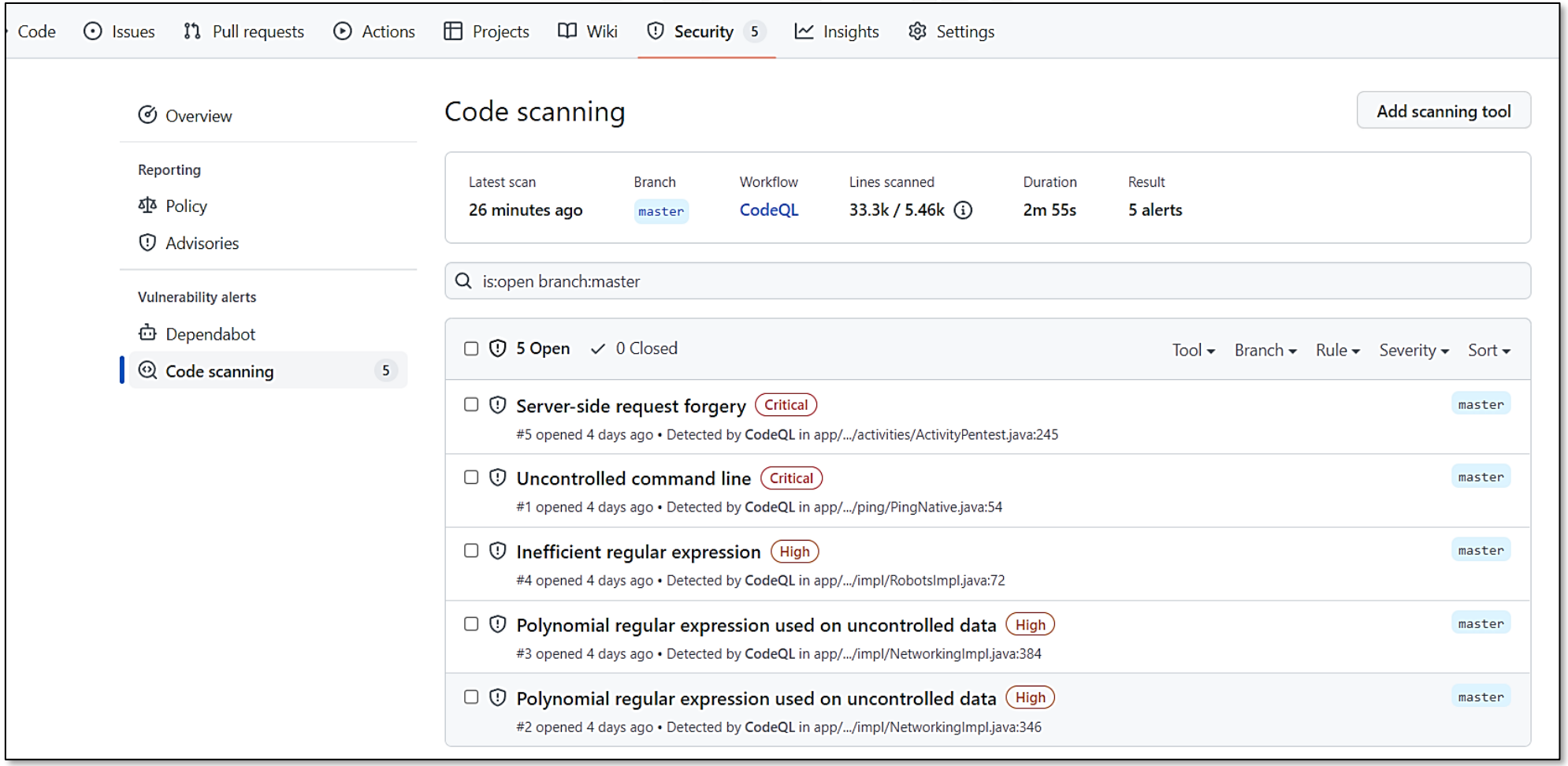
3. **Dependency review** makes a SCA and a SAST Scan with the **CodeQL** product. Accept the created file and commit it

4. Go to “Actions” and wait for the task to finish



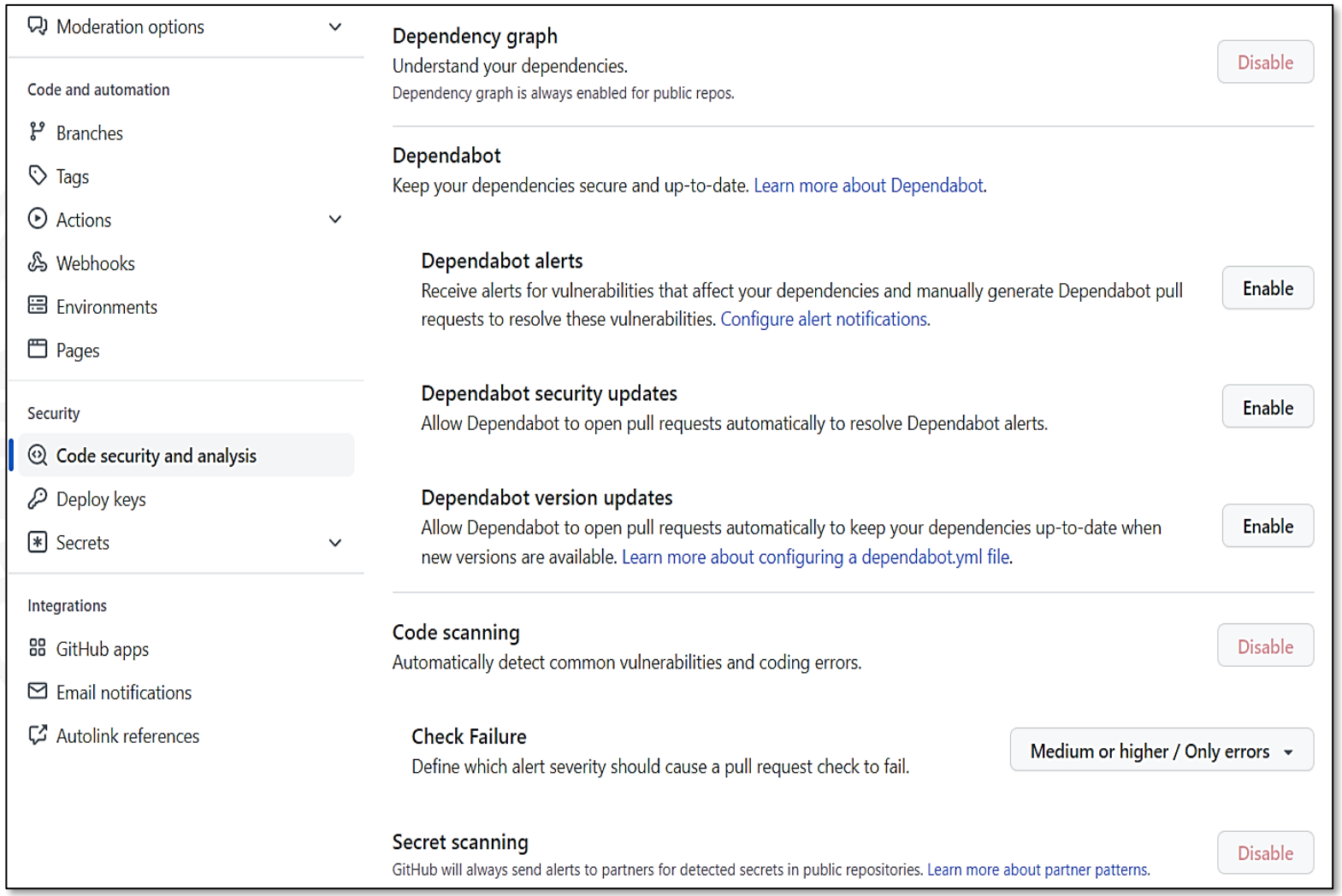
Easy SCA and SAST with GitHub Actions and Workflows

- And just like that we have SCA (Dependabot) and SAST (Code scanning) features for our code!



Easy SCA and SAST with GitHub actions and workflows

- The **code security and analysis** options also activate automatic SCA and SAST
- Also, **secret scans**
  - Hardcoded private information in your code will be detected!
- These options are **KEY** for implementing secure code!



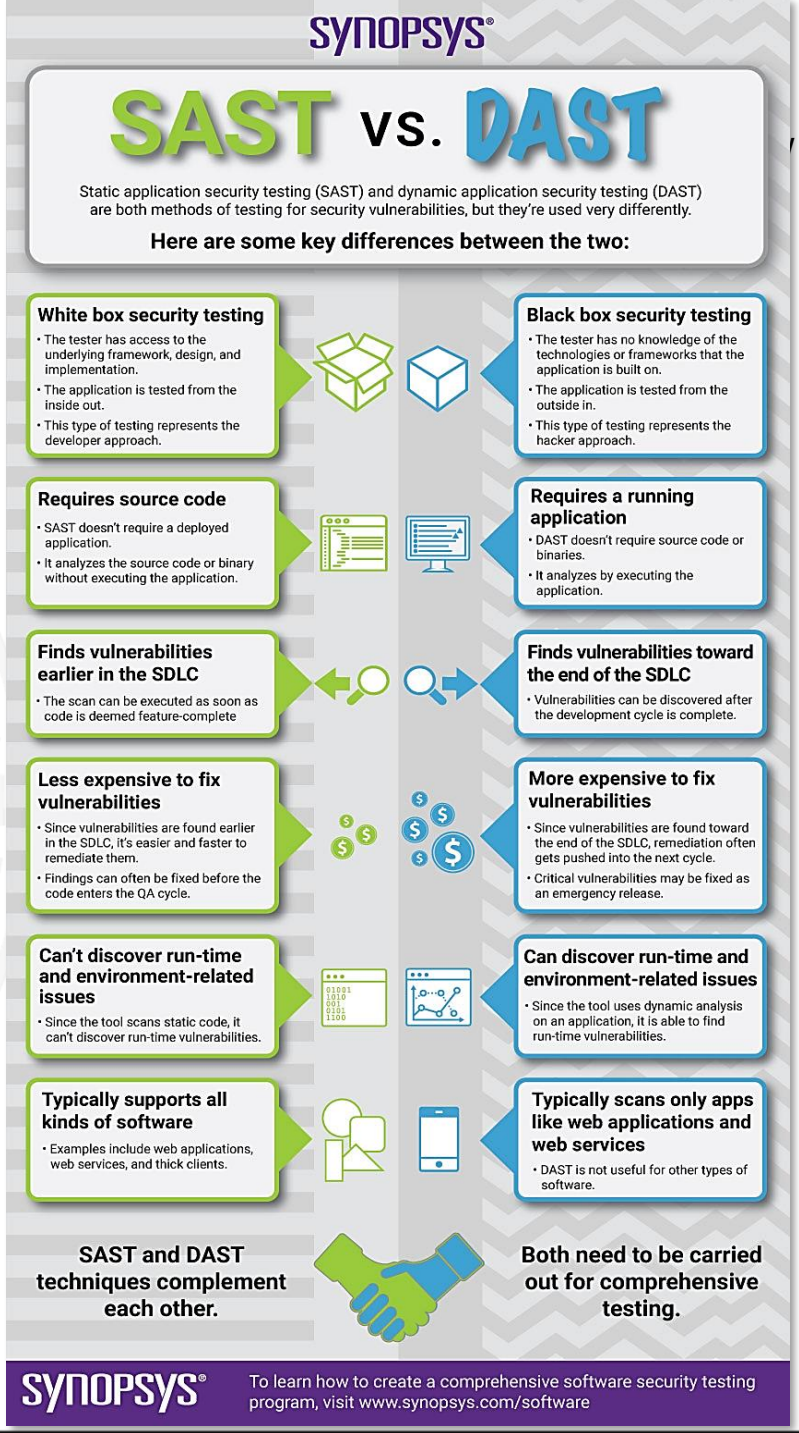
Dynamic tools: DAST and IAST

Analyzing the code after its execution

*Icons by Bing Chat AI

DAST (Dynamic Application Security Testing) tools

- **Black box test** method that introduces errors to test the resulting execution paths
  - Requires no source code or binaries; scans the running application
- **Advantages**
  - Unlike SAST, allows you to detect **runtime issues**
    - Authentication and network configuration errors, issues after logon…
  - There are fewer false positives
  - Supports all programming languages and/or custom frameworks
- **Disadvantages**
  - **Do not provide information about the vulnerability causes**
  - Not for the 1st stages of the project (need the application running)
  - It does not simulate a part of the possible attacks
- **Want to incorporate a free one to your pipeline?**
  - Check OWASP ZAP: https://owasp.org/www-project-zap/

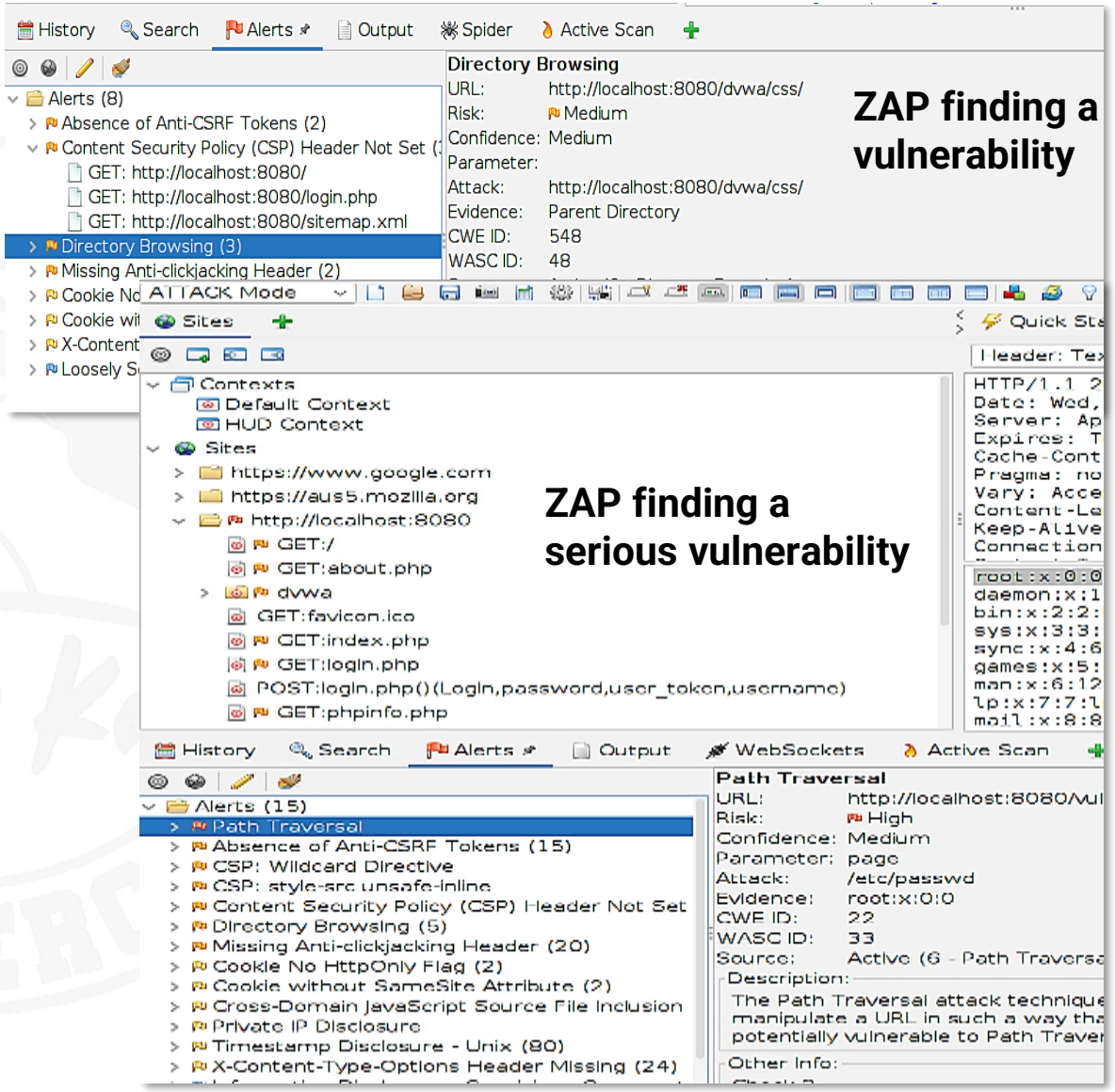


*Icons by Bing Chat AI

Source: https://www.synopsys.com/blogs/software-security/sast-vs-dast-difference/

DAST (Dynamic Application Security Testing) tools

- A very popular DAST tool is OWASP ZAP (in the image)
  - https://www.zaproxy.org/
  - A list of these tools can be found here:
    - https://www.comparitech.com/net-admin/dast-tools/
    - Some of them are more than just a DAST tool
- Manual easy DAST with ZAP
  - Easy and recommended for your **TFG** ☺
  - https://medium.com/@renbe/dast-with-owasp-zap-89fb5e1fa243
- Pipeline-integrated automated DAST
  - https://www.everable.com/blog/automated-dast-in-ci/cd-pipeline-using-owasp-zap-a-comprehensive-guide



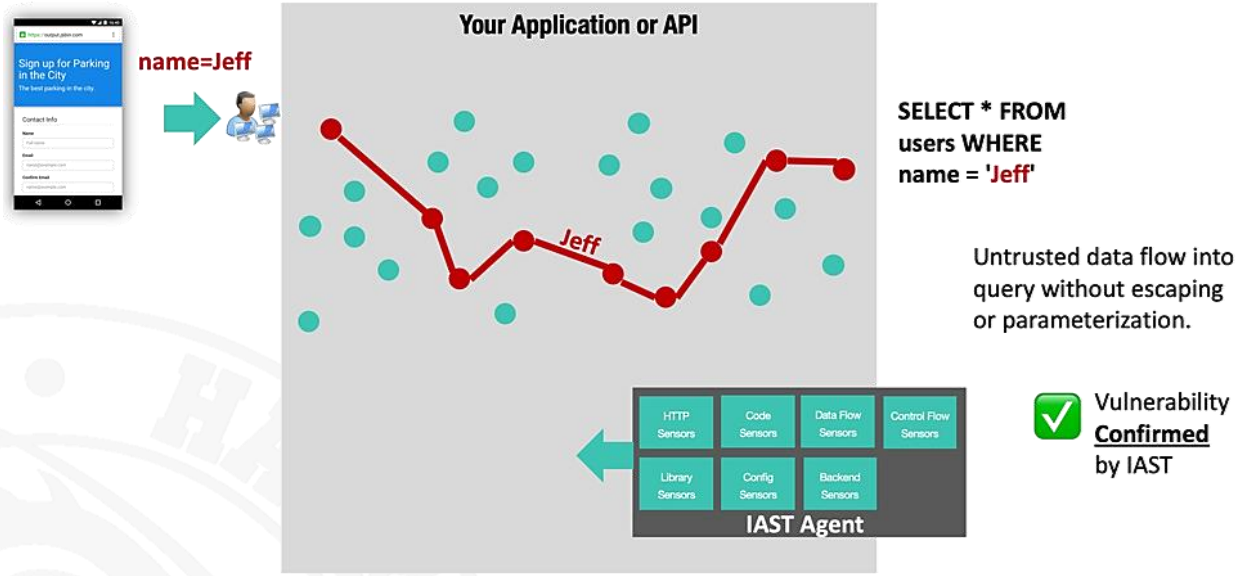
IAST (Interactive Application Security Testing) tools

Computer Security

- **Instrument software to evaluate its performance and detect vulnerabilities**
  - **Agent-like approach**: agents and sensors continuously analyze the application behavior
    - **During automated / manual tests or during execution**
  - Analysis and feedback are done at real time in the IDE, the CI environment, in the quality assurance processes, or in production
  - Sensors have access to complete source code, data and control flow, system configuration data, web components, and back-end connection data
    - Greater coverage and more accurate than SAST and DAST
  - **Examples**: https://owasp.org/www-community/Source_Code_Analysis_Tools
- **Advantages**
  - Potential problems can be detected earlier ("shift left"), minimizing costs and delays
  - Display more complete data from problematic lines of code, making it easier to fix them
- **Disadvantages**
  - They can slow down the application operation (due to code instrumentation)

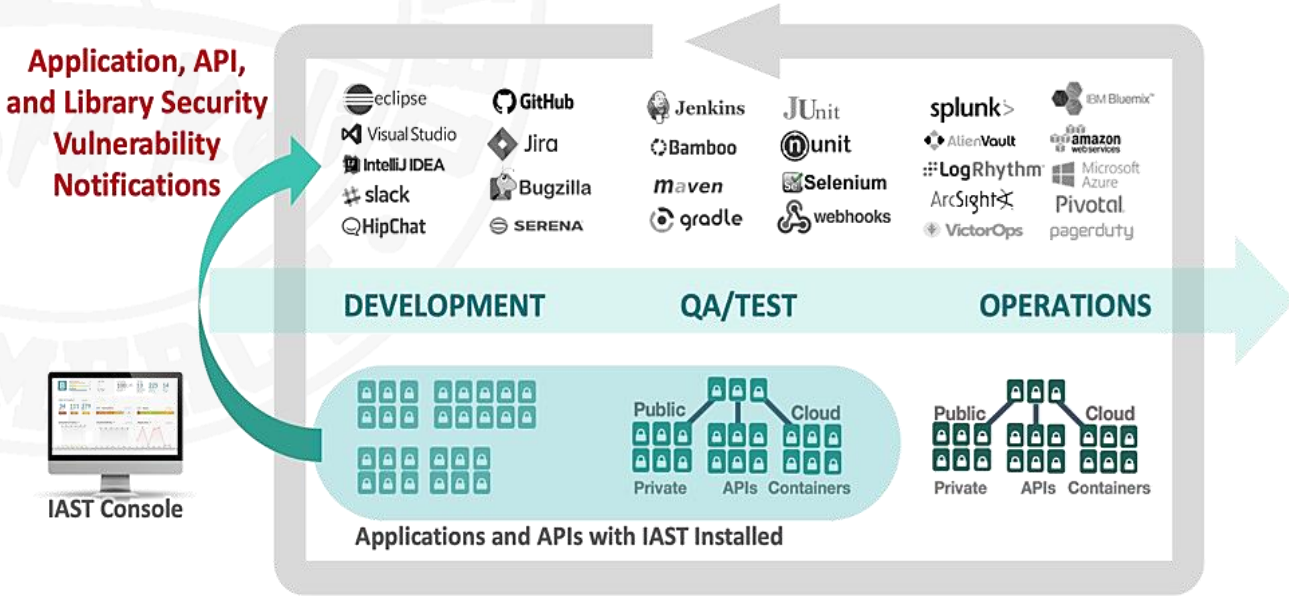
*Icons by Bing Chat AI

IAST TOOLS



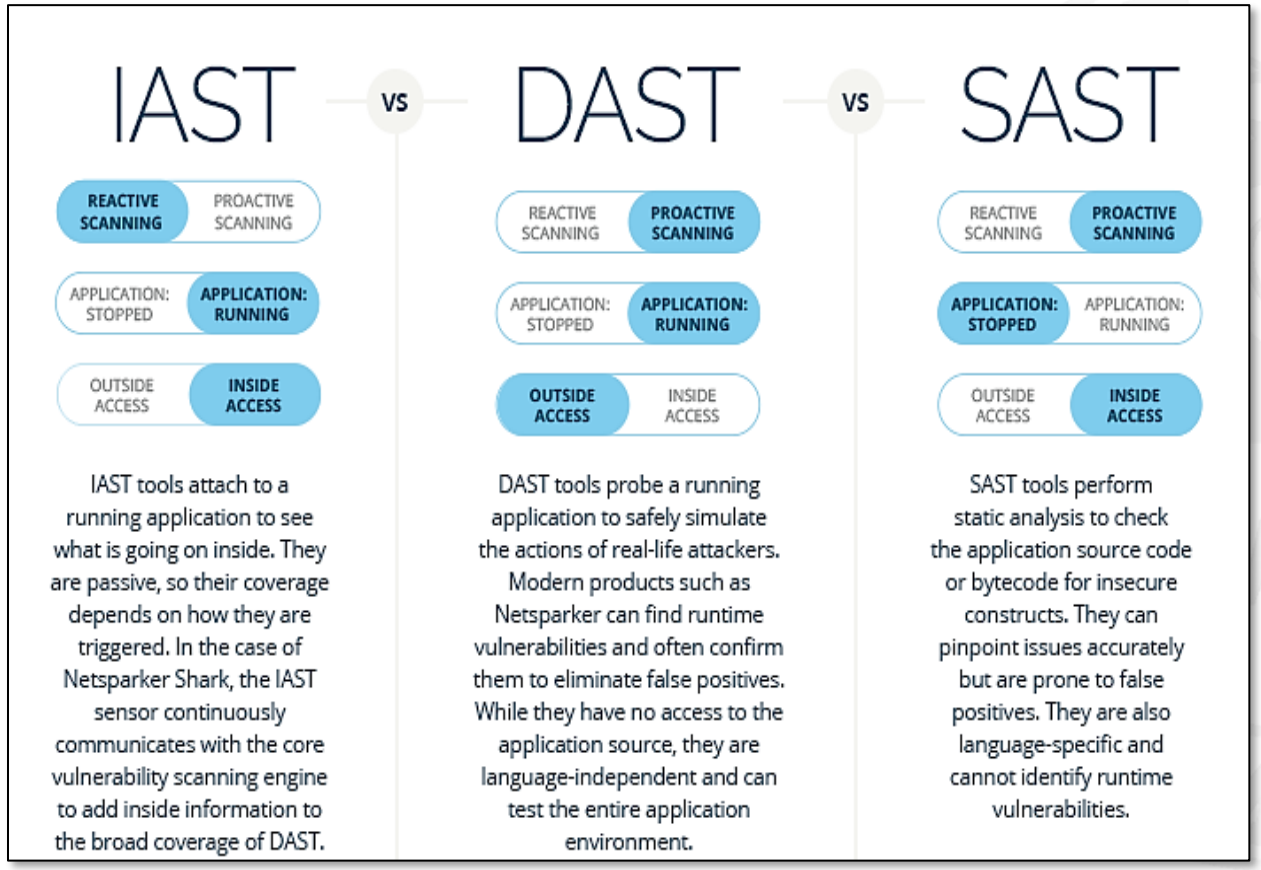
- **With IAST the build, test, and deploy process does not change**
  - But the tool is constantly monitoring the application in the background
- **And notifies the corresponding development pipeline element when dangerous code or libraries are introduced**

An IAST tool is integrated into your app and detects dangerous inputs and actions, informing other elements of the pipeline. Source: https://dzone.com/refcardz/introduction-to-iast



IAST TOOLS

- A list of these tools and more of the previous categories can be found here
  - https://owasp.org/www-community/Source_Code_Analysis_Tools



Source: https://www.scmagazine.com/resource/application-security/devsecops-what-it-is-and-how-to-approach-implementation

Source: https://www.cyberseguridad.com.mx/netsparker-web-application-security/

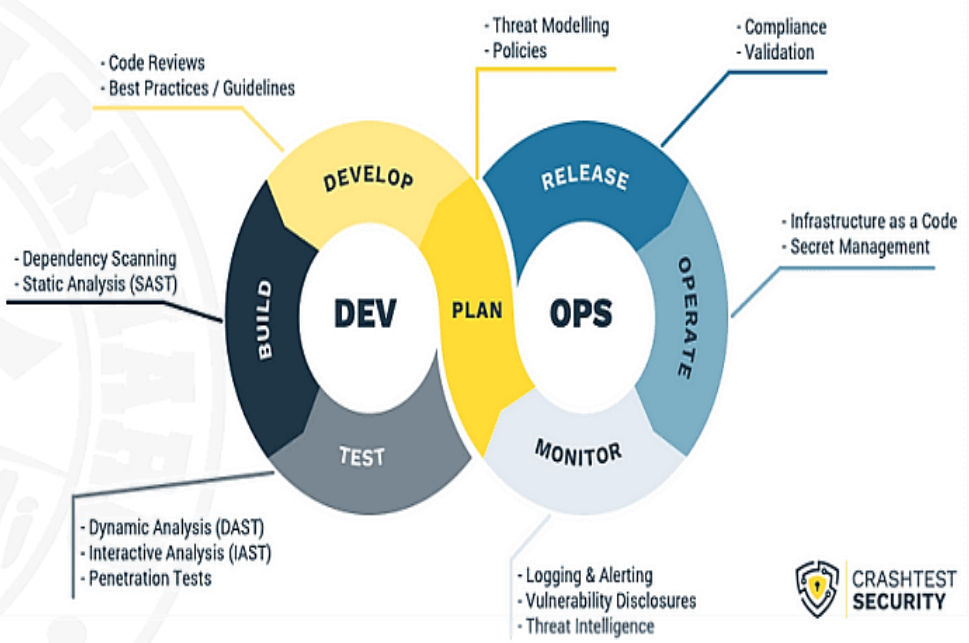
*Icons by Bing Chat AI

Security tools in DevOps

- **The advantages of integrating security tools into the CI/CD pipeline are greater the sooner they are fully integrated in the development/build environment**
  - **"shift left" is also a rule here!**
  - It's vital that **developers feel comfortable using them** as part of their daily work
  - To ensure continuous integration, delivery, and deployment
  - If not, they will generate rejection in DevOps teams, that will not use them properly
- **The tools must also offer**
  - The ability to be invoked from the command line
  - Integration with the metrics dashboard used in the project
  - Tracking the found defects
  - The ability to interrupt compilation and send push notifications via email
- **Languages, frameworks, and APIs are constantly changing**
  - Tools must provide fast compatibility with new versions of the tools

Other SecDevOps tools

- Other tools
  - Code Sight (SCA, SAST): https://community.synopsys.com/s/article/Introduction-to-Code-Sight
  - Black Duck (SCA, SAST): https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html
  - Coverity (SAST): https://scan.coverity.com/
  - TinFoil (DAST): https://www.tinfoilsecurity.com/
  - Seeker (IAST): https://www.synopsys.com/software-integrity/security-testing/interactive-application-security-testing.html
- Our development framework may also include part of these features
  - Ex. Node.js (SDI) automatically audits the security of dependencies
  - https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities



You use these tools in cycles. Source: https://sudip-says-hi.medium.com/an-overview-of-security-testing-tools-in-devops-d955687f2913

Think you've understood it all? Evaluate yourself!

?

- **You can do the following**
  - *Understand that secure development requires the integration of security testing tools into the development chain*
  - *Understand what an SCA tool is for and the type of problems it prevents*
    - *Especially their critical work in identifying and preventing CVEs of third-party software that we are using*
  - *Understand what a SAST tool is for and the type of problems it prevents*
    - *And that they're able to tell us the exact line of code where they detected a problem*
    - *Plus, how to get a SAST (and SCA) up and running very easily on GitHub*
  - *Understand what a DAST tool is for and the type of problems it prevents*
    - *And that in the end they treat the application as a "black box" in execution*
  - *Understand what an IAST tool is for and the type of problems it prevents*

< Go to Index

Secure deployment and maintenance

Things to consider once the application is in production

*Icons by Bing Chat AI

Source: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

What are you going to find in this block?

- **In this block you will learn**
  - What **WAF** tools are
  - What **RASP** tools are
  - What **UEBA** tools are

Security in application Deployment & Maintenance

- **When keeping our application deployed, we must consider**
  - What you've learned in ASR about SSOO **management** and our **Topics 3, 4, and 5**
    - Especially hardening with CIS Benchmarks
  - **Continuous monitoring**, logging, and security alerts (**Topic 5**)
  - To have **incident response procedures**
  - If we create an API, **use an API Gateway**
    - Fully managed service that eases to create, publish, maintain, monitor, and protect an API
    - They act as the “door" for the applications to access the data, business logic, or the functionality of your backend services. E.g.: https://aws.amazon.com/es/api-gateway/
  - Use **active protection** applications once deployed: WAFs, RASP, and UEBAs
  - If we do a **cloud deployment**, we may have to choose cloud-enabled AST tools
    - A very good SCA tool for cloud environments is Snyk (https://snyk.io/)
      - https://snyk.io/blog/what-is-software-composition-analysis-sca-and-does-my-company-need-it/
    - Fortify WebInspect: combines DAST and SAST also in cloud deployment environments
      - https://www.microfocus.com/es-es/products/webinspect-dynamic-analysis-dast/overview

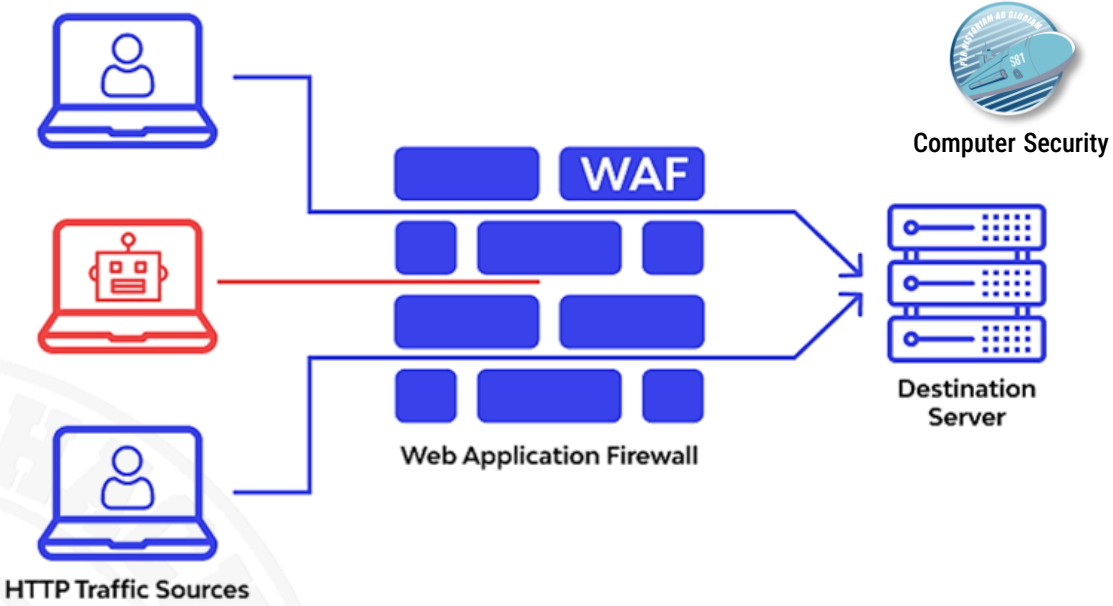
*Icons by Bing Chat AI

Web Application Firewalls (WAFs) (mod_security)

- **Web server modules that automatically protect their hosted applications against a wide range of application attacks: do not deploy without one**
  - Add an additional protection layer to any web application (defense in depth)
  - `mod_security` is probably the best-known free WAF (shown in **Lab 10**)
  - They need rules; the best known are CSR (Core Security Rules): https://modsecurity.org/crs/
  - *Not deploying your own?* Rent it (e. g.: CloudFlare, https://www.cloudflare.com/es-es/lp/ppc/waf-x/)
- **Once active it can perform the following operations automatically**
  - **HTTP Protection**: Detects protocol violations
  - **Real-time IP blocklists**: Using third-party IP reputation servers
  - **Detection of malware, bots, crawlers…**: Google Safe Browsing API and other integrations
  - **Detection of DoS and other common web attacks**: XSS, SQL Injection, CSRF...
  - **Identifying sensitive data and preventing accidental source code leaks** (e. g. credit cards)
  - **Detect error messages giving technical details and mask the identity of the server**
  - *... (many more features)*

*Icons by Bing Chat AI

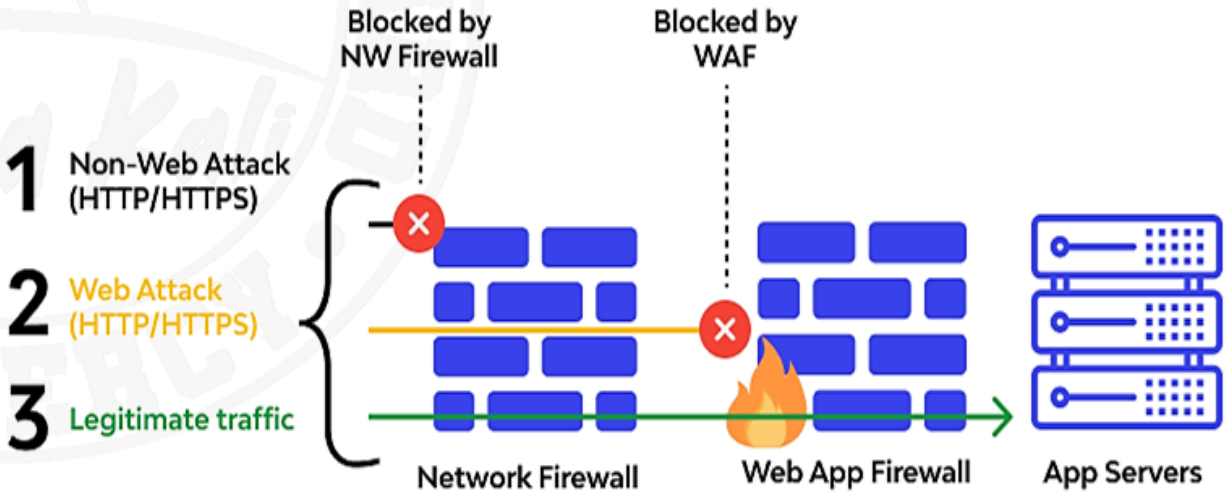
Web Application Firewalls



- **You must differentiate**
  - A firewall (**Topic 5**) analyzes network traffic, but not its contents
    - Source, destination, port...
  - A WAF analyzes traffic content, understands and process it
    - This way it can find malicious requests inside the network traffic
    - Suele encontrarse en un **reverse proxy** que controla el acceso a muchas aplicaciones

Don't get confused, even if they both have the word "firewall" in them, they are not the same. What's more, they are often combined. Source: https://www.wallarm.com/what/waf-meaning

- **The 1st belongs to the network layer, the 2nd to the application layer**
  - **They are not antagonists**, in fact, both should complement each other
  - Having a WAF does not mean you can get rid of your firewall and vice versa!



➢ Exercise **L10B1_REVPROXY**, **L10B2_WAF**, **L10B2_WAFPROT**

RASP (Runtime Application Self-Protection) tools

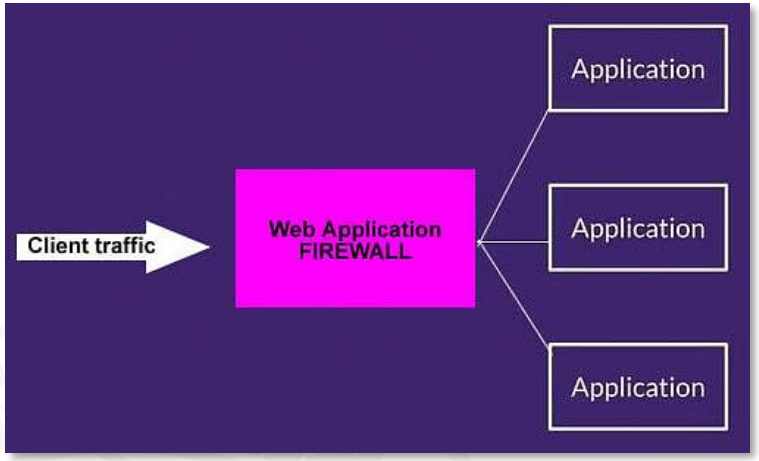
- **Able to inspect application behavior and context**
  - Capture all requests to ensure they are secure, and control the validation of requests within the application
  - May work in **diagnostic** (alert) or **protection** mode (responds stopping the application, ending the session…)
- **RASP and IAST have similar usage, but RASP does not perform full analysis**
  - Runs as part of the application by inspecting your traffic and activity
  - Both report attacks when they occur, but IAST in testing/deployment and RASP in production
- **Advantages**
  - Complement SAST and DAST, being an additional layer of protection in production (defense in depth)
  - Allow you to examine and control unexpected inputs
  - They can respond quickly to an attack by providing a thorough scan and locating vulnerabilities
- **Disadvantages**
  - They run on the application server, and can have a negative impact on its performance
  - **Give a false sense of security**: detected problems need to be fixed and hence application may be stopped
    - Do not a substitute application security testing, as they do not provide complete protection (overconfidence)
- **Examples:** https://www.g2.com/categories/runtime-application-self-protection-rasp

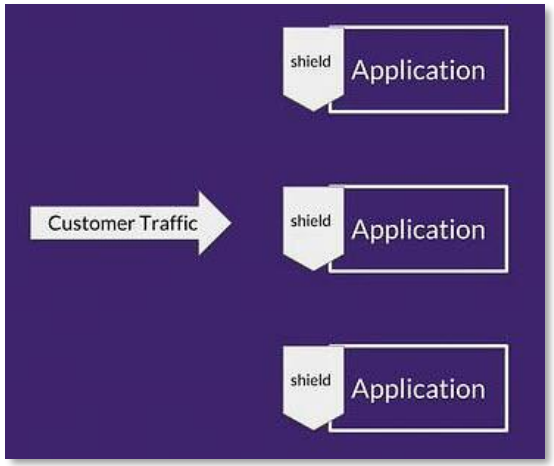
*Icons by Bing Chat AI

RASP TOOLS

Computer Security

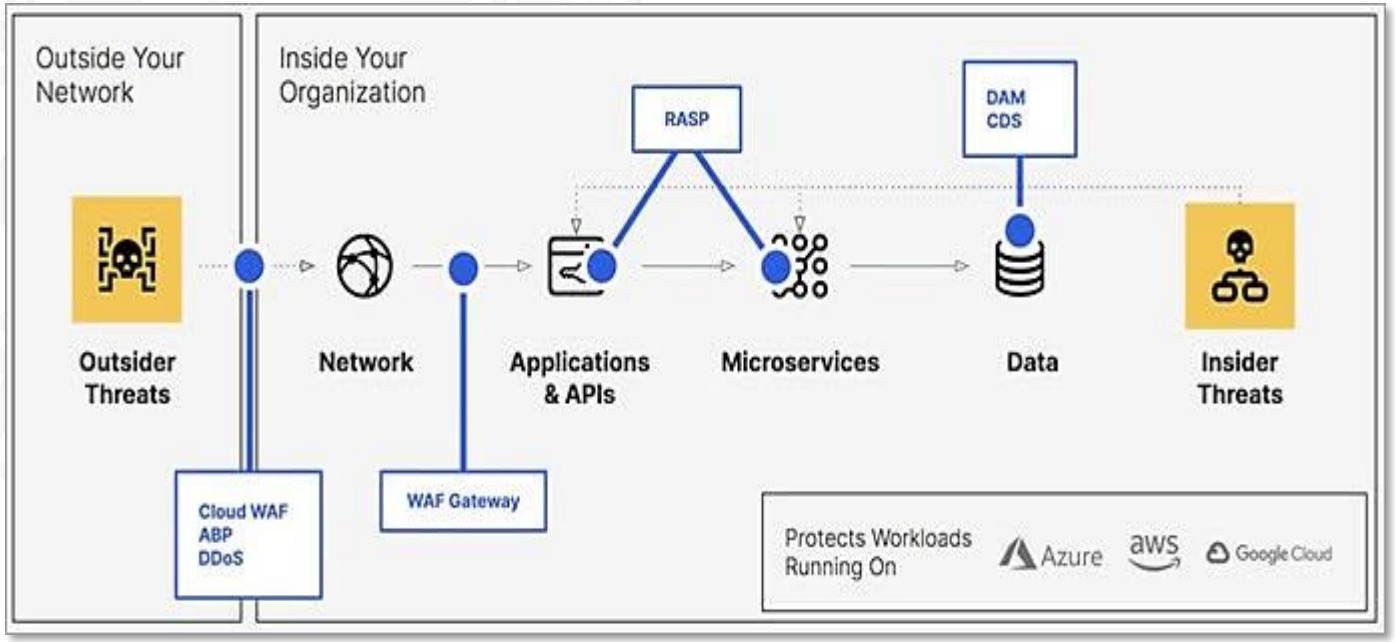
- A RASP tool is an extra layer that is placed in the main application
  - It monitors all incoming traffic
  - Both to the server and to the application API
- If any attack vector is detected, it activated all its embedded protections
  - All of this happens at runtime
- This protects the application from attacks that may come later
  - **It has three working modes**: Off, monitor, and block





Difference Between the Approximation of a WAF and a RASP. Source: https://www.softwaretestinghelp.com/rasp-tutorial/

Where a RASP operates in a running application. Source: https://dzone.com/refcardz/introduction-to-rasp



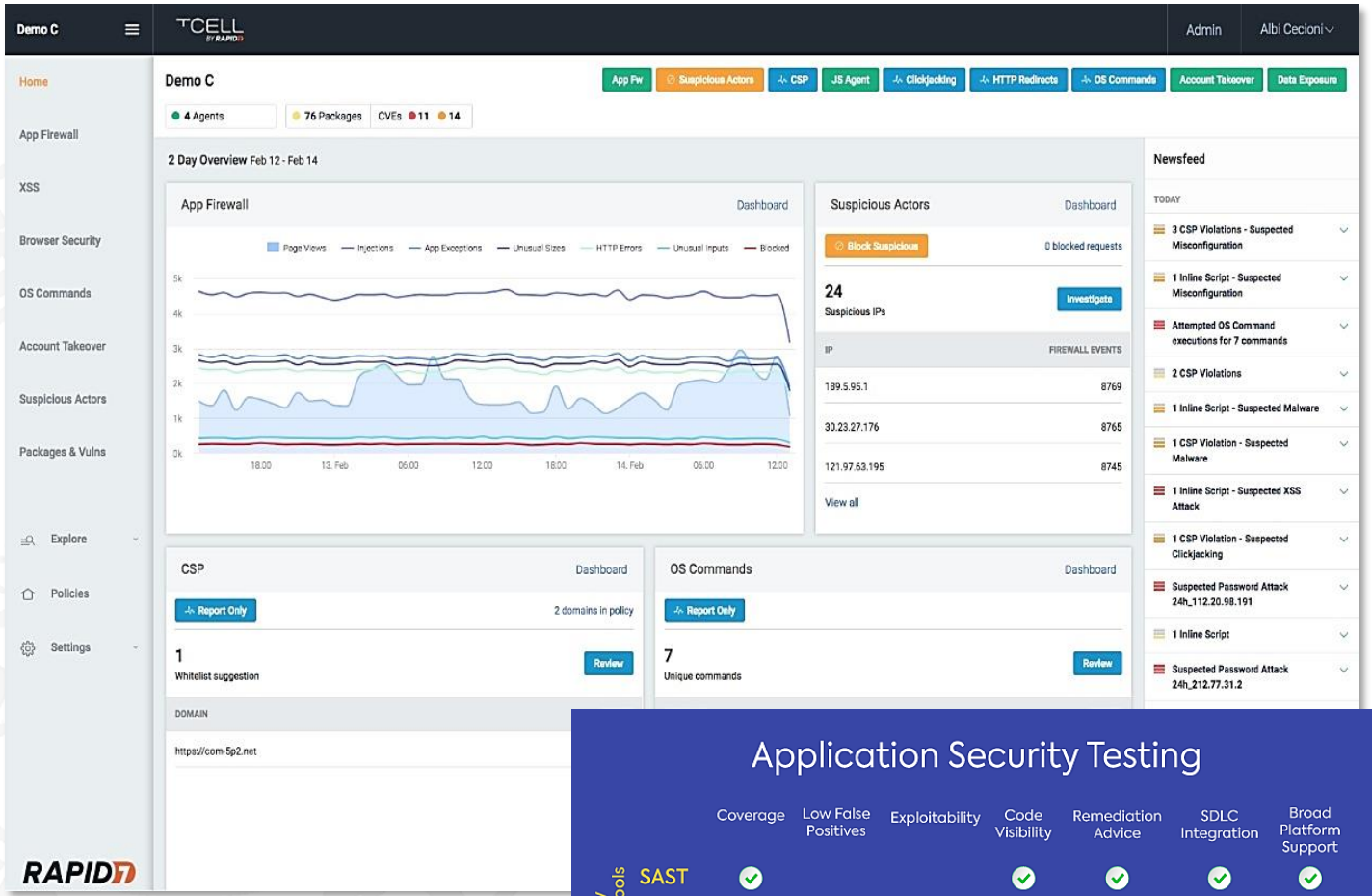
RASP TOOLS

- RASP tools examples are
  - FreeRASP: https://github.com/talsec/Free-RASP-Community
  - More examples: https://www.g2.com/categories/runtime-application-self-protection-rasp

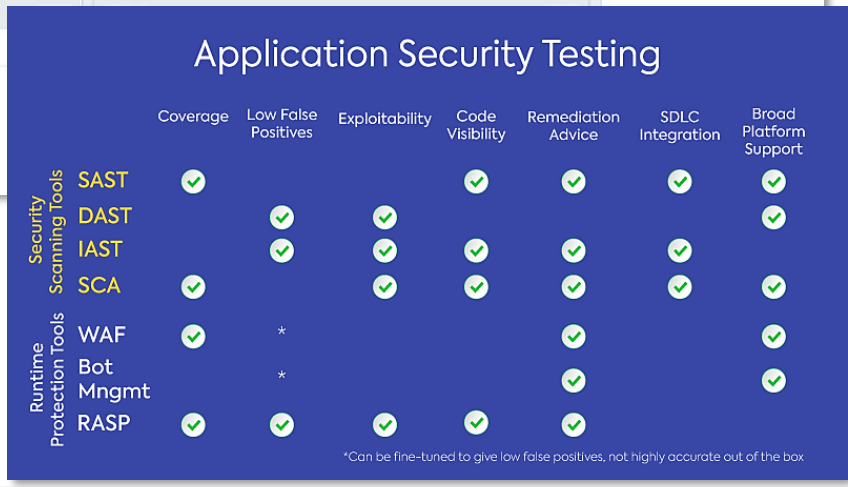
WAF	VS	RASP
WAF stands for Web Application Firewall		RASP stands for Run-Time Application Self-Protection
WAF is a tool that protects web servers from cyber-attacks		RASP is a technology that runs on a server when an application runs
Primarily attack prevention		Detects both attacks and vulnerability
Mitigates DDoS Attacks and Block Brute Force Attacks		Injects security at runtime
False Positives may occur		False Positives are eliminated
WAF protects web apps by filtering, monitoring, and blocking any malicious HTTP/S traffic traveling toward the web application by adhering to a set of policies		RASP is designed to detect attacks on an application in real time. When an application begins to run, RASP can protect it from malicious input or behavior by analyzing both the app's behavior and the context of that behavior
Alerts are based on Predictions		Alerts are based on Application behaviour
WAF creates set of policies against all type of attacks like Brute Force Attacks, a SQL injection attacks & XSS Attacks and many others		RASP tools offers multiple detection techniques like sandboxing, semantic analysis, input tracing and behavioral analysis to go along with signatures
Flexible, hybrid deployment		Minimal overhead of deployment
Multi-layer defense		Applies defense inside the application

*Icons by Bing Chat AI

Source: https://www.rapid7.com/products/tcell/



Source: https://www.facebook.com/Appsealing/posts/waf-vs-raspa-web-application-firewall-waf-is-the-solution-for-protecting-web-app/497757835109742



UEBA (User and Entity Behavior Analytics)

- **IA-powered anomaly detection tools**
  - **Analyze the behavior** of the users, servers, accounts, laptops, applications, etc.
    - Everything connected to an organization's network!
  - They are used to detect threats, external failures and intruders
    - Inside jobs, rogue users…they are an increasing security risk
  - They **learn** what they do to set what is considered **"normal" behavior**
    - *Where do they connect from? What servers, files, applications do usually access? From which devices?...*
  - Model “typical" behaviors to distinguish them from “anomalous ones”
    - When something unusual happens, the UEBA will detect it and alert those responsible
- **Examples of these tools are IBM Security Qradar, Idaptive Next-Gen Access, ActivTrak, InsightIDR, Teramind or Exabeam Security Management Platform**
  - More complete list: https://www.g2.com/categories/user-and-entity-behavior-analytics-ueba/free
- **Kassandra** is a research project that follows this philosophy developed by Alba Cotarelo Tuñón, a graduate of this school
  - https://github.com/Egida-Kassandra/kassandra

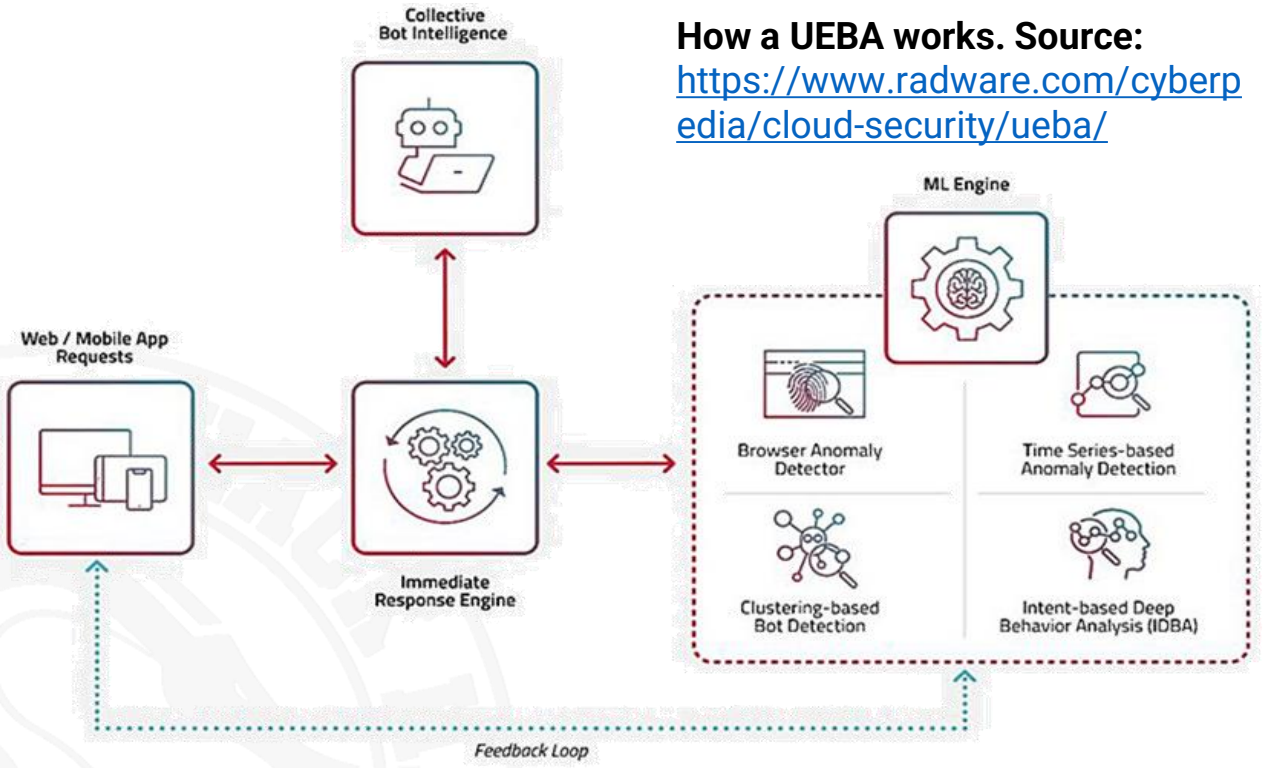
*Icons by Bing Chat AI

UEBA

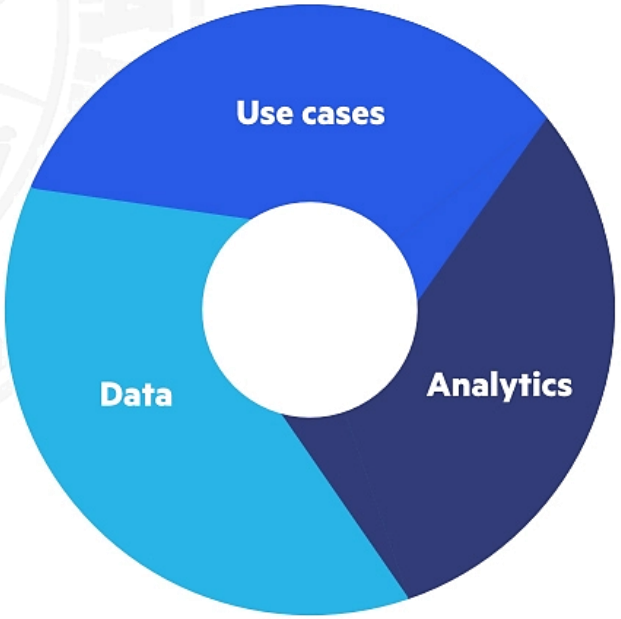
Computer Security

- A UEBA is a **Machine Learning** (ML) system that learns from user actions
- It analyzes several data sources to identify known use cases
  - Malicious or compromised users, etc.
- Data include external and internal application sources
  - Ex.: Application logs
  - It is one of the most modern and interesting applications of AI for cybersecurity

How a UEBA works. Source: https://www.radware.com/cyberpedia/cloud-security/ueba/



Data used by a UEBA. Source: https://www.imperva.com/learn/data-security/ueba-user-and-entity-behavior-analytics/



Use cases
- Malicious insider
- Compromised user
- APT and zero-day
- Known threats

Analytics
- Supervised machine learning
- Unsupervised machine learning
- Statistical modeling
- Rule-based system

Future:
- Generative adversarial networks
- Ensemble networks
- Deep learning

Data
- Events and logs
- Network flows and packets
- Business context
- HR and user context
- External threat intelligence

Think you've understood it all? Evaluate yourself!

?

- **You can do the following**
  - *Understanding the concept of API Gateway*
  - *Understand what a WAF is and how it protects in the context of web application development*
  - *Understanding what a RASP is and how it protects an application*
    - *And that it is an add-on that is located within the application itself*
  - *Understand what a UEBA is and how it protects an application*
    - *And that it really is a way to detect deviations in the behavior of users and entities*

References

- To read more about **DevSecOps**
  - https://blog.pentesteracademy.com/devsecops-learning-path-integrating-security-with-devops-1cc03670552f?gi=b60a95ce4fcb
- To learn an alternative approach to **DevSecOps**, it is advisable to read the **OWASP Proactive Controls**
  - https://owasp.org/www-project-proactive-controls/
- To work with a **Jenkins** pipeline (used in **Software Architecture**, **ASW**) with the different testing tools we saw
  - Installing **Jenkins** in a **Docker** container (suitable for testing)
    - https://www.jenkins.io/solutions/docker/
  - **Fortify** plugin for **Jenkins** (**SCA**)
    - https://plugins.jenkins.io/fortify/
  - **SonarQube** plugin for **Jenkins** (**SAST**) (also for other similar products)
    - https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-jenkins/
  - **DAST** with **OWASP ZAP** and **Jenkins**
    - https://digitalvarys.com/devsecops-dynamic-analysis-dast-with-owasp-zap-and-jenkins/

TOPIC 6
INTRODUCTION TO APPLICATION SECURITY

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo