TEMA 6
INTRODUCCIÓN A LA SEGURIDAD DE APLICACIONES

Creando un código seguro

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

Fuente: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Logo: @creative_vanesa (*Instagram*)

ÍNDICE

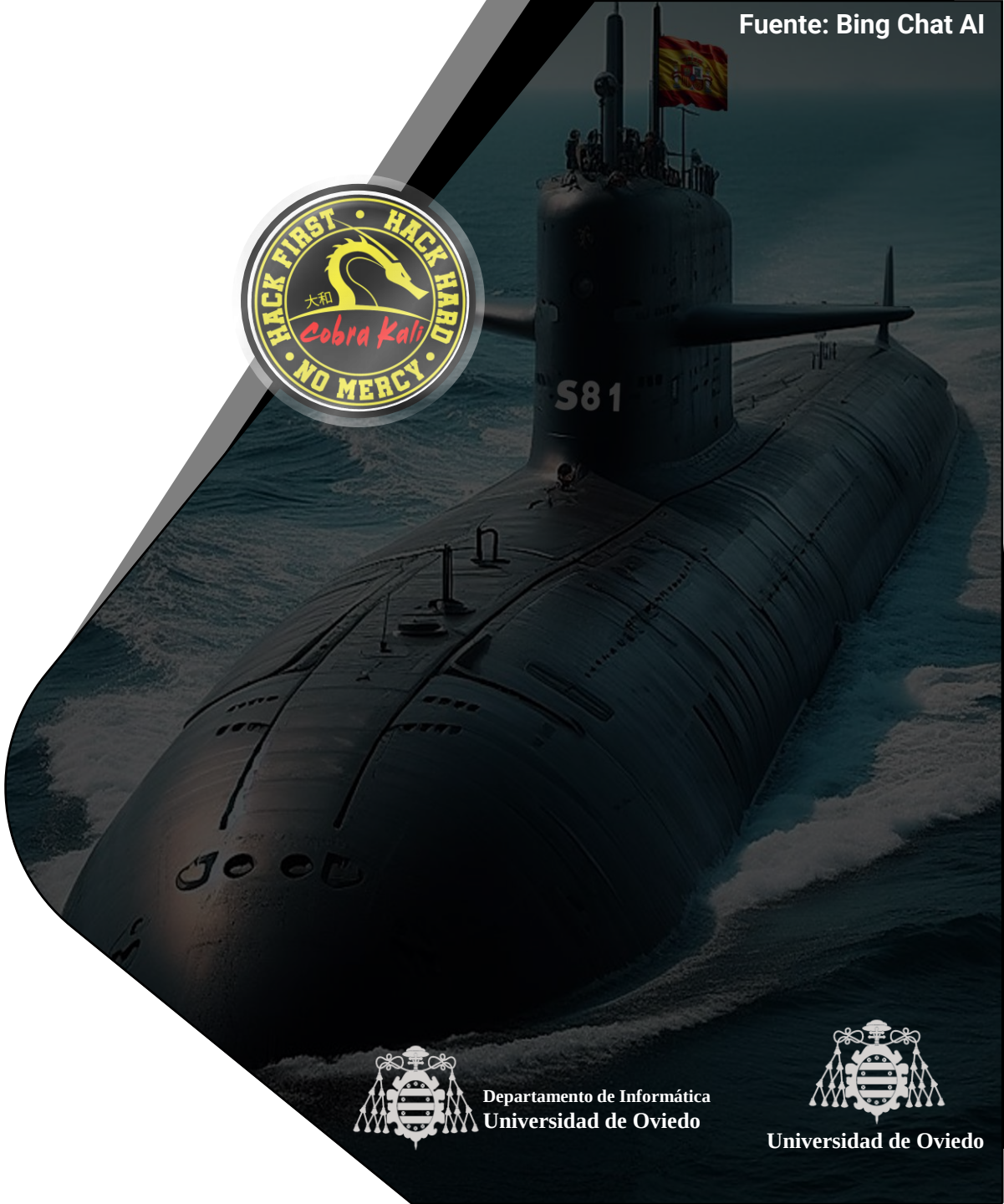
- Fundamentos de Seguridad de Aplicaciones
- Seguridad a Nivel de Requisitos
  - Manejo de datos
  - Autenticación
  - Otros
- Diseño Seguro
  - Threat modeling
- Creación de Código Seguro
  - Autenticación y autorización
  - Log y errores
  - Usabilidad
- Application Security Testing (AST)
  - Herramientas estáticas
  - Herramientas dinámicas
- Despliegue y Mantenimiento Seguros

< Ir al Índice

Fundamentos de Seguridad de Aplicaciones

Conceptos básicos antes de empezar

*Iconos por Bing Chat AI



Fuente: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Principios de diseño de software seguro (para TODO el software)

- **Mínimo privilegio**
  - Un software o usuario debe tener sólo los privilegios necesarios para realizar su labor
- **Valores por defecto a prueba de fallos**
  - A menos que el software reciba acceso explícito a un objeto, se debe denegar el acceso por defecto
- **Mediación completa**
  - Se debe validar cada acceso a objetos para asegurarse de que está permitido
- **Separación**
  - El software no debe conceder acceso a un recurso, o tomar acciones relevantes de seguridad, basándose en una única condición
- **Confianza mínima**
  - El software debe comprobar todas las entradas y los resultados de todas las acciones relevantes para la seguridad
- **Economía de mecanismos**
  - Las funciones de seguridad deben ser lo más simples posible

Principios de diseño de software seguro (para TODO el software)

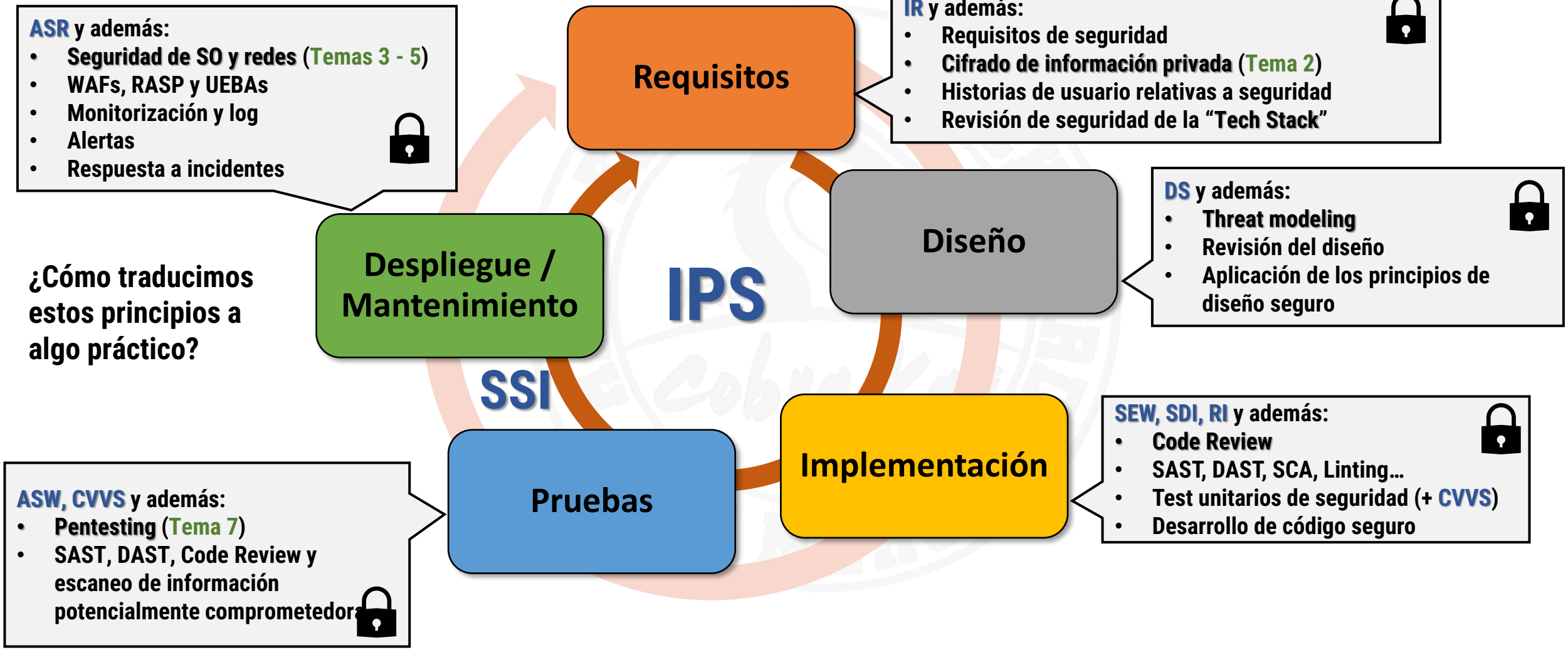
- **Minimizar mecanismos comunes**
  - Reducir el uso de recursos compartidos al mínimo
- **Menor asombro/sorpresa**
  - Las funciones y mecanismos de seguridad del software deben ser diseñados de tal manera que su manejo sea tan lógico y sencillo como sea posible
- **Diseño abierto**
  - La seguridad del software y sus funciones no debe depender del secreto de su diseño o implementación
- **Capas (principio de separación)**
  - Organizar el software en capas para que los módulos de una capa sólo puedan interactuar con los módulos de las capas inmediatamente inferior y superior
  - Permite probar por capas, usando técnicas top-down o bottom-up, y reduce los puntos de acceso

Principios de diseño de software seguro (para TODO el software)

- **Abstracción**
  - Ocultar el interior de cada capa, dejando sólo sus interfaces públicas
  - Esto permite cambiar la implementación de una capa sin afectar a los componentes de otras capas
- **Modularidad**
  - Diseña e implementa el **software** como una colección de componentes cooperativos (**módulos**)
  - Cada interfaz de módulo es una abstracción
- **Vinculación completa**
  - Enlazar el diseño e implementación de la seguridad de un software a sus requisitos de seguridad
- **Diseño por iteraciones**
  - Planificar el diseño para que pueda cambiar si se necesita
  - Minimiza los efectos de los cambios de diseño en la seguridad

Del SDLC al S-SDLC (tercer curso, integrado y "securizado")

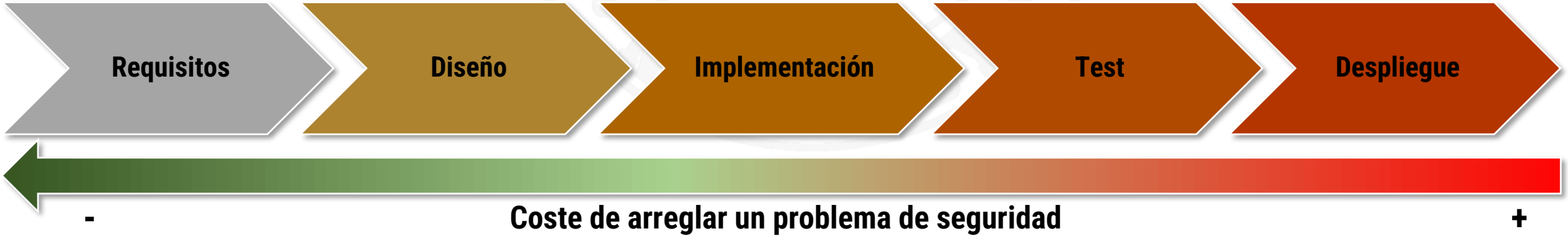
Seguridad de Sistemas Informáticos



¿Cómo traducimos estos principios a algo práctico?

La filosofía "pushing left"

- Cuanto más tarde se descubra un problema de seguridad en el S-SDLC, más caro será solucionarlo (en tiempo y recursos)
- Por tanto, debemos estar seguros de que todas las características de seguridad estén establecidas lo más a la izquierda posible de la cadena del SDLC
  - A esto se le suele denominar **filosofía "pushing left"**
- Por ello, este tema también explica más los elementos iniciales de la cadena
- El resto de los elementos están también cubiertos por otras asignaturas (SEW, SDI, CVVS, IPS, ASR), o por otras partes de la asignatura



< Ir al Índice

Seguridad a nivel de Requisitos

La seguridad se planifica desde el minuto cero

*Iconos por Bing Chat AI

Fuente: Bing Chat AI

Manejo de datos >

Autenticación >

Otros >

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ TE VAS A ENCONTRAR EN ESTE BLOQUE?

En la primera sección de este bloque aprenderás

- La importancia tan grande que tiene **tratar cualquier entrada que recibe la aplicación como insegura**, y por qué es imprescindible su validación
  - Y también los distintos **puntos donde puedes tener entrada de datos**
- La importancia (y diferencia) entre **codificación y "escapado**" de caracteres
- Lo que debes hacer para **crear cookies seguras** en tu aplicación web

En la segunda, aprenderás

- Las claves para una **autenticación segura**: Requisitos de gestión de contraseñas y MFA
- La diferencia entre una **cuenta de servicio y de usuario** y porqué las primeras son mejores para usarlas con aplicaciones
- Como plantear el requisito de **recuperar una password** olvidada de forma segura

Y en la tercera aprenderás

- Lo enormemente importante que es **minimizar el uso de componentes de terceros**
- **Otros requisitos importantes de seguridad**: Cabeceras HTTP, backups, subida de ficheros, logs, formas de autenticación y autorización, mínimo privilegio, etc.

PREGUNTAS DE SEGURIDAD EN LOS REQUISITOS

- **En 4º curso hay una asignatura llamada Ingeniería de Requisitos (IR)**
  - Debes usar sus contenidos para modelar los requisitos de la aplicación, **y añadir los de seguridad**
  - No importa la metodología de desarrollo que uses (IPS, ASW), lenguaje (TPP) o framework (SDI, SEW)
- **Una persona o equipo debe desarrollar los requisitos de seguridad de la aplicación**
- **Deben hacer preguntas con las que se elaborarán**
  - *¿Contiene o maneja información confidencial, sensible o que identifica a las personas (Personal Identifiable Information, PII)?*
  - *¿Dónde y cómo se guardan los datos?*
  - *¿Está disponible públicamente (Internet)?*
    - *¿O no?* (solo para una Intranet)
  - *¿Va la aplicación a realizar tareas sensibles o esenciales?*
    - Transferencias de dinero, manejo de datos médicos…
  - *¿Hace algo considerado potencialmente peligroso?*
    - Ej.: Permitir a los usuarios subir ficheros, manejar datos de pago, tener varios roles de usuarios...
  - … (¡No te cortes!, *"Arrasa con lo que veas, y generoso no seas"*)

Requisitos de Cifrado

- **Ya vimos los tipos de cifrado en el Tema 2, y hay que recordar…**
  - Las claves y la información sensible **NUNCA** **se guardan en texto plano**
  - Se usan técnicas de **hashing** para claves (**Key Derivation Functions**)
    - Y cualquier información cuyo valor original no necesitemos
    - Identidad, autenticación, integridad de los datos…
  - Para el resto de los casos, usaremos algoritmos de **cifrado/descifrado fuertes**
    - Siempre cifrar la información confidencial **en tránsito** (usuario <-> **frontend**, **frontend** <-> **backend**, **backend** <-> SGBD, API <-> API, etc.)
    - Y **cuando se guarde** (en el SGBD) (también llamado "**cifrado en reposo**")
  - Esto **garantizaría la confidencialidad e integridad** de la información (no la disponibilidad)
  - Cifrar los datos en RAM es posible, pero solo con datos extremadamente sensibles
    - Es mejor "limpiar" la memoria una vez los usemos

Requisitos de manejo de datos

Distintas técnicas para tratar tus datos de forma segura

*Iconos por Bing Chat AI

Fuente: Bing Chat AI

REQUISITOS DE MANEJO DE DATOS

- **Nunca confíes en la entrada del usuario**
  - **Toda entrada** que llegue al sistema puede ser falsificada o manipulada maliciosamente
  - Esto puede causar que una aplicación falle de formas no previstas y se vuelva inestable
    - En esta situación un atacante puede usar nuestra aplicación para desencadenar un ataque muy peligroso
  - Por tanto, **TODA ENTRADA debe ser validada** y manejada adecuadamente
    - No permitas que la aplicación caiga en estados inestables
  - Siempre hay que **basarse en listas de entradas admitidas** si es posible
    - Además, las normas para validar cada tipo de entrada son conocidas y están clasificadas
    - https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
  - **La validación SÓLO en JavaScript es inútil desde el punto de vista de seguridad**: se puede desactivar
    - Los web proxies pueden cambiar el contenido antes de transmitirlo al servidor, aunque haya sido validado
    - Ej.: Burp, https://portswigger.net/burp/documentation/desktop/tools/proxy, ver **Seminario 3**
- **¿Pero si se sabe cómo y dónde hacerlo, cuál es el problema?**
  - Que muchas veces no sabemos la **ENORME CANTIDAD** de fuentes de datos que manejamos
  - Veamos una lista no exhaustiva de cosas manipulables

Posibles puntos de entrada de datos

- **Cualquier entrada del usuario (la obvia), cabeceras HTTP, campos "hidden"….**
- **Información que se reciba de**
  - **SGBDs** (sean tuyos o no)
  - **APIs** (incluso si la has hecho tú mismo)
  - Código incluido en la aplicación y compilado con ella (**dependencias de terceros**)
    - Bibliotecas, frameworks, código de GitHub...
  - **De otra aplicación**
    - Especialmente si va a integrarse con la desarrollada o acepta entradas de ella
    - Incluye aplicaciones serverless, scripts, frameworks externos, información para el log (¡díselo a Log4j!) …
- **Valores en la URL (parámetros), en las cookies, en los ficheros de configuración…**
  - Especialmente si las cookies no tienen los flags "secure" y "HTTPS only"
  - Incluye valores que se usen de sistemas de almacenamiento online (S3, Firebase…)
- **Datos o comandos que vengan de aplicaciones en nube**
- **Imágenes que recibas de otros sitios**

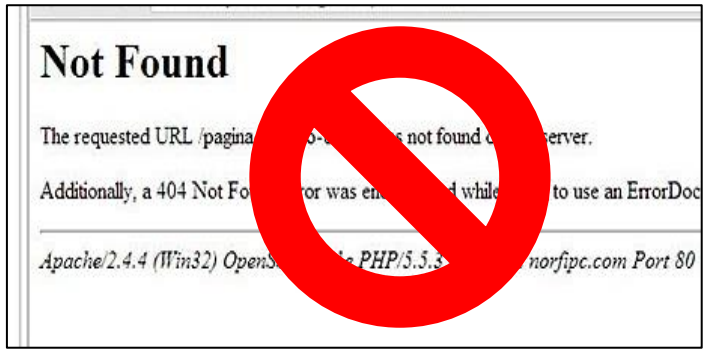
Requisitos de manejo de datos: Manejo

- Una entrada **NO es fiable hasta pasar tus procedimientos de validación**
  - **NUNCA antes de eso**
  - La validación son todos los test que aseguren que **la entrada es apropiada y tiene el tipo adecuado**
  - Cualquier discrepancia debe resultar en que **la aplicación rechace la información recibida**
- Ejemplos
  - Si esperas una **fecha de nacimiento**...
    - Comprueba que lo es, y que es realista en el contexto de la aplicación
  - Si esperas un **nombre**...
    - Comprueba que tiene las características esperables en tu caso (O'Reilly, caracteres japoneses, etc.)
  - Si esperas un **email**...
    - Hay expresiones regulares que pueden validarlos sin error
  - Si recibes el **resultado de una búsqueda**...
    - Valida que tiene el tipo y formato esperado (XML, JSON, valores…)
  - Si llamas a una API y te devuelve **un tipo conocido**...
    - DEBES comprobar que lo que recibes lo tiene, lo que incluye que no sea excesivamente largo
    - En general **es aconsejable fijar un tamaño máximo esperable para cualquier dato** manejado, si es posible

Requisitos de manejo de datos: Manejo

- Si tu aplicación debe aceptar **URLs** y manejarlas...
  - **Debes entender que estás en una situación potencialmente peligrosa (open redirect)**
  - Intenta **evitar** ese escenario en la medida de lo posible
  - Valida que realmente es una URL (expresiones regulares)
  - Asegúrate de que llega por un canal cifrado
- Si tu aplicación (o parte) debe hacerse en un lenguaje que no es memory-safe (como C/C++)
  - Esto se hace cuando se prioriza el rendimiento, pero introduce nuevas amenazas
  - Usa **Rust**: Rendimiento similar y es memory-safe: elimina el problema de los buffer overflows
- … (piensa y adáptalo a tu caso)

- **Si después de todo esto detectas un error en la entrada, siempre rechaza los datos e informa del error**
  - **NO intentes corregir los datos**
  - Nunca des demasiada información al usuario
    - Que podría usar en tu contra
  - Trata de **no reproducir el dato** que produjo el error en el mensaje
    - Si debes hacerlo, entonces recurre a su **codificación y "escapado"**



Requisitos de manejo de datos

- **Privacidad de datos**
  - Debe existir una política especificando los datos que se recogen en las **cookies**
  - Adaptada a la legislación actual (ver ASLEPI)
- **Clasificación de datos**
  - Los datos de la aplicación deben ser **clasificados y etiquetados** de acuerdo a lo privados que son
    - Así cualquier usuario que esté trabajando en la aplicación sabrá con qué tipo de datos está trabajando
    - Y, por tanto, qué cosas debe proteger especialmente para que se cumplan los requisitos de privacidad
  - El tipo de clasificación depende del **país/legislación aplicable**
    - Ej.: Classified, Secret, Top Secret (ver ASLEPI)
    - Dependerá de si conocer un dato puede dañar a una persona, una empresa o incluso un país entero
  - Si no hay reglas aplicables, se pueden seguir las del **NIST**
    - "Guide for Mapping Types of Information and Information Systems to Security Cathegories":
      - https://www.nist.gov/publications/guide-mapping-types-information-and-information-systems-security-categories-2-vols

Requisitos: Aplicación de codificación y "escapado"

- "**Escapar**" un carácter o un valor implica hacerle perder "el poder" que tiene en un determinado contexto
  - Normalmente, la capacidad de que **sea interpretado como código**
  - Habitualmente se hace añadiendo un `\` al carácter
- **Codificar** un valor es cambiarlo por otro en un formato diferente
  - Normalmente, el formato destino es **inocuo** en el contexto en el que se va a usar
    - ¡Y nunca se interpreta como código!
    - Ej.: en HTML "`>`" se convierte en "`>`"
  - El tipo de codificación depende del contexto en el que se vaya a usar
    - URL, base64, HTML encoding…
  - La codificación **ES** reversible y **NO ES** cifrado
- **Todo esto se hace para evitar ataques de inyección de código**
  - Ej.: El muy conocido XSS (del que hablaremos luego)
- **Los requisitos deben determinar las situaciones en las que aplicar estas técnicas**

Requisitos de seguridad en cookies

- **Recordad que las cookies son el mecanismo que se integró en el protocolo HTTP para poder tener estado (SEW)**
  - **No es lo mismo una Cookie que la "Local Storage" del navegador**
  - Tienen reglas de seguridad **muy distintas**
    - https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html
- **Si se van a usar cookies, los requisitos a seguir son**
  - Todo lo escrito en una cookie **debe ser validado** antes de usarse, y rechazarse si no es correcto
  - El ID de sesión debe pasarse SIEMPRE **en una session cookie**
    - Nunca en una persistent cookie (tracking cookie)
    - Una session cookie se destruye al final de una sesión
  - Las cookies deben enviarse **siempre bajo conexiones cifradas** (Secure Flag, `Set-Cookie: Secure`)
  - No debe poder accederse a las cookies desde JavaScript (HTTPOnly Flag, `Set-Cookie: HttpOnly`)
    - Esto previene algunos vectores de ataque XSS
  - Las cookies persistentes **deben caducar** (atributo `Expires`)

Requisitos de seguridad en cookies

- **Dominio (Domain)**
  - Por defecto **las cookies de un dominio solo pueden ser accedidas por webs de ese dominio**
  - La configuración es segura por defecto (secure by default)
  - Si deseamos que otros dominios lean nuestras cookies, podemos hacerlo con `Set-Cookie: Domain: ads.vinuesa.com`)
    - Si en lugar de eso ponemos `vinuesa.com`, cualquier página de ese dominio podría leer nuestra cookie
  - Se recomienda dejar la opción por defecto o restringir los dominios al máximo
- **Path**
  - **Muchas aplicaciones realmente son un conjunto de aplicaciones distintas que forman una mucho más grande**
  - Todas bajo una URL base
  - En ese escenario cada cookie debería ser leída solo por su "trozo" de aplicación
  - Cada "trozo" se especifica con su ruta dentro del a URL (`Set-Cookie: path=/contabilidad`)

Requisitos de seguridad en cookies

- **Same-Site**
  - **Es una medida contra ataques CSRF** (vistos después)
  - El valor `Strict` (recomendado) hace que las **cookies** solo puedan provenir del mismo sitio que las usa, no de un enlace o de otra página
    - `Set-Cookie: same-site=Strict`
  - El valor `Lax` bloquea las cookies de otras páginas que vengan vía `POST`, pero permite enlaces
- **Prefijos de Cookie**
  - Nueva medida de defensa en profundidad
  - Que asegura que solo determinados subdominios puedan acceder a una cookie
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie

¿CREES QUE LO HAS ENTENDIDO TODO? ¡AUTOEVALÚATE!

?

- **Eres capaz de hacer lo siguiente**
  - *Comprender que para pensar en requisitos de seguridad hay que hacer las preguntas adecuadas*
  - *Distinguir entre cifrado en tránsito y cifrado en reposo*
  - ***Sobre requisitos de manejo de datos***
    - *Tener muy claro que ninguna entrada de usuario es confiable, en ninguna circunstancia*
    - *Comprender que la entrada de usuario no son solo los componentes **textfield** sino, además, muchas otras cosas relacionadas con el protocolo HTTP, **software** de terceros con el que interaccionemos...*
    - *Entender que el tipo de datos esperado es algo clave a la hora de validarlos*
    - *Tener claro que los datos se deben etiquetar según su privacidad y la legislación aplicable*
    - *Distinguir entre codificar y escapar un dato, cuando y para qué se usan*
    - *Comprender la función de todas las herramientas de seguridad de las cookies: **Secure**, **HttpOnly**, **Expires**, **Domain**, **Path** y **Same-Site***

Requisitos de autenticación

Lidiando con passwords e identidades de usuario

Fuente: Bing Chat AI

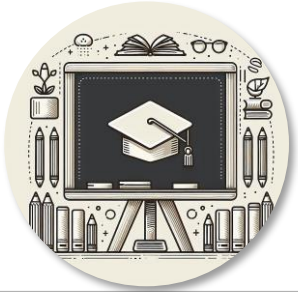
*Iconos por Bing Chat AI

REQUISITOS DE CLAVES DE USUARIO Y SECRETOS DE LA APLICACIÓN

- **Los usuarios deberían usar un password manager (Ej.: KeePass, https://keepass.info/)**
  - Esto evita recordar muchas passwords y se asegura de que todas sean complejas
  - Tienen **características adicionales**: consultar si las claves están en una brecha de datos conocida, si el servicio puede usar MFA (visto luego)...
  - De esta forma **se optimiza la gestión de claves y se obliga a que sean fuertes**
    - En caso de robo de datos se minimiza el impacto (se evita la reutilización de claves y el credential stuffing)

    Password manager + password únicas por cada web + MFA = defensa en profundidad personal

- **Las secret stores son similares, pero para software, no para personas**
  - Es un archivo ("vault") que encripta secretos de una aplicación
    - Claves, hashes, certificados, tokens de servicios en nube, cualquier cosa que requiera protección…
  - **Acceso programático**: desde el código de aplicación o desde el build de un pipeline CI/CD (ASW)
    - Es importante mirar cómo implementa estas stores el framework de desarrollo que vaya a utilizarse
    - A nivel de requisitos, debe determinarse **QUÉ** debe ir en estas stores
  - **Es lo que DEBE hacerse en lugar del "hardcoding" secretos de la aplicación**
    - Este problema también se da con las aplicaciones hechas **con lenguajes de bajo nivel**
    - Lo veremos en el **Seminario 4** (reversing)



Requisitos para passwords de usuario

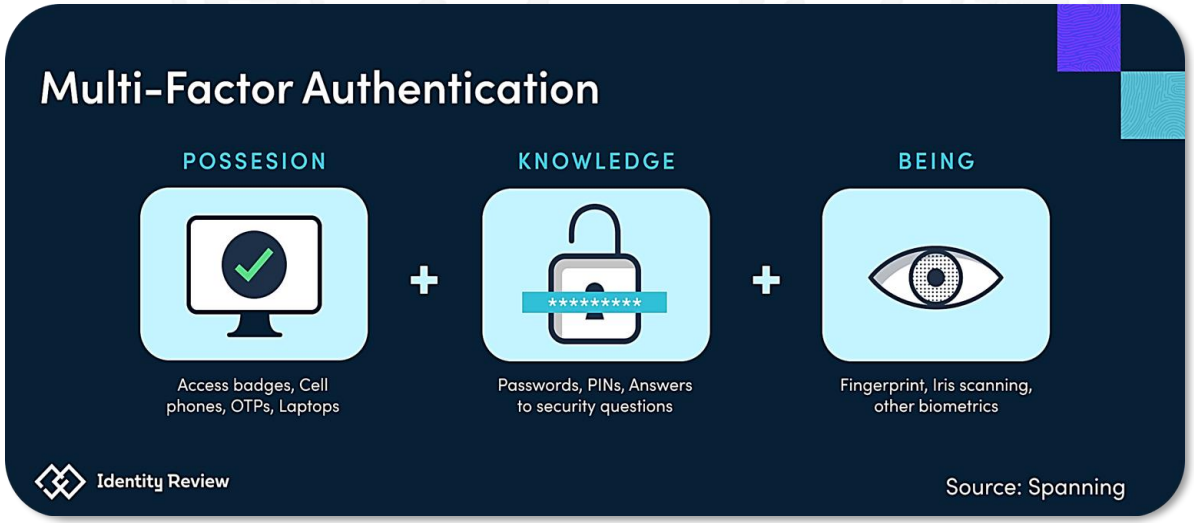
- **Intenta NO guardar passwords en tu aplicación**
  - Usa un servicio de autenticación de terceros si es posible (Google, Outlook…)
- **Si no es posible, guarda claves como se indicó en el Tema 2**
  - Usando un algoritmo KDF, largas, complejas, **salt** (28+ caracteres), **pepper**, **strong hashing**...
  - Puede usarse la API de `haveibeenpwned` para comprobar que una nueva clave no ha sido ya comprometida
    - https://haveibeenpwned.com/API/v2
  - **Nunca informar al usuario** de si lo que ha introducido mal es el nombre de usuario o la clave
    - Cuando intente hacer login sin éxito, usa un mensaje genérico: "Datos incorrectos"
  - Nunca permitir la **reutilización de claves** ni usar claves muy parecidas ("Pass1", "Pass2", etc.)
  - **Usar MFA para todas las cuentas importantes**
    - Esto es aplicable tanto en nuestras cuentas personales como en las de las aplicaciones
    - Usar preferiblemente una aplicación o dispositivo más que un SMS (previene el SIM-Swapping)
  - **No hagas un cambio de password periódico forzoso**, solo cuando se descubra una brecha

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

- **Proceso de requerir a los usuarios verificar su identidad de dos o más formas únicas e independientes antes de acceder a un sistema**
- **Extiende el esquema clásico de nombre/clave con un paso de autenticación extra**
  - Típicamente requiriendo al usuario introducir un token o one-time password (OTP) de un solo uso generado dinámicamente y mostrado por un método al que solo puede acceder el usuario
  - Este token es mucho más difícil de obtener que una clave para el atacante
- **Los MFA son cada vez más comunes dada la debilidad de las passwords**
  - La complejidad de las password, si es obligatoria, se contrarresta usando la misma clave en distintos servicios (con distintos niveles de seguridad), el phising, la ingeniería social….
- **Con MFA, incluso con la clave comprometida no es posible acceder a una cuenta**
  - Es necesario saber el método de MFA y el OTP o token
  - Un fallo repetido al introducir un MFA también generará una alerta de clave comprometida
- **Soportado por la FIDO Alliance**: https://fidoalliance.org/

Tipos de MFA. Principios básicos

- **Los esquemas MFA se basan en que los usuarios den 2 de 3 “algos”**
  - Algo que **Saben**, como una clave o PIN de una cuenta
  - Algo que **Tienen**, como un dispositivo físico (móvil) o una aplicación que genera OTPs
  - Algo que **Son**, una característica biológica única como una huella, la voz o la retina
- **Hay muchas formas de implementar MFAs**
  - Todas tienen sus desventajas, pero todas incrementan la seguridad de las cuentas
  - Todos los métodos fuerzan al usuario a introducir una 2º clave tras la 1º para poder acceder
  - Esta segunda password se genera dinámicamente y cambia constantemente



Tipos de MFA

- **Token SMS**: Envía un token único por SMS a un teléfono registrado y validado
  - **Fallos**: pérdida de señal, clonado o robo de la SIM (SIM Swapping, suplantación de usuarios)
    - ¡O interceptación del SMS! (no se cifran)
  - PlayStation Network y Amazon pueden usar este método (pero se recomienda elegir otro)
- **Token por Email**: Idéntico al anterior, pero el código se recibe vía email
  - **Fallos:** Es aún más fácil suplantar al usuario si se roban las credenciales del correo
  - Una implementación popular es Steam Guard (pero se recomienda elegir otro)
- **Token Hardware**: Los usuarios tienen un dispositivo (como un USB) que contiene un token de acceso generado dinámicamente y que debe conectarse al PC
  - Google Titan es una aproximación similar, pero usando una infraestructura PKI
    - https://cloud.google.com/titan-security-key/?hl=es
  - Su popularidad está en aumento, al haber dispositivos más baratos cumpliendo los protocolos FIDO
    - https://www.amazon.es/YUBIKEY4-YubiKey-4/dp/B018Y1Q71M
  - Método típico para prevenir la exfiltración de datos privados de una empresa de PCs robados

Tipos de MFA

- **Token Software**: Se instala una aplicación que genera tokens dinámicos
  - **El método recomendado por tener mejor relación coste / beneficio**
  - **Problemas**: Las aplicaciones pueden ser caras de implementar o mantener, y...
    - Requieren la instalación de un programa de terceros (que puede estar restringido en ciertos entornos)
    - Si se compromete, entonces nuestro sistema de autenticación también lo estará
  - **Ejemplos**: Google Authenticator, Microsoft Authenticator, Blizzard Authenticator o Steam Guard
- **Teléfono**: se llama al usuario para decirle el de acceso
  - **Problemas** : Llamada interceptada/redirigida, mensajes de voz intervenidos, necesita cobertura
  - Se usa para validar a los empleados que hacen su 1ª conexión o re-conectan tras caída de red, o al conectarse a ciertas VPN
- **Verificación Biométrica**: el usuario es el token
  - Guardar datos biométricos (Ej.: Huellas) plantea **muchas dudas sobre privacidad**
    - Se pueden suplantar usuarios con datos robados
  - Requiere dispositivos especiales para leerlos (cámaras, scanners...) que suponen un **coste adicional**
  - Hollywood usa esto para impresionar a los espectadores ☺, ¡pero también se usa en la vida real!

The Consumer Authentication Strength Maturity Model (CASMM) V6

Seguridad de Sistemas Informáticos

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

Cuentas de servicio y no de usuario para la aplicación como requisito

- **Las cuentas de servicio son cuentas de usuario que están destinadas a ser usadas por ordenadores**
  - **Nunca usuarios "humanos"**
  - Esto significa que una cuenta de servicio nunca podrá tener acceso a shell o a una GUI
- **Si una aplicación necesita acceder a una base de datos...**
  - Debe hacerlo a través de una cuenta de servicio
  - **NUNCA** la de un usuario real con capacidad de hacer login
  - Si un empleado se va, no se compromete la aplicación por no poder acceder a través de su cuenta
- **Cada aplicación debe tener su propia cuenta de servicio para este tipo de accesos**
  - Esto facilita su monitorización respecto a las otras
  - Además, así se distingue perfectamente lo que hace cada aplicación frente a cada usuario
  - Otra ventaja es que así pueden restringirse los permisos al máximo
  - ¡**Algo que es SIEMPRE aconsejable**!

REQUISITOS: FUNCIONALIDAD DE PASSWORD OLVIDADA

Seguridad de Sistemas Informáticos

Nunca revelar si el nombre de usuario es inválido o no

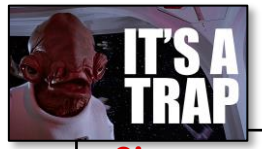
Página específica para claves olvidadas. El usuario pide reiniciar su clave

Verificar la identidad del usuario

Usar un MFA si es posible

Enviar al usuario un enlace al email con el que se haya registrado, NUNCA pedir un email

El sistema registra la petición de cambio de clave



Si esto ocurre 10+ veces en menos de 24 horas, bloquear la cuenta e IP y crear una alerta

El usuario usa el enlace para cambiar la clave

El sistema registra el cambio de clave

Si se cambia la clave 10 o más veces en 24 horas, bloquear la cuenta o IP y generar una alerta

El sistema manda un email de confirmación de que se ha cambiado la clave

Fin del proceso

Nunca incluir un enlace para que no parezca phising

¿Crees que lo has entendido todo? ¡Autoevalúate!

?

- **Eres capaz de hacer lo siguiente**
  - *Distinguir entre un gestor de contraseñas y una secret store, para qué se usan y quien lo usa*
  - *Entender qué política es la mejor a la hora de gestionar contraseñas y qué debes hacer si tienes que hacerlo tú mismo*
  - *Comprender qué es un MFA, sus tipos (con sus ventajas y desventajas), y cuál es el mejor desde el punto de vista de coste de implementación / seguridad*
  - *Entender qué es una cuenta de servicio, cómo y para qué debemos usarla en nuestras aplicaciones*
  - *Saber cómo plantear los requisitos de un servicio de funcionalidad de contraseña olvidada*

Otros requisitos de seguridad

Cosas adicionales que debemos considerar

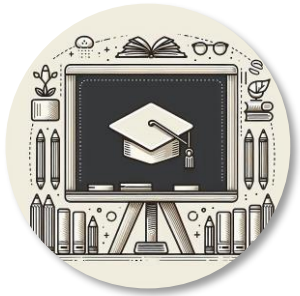
*Iconos por Bing Chat AI

Fuente: Bing Chat AI

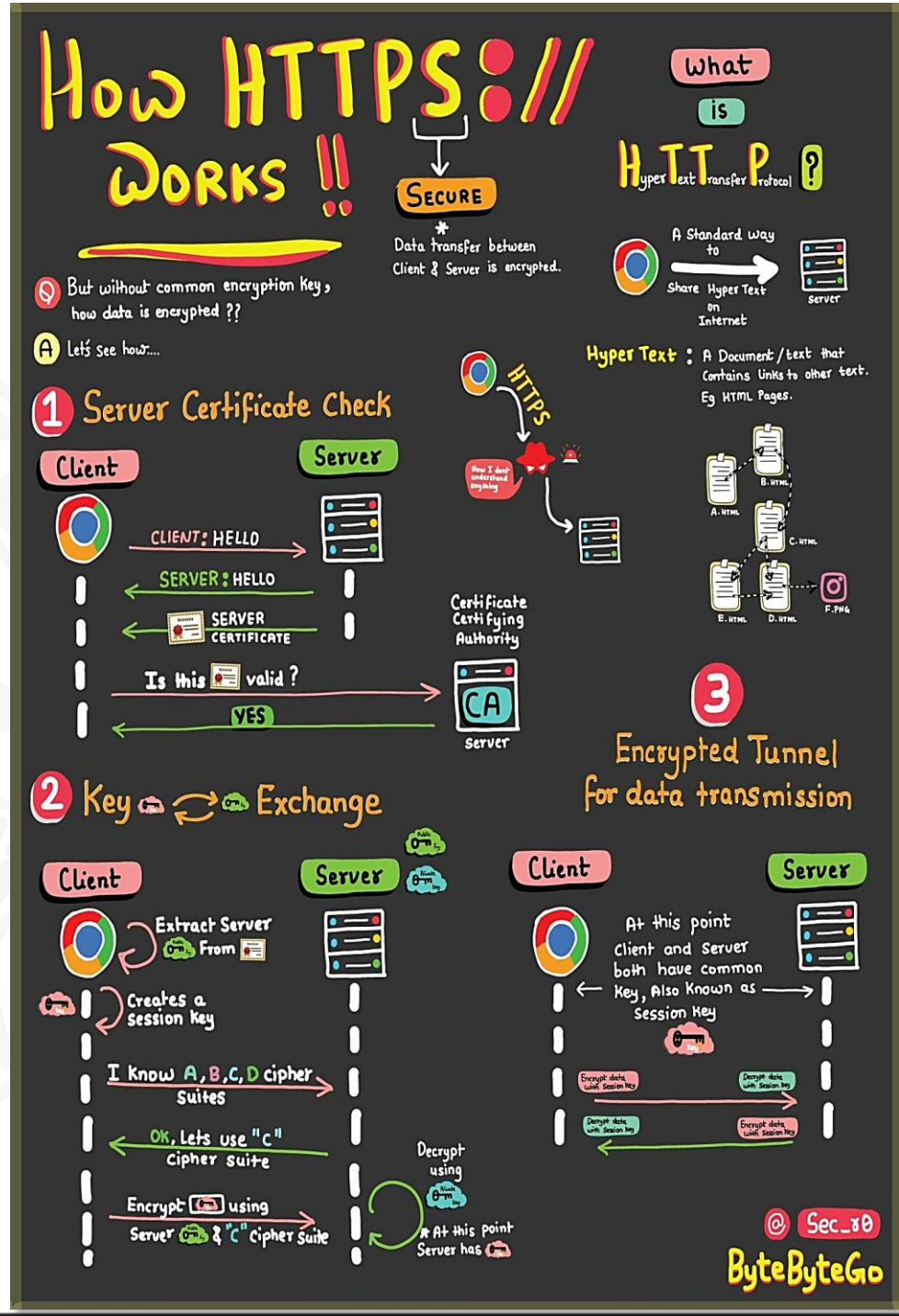
Requisitos: Minimizar componentes de terceros

- **TODA** línea de código incluida de una librería, **framework**, plugin, componente de 3ºs, etc. **ES UN RIESGO** que incorporas a tu aplicación
  - **Son ataques a tu "cadena de suministro" de software (supply chain attacks)**
    - **¿Esto no es demasiado extremo? NO, es la realidad** (¡cada vez son más frecuentes!)
  - Toda vulnerabilidad en cualquier código de 3ºs incluido en tu aplicación es ahora **TU vulnerabilidad**
  - Verificar periódicamente **TODO** el código de terceros que se use es un requisito más de la aplicación
    - Para comprobar que no tiene vulnerabilidades conocidas (CVEs, ver **Tema 1**)
- Por suerte, hay herramientas que **automatizan** el proceso (**SCA**)
  - **Úsalas en tus repositorios de código** de aplicaciones antiguas regularmente
  - **Intégralas en tu pipeline CI/CD** (ASW) y detén el despliegue si se encuentra alguna vulnerabilidad
  - Otra medida es **MINIMIZAR el uso de componentes de terceros**
    - Trata de **usar solamente los estrictamente necesarios**
    - Esto se llama **minimizar tu superficie de ataque**, que es otro principio básico a cumplir desde el diseño
    - ¡Así minimizas el riesgo de ser afectado por una **vulnerabilidad desconocida (zero-day)**!

Otros requisitos de seguridad importantes



- **Usar cabeceras de seguridad HTTP (Seminario 6)**
- **Usar HTTPS SIEMPRE**
  - No sabemos cómo se conecta el usuario a la aplicación
    - Cable, Wi-Fi, datos, su propio router, uno comunitario...
  - Usar HTTPS en toda la aplicación tiene **ventajas**
    - Evita network sniffing e **interceptación de datos**
      - Con aplicaciones especializadas en eso (Ej.: Wireshark)
    - El tráfico sin cifrar **puede modificarse "en ruta"**
      - Inyectando malware, elementos maliciosos, o desinformación
- **La configuración HTTPS debe ser robusta**
  - Siguiendo los últimos estándares, recomendaciones y prácticas
  - Que están todas disponibles gratuitamente y públicas aquí: https://ssl-config.mozilla.org/



Otros requisitos de seguridad importantes

- **Eliminar cualquier comentario del front-end de la página**
- **Backups**
  - Todos los datos que la página maneje deben ir a un backup periódico
  - Contando además con otro semanal en una localización geográfica distinta
  - **El proceso de restauración de dichos backups debe probarse y practicarse también**
- **Características de seguridad del framework**
  - Hay que usar **TODAS** las características de seguridad de tu framework **SIEMPRE**
  - En lugar de desarrollar una propia (codificación, manejo de sesiones...)
  - Siempre estarán más probadas que cualquier cosa que podamos desarrollar nosotros
- **A nivel de requisitos, debemos lidiar con la posible "deuda técnica" de la empresa**
  - Hard coding, software obsoleto, inseguro, sin mantenimiento o sin parchear...
  - Backdoors introducidas "por agilidad" ☹
  - Hace a las organizaciones lentas de reacción ante incidentes, y hoy en día no se puede permitir
  - Como ingenieros de software debemos mostrar a la empresa los riesgos que tienen estas prácticas

Otros requisitos de seguridad importantes

Subida de ficheros

- **Intenta evitar esta funcionalidad** y, si no es posible, usa componente de 3ºs probado y verificado
- Es **probablemente la funcionalidad más peligrosa** (especialmente si se abre a cualquier usuario)
- Hay que seguir una serie de recomendaciones si esta funcionalidad es necesaria
  - https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html

Errores y log

- Los errores no deben tener **NUNCA** trazas de pila o errores de BBDD
  - Deben capturarse de manera que no se dé información técnica al usuario
- Si la organización tiene un SIEM (ver **Tema 5**)
  - Deben escribirse los logs en el lugar y con el formato admitidos por el mismo para poder analizarlos
- **Jamás debe registrarse información sensible** en los logs
  - Claves, nºs de tarjeta, etc., en general, nada que identifique a una persona
- Si se detecta alguna actividad sospechosa, la aplicación no solo debe registrarla
  - Sino también **mandar una alerta** (al email del equipo responsable de seguridad, no a una persona…)
  - **Ejemplos**: acceso a una característica no permitida, intentos excesivos de login, fallos de la aplicación...

Otros requisitos de seguridad importantes

Autorización y Autenticación

- Si la aplicación tiene diferentes tipos de acceso para distintos usuarios (Role-Based Access Control, RBAC), estos deben ser definidos durante la fase de requisitos (AuthZ)
  - Es decir, establecer los roles de los usuarios claramente desde el principio
- También debe definirse el método de autenticación (AuthN) de los usuarios
  - Es decir, cómo se van a tratar sus identidades

Usa parametrized queries

- Ningún usuario o aplicación debe poder interactuar directamente con la base de datos
- El uso de este tipo de queries evita esto
  - Las entradas del usuario son los parámetros de sentencias pre-construidas
  - Esto previene ataques de SQL Injection (vistos más adelante)
- Por tanto, debe ser un requisito establecer cómo se construyen este tipo de consultas
  - Usando las funcionalidades del framework usado
  - **Y obligar a usarlas en todo momento**

Otros requisitos de seguridad importantes

Parámetros de URL

- No poner parámetros importantes o datos privados en la URL
- Por ejemplo, nombres de usuario, IDs…
- Debe restringirse solo a datos irrelevantes (por ejemplo, el idioma de la página)
- Además de dar facilidades a usuarios maliciosos, ¡quedan registrados en el log!

Principio del mínimo privilegio

- **Forzarlo en todas las partes de la aplicación**, sobre todo en accesos a BBDD o APIs. Ej.:
- Cuando interactuamos con un SGBD, debemos crear **dos cuentas de usuario**
  - Una con **permisos sólo de lectura** para ejecutar sentencias `SELECT`
  - Otra con permisos para implementar un CRUD de las entidades
    - NUNCA database owner (DBO)
  - La primera **debe usarse siempre en todas las operaciones posibles**
- Se debe crear una cuenta de servicio **para cada API** que la aplicación use (aislamiento)
  - Y cada una debe tener solamente los permisos estrictamente necesarios para su función
- Dar permisos de lectura/escritura al **repositorio de código solo a los desarrolladores**

¿Crees que lo has entendido todo? ¡Autoevalúate!

?

- **Eres capaz de hacer lo siguiente**
  - *Entender que todo componente de terceros incluido en una aplicación es un riesgo y que hay que minimizar su uso*
  - *Saber la importancia de configurar el protocolo HTTP de la forma más robusta posible (cabeceras, cifrado)*
  - *Comprender que hay **otra serie de requisitos de seguridad importantes** que debemos considerar, como*
    - *Hacer backups de todo el código y datos de la aplicación*
    - *Usar todas las funcionalidades de seguridad que tengamos a nuestro alcance (librerías, **frameworks**)*
    - *Que la situación no va a ser la ideal debido al concepto de deuda técnica, y que tenemos que considerar su existencia*
    - *Que debemos aspirar al menos a usar roles para gestionar usuarios como requisito*
    - *La importancia de las **parametrized queries** a la hora de hacer peticiones a los SGBD*
    - *Que los parámetros de la URL son un punto de entrada para ataques frecuente*
    - *Y que jamás debemos otorgar a un usuario y/o parte de la aplicación más privilegios que los estrictamente necesarios*

< Ir al Índice

Threat modeling >

Fuente: Bing Chat AI

Diseño Seguro

Aspectos de seguridad en el diseño de aplicaciones

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

*Iconos por Bing Chat AI

¿QUÉ TE VAS A ENCONTRAR EN ESTE BLOQUE?

En este bloque aprenderás

- Los conceptos principales que modelan un **diseño seguro**
  - Viéndolos como una forma de plasmar los requisitos en elementos concretos de actuación
- Qué es y cómo se hace el **threat modeling**
  - Incluyendo los **actores** involucrados y la importancia de lo que se obtiene al final del proceso

Fallo de diseño vs bugs y conceptos de diseño seguro

- **Todos los requisitos de seguridad deben ser puestos en práctica en el diseño**
  - Para así evitar en lo posible los fallos de diseño (no confundir con los bugs de seguridad)
    - Un fallo de diseño permite a los usuarios hacer cosas que no deberían
    - Un bug es un fallo en el código de la implementación de una característica
  - **Recuerda**: cuanto 1º se descubra un fallo de diseño relativo a seguridad, más barato es arreglarlo
  - Esto se consigue con el **Threat modeling**
- **Protección de datos sensibles**
  - **Implementar todo lo dicho en requisitos sobre protección de datos**
  - Si hay políticas de empresa de cómo tratar datos sensibles, se deben cumplir
    - Si no existen, puede ser una buena idea crearlas en este punto
    - Ej.: Decidir cuánto tiempo se guardan datos sensibles y borrarlos automáticamente una vez transcurrido
  - Si se van a poner datos en la nube, ser especialmente cuidadoso con cuáles
    - Ojo con la legislación vigente, ASLEPI
  - Intentar contratar servicios profesionales por si hay filtraciones de nuestros datos sensibles online

*Iconos por Bing Chat AI

Conceptos de diseño seguro

Never Trust, Always Verify/Zero trust/Assume breach

- Solo usar datos previamente validados en el servidor para tomar decisiones
- **Asumir que cualquier dato podría estar comprometido** al diseñar la aplicación

Denegar por defecto

- **Diseñar todas las funciones para comprobar que un usuario está autorizado a usarlas**
- **Verificar siempre la identidad (AuthN) antes de comprobar si se tiene permiso (AuthZ)**
- Reverificar el acceso en cada página o función de la app, incluso aunque solo se recargue
- Siempre cerrar la transacción correctamente en caso de fallo
  - Nunca caer en estados desconocidos
- Verificar la identidad (AuthN) y si se tiene acceso a una característica (AuthZ) en las APIs
  - En ambos sentidos (llamadas de una API a la aplicación y de la aplicación a la API)
- Bloquear cualquier protocolo, puerto, verbo o método HTTP que la aplicación no use

Aislamiento por diseño

- Diseñar el sistema para desplegar **solo una aplicación por servidor**, PaaS o contenedor

*Iconos por Bing Chat AI

Conceptos de diseño seguro

Separación de código por diseño

- **Separar el código que lidia con aspectos de seguridad del código funcional**
- Clases separadas, otra aplicación, usar un sistema de autenticación de un proveedor externo tipo Google, etc.
- Incluso en un servidores o contenedores separados
  - Protegido por un **firewall** que restrinja el acceso solo a las aplicaciones autorizadas
- También con los logs y monitorización, IDS/IPS, WAFs... (ver **Tema 5**)

Separación de tipos de funcionalidad por diseño

- **Separar el código de los administradores de la funcionalidad "estándar" de la aplicación**
- De la misma forma anterior

Diseño de la gestión de secretos de la aplicación

- **Definir dónde guardamos los secretos** (conexión a la BBDD, certificados, API Keys,…)
- Definir una **secret store**, incorporarla a la cadena CI/CD o definir cómo acceder por programa

*Iconos por Bing Chat AI

Conceptos de diseño seguro

- **CSRF**
  - Identificar los puntos en los que el usuario hace operaciones sensibles
  - Compras, cambios de clave…
  - Para pedirle su clave, un captcha o un token que solo él conozca
- **NUNCA usar datos de producción para testing**
  - Diseñar fuentes de datos separadas
- **Diseñar cómo proteger el código fuente de la aplicación a toda costa**
  - **Políticas y permisos de acceso a los repositorios**, ofuscación, backup, monitorización y log...
  - Los supply chain attacks son tan serios y frecuentes que se necesita una estrategia concreta para proteger tus repositorios de código
  - Ser open source permite a cualquiera ver el código y sugerir cambios, pero los cambios deben supervisarse adecuadamente para evitar introducir código malicioso
  - Para hacerlo correctamente necesitas seguir el OWASP Software Component Verification Standard
    - https://owasp.org/www-project-software-component-verification-standard/

Threat modeling

Identificando amenazas potenciales antes de construir la aplicación

*Iconos por Bing Chat AI

Fuente: Bing Chat AI

Threat modeling ("evil brainstorming")

- **Identificación de potenciales amenazas** para un negocio, sistema o software
- **Brainstorming** donde se piensan, buscan y definen los posibles peligros a los que nos podemos enfrentar
  - Intercepción o filtrado de datos con su posterior venta, y el impacto que puede tener
  - Cómo protegerse ante posibles ataques que identifiquemos
  - … (cualquier peligro, cualquier problema que imagines, piensa mal, piensa en peligros, adopta el lado oscuro como si fuera tu padre)

- **Si identificamos alguno debemos**
  - Cambiar el diseño de la aplicación para contrarrestarlo
  - Que un miembro del equipo responsable de ello acepte el riesgo de una forma documentada (DPPI)
- **Es algo que puede ser muy complejo, aquí daremos una introducción**
  - Para identificar riesgos nos podemos apoyar en los elementos del MITRE ATT&CK (**Temas 5 y 7**)

Threat modeling ("evil brainstorming"): ¿Qué necesitamos?

- **Un representante de cada stakeholder de la aplicación**
  - El **cliente**
  - Un **especialista en seguridad ofensiva**
  - **Arquitecto del sistema**
  - Responsables de la **plataforma de despliegue**
  - …
- **Cada grupo debe aportar los riesgos que ve en el sistema**
  - *"Si fueras a atacar el sistema, ¿Cómo lo harías?"*
  - *"¿Quiénes crees que atacarían el sistema?"*
  - *"¿Cómo te prepararías para un ataque?"*
  - *"¿Qué podría ir mal?"*
  - *"¿Cómo protegemos al usuario?"*
  - *"¿Qué es lo peor que puede pasar?"*
  - …

*Imágenes por Bing Chat AI

Threat modeling ("evil brainstorming"): ¿Qué necesitamos?

- **Deben verlo desde la mentalidad "del mal", como si fueran adversarios**
  - Convertir las "user stories" en "abuse stories" o "*casos de uso negativos*"
    - Ej.: Si una aplicación se supone que hace X, ¿qué pasa si hace Y?
  - Requiere práctica y detalle
- **Pero existen metodologías que facilitan u implementación**
  - 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
- **Todas estas metodologías permiten al equipo hacerse las preguntas necesarias**
  - Para cubrir todo lo que puede ir mal

THREAT MODELING ("EVIL BRAINSTORMING")

- El **threat modeling** permite pensar en...
  - Qué tiene nuestra aplicación de valor (dinero, datos…)
  - Cómo alguien podría robarlo, manipularlo (reputación, DoS…), romperlo o usarlo mal
- Las respuestas son una "**lista de problemas**"
  - Hay que evaluar cuáles **son riesgos**, asignándoles una **puntuación** (DPPI)
  - Y elegir si **mitigarlos** (cambiar los requisitos, el diseño, el código...) o **aceptarlos**
    - Solo por parte de la dirección del proyecto
  - Todo este proceso debe quedar **documentado**
  - El método cambia según cómo se hace el diseño
    - En **cascada** se hace una sesión al comienzo
    - Un **diseño iterativo** funciona mejor con varias sesiones más cortas en cada iteración…

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

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

¿Crees que lo has entendido todo? ¡Autoevalúate!

?

- **Eres capaz de hacer lo siguiente**
  - *Entender que el diseño debe ser un fiel reflejo de los requisitos*
  - *Comprender que, a la hora de proteger los datos, siempre debe seguirse una política existente aplicable en el contexto de la aplicación*
  - *Entender por qué es mejor diseñar asumiendo que cualquier dato puede ser comprometido*
  - *Comprender que, ante la duda sobre lo que hacer, lo más seguro es denegar una petición*
    - *Incluyendo que antes de autorizar algo primero hay que asegurarse verificar la identidad y luego los permisos sobre ese algo de dicha identidad*
  - *Entender que es mejor diseñar una aplicación por partes aisladas todo lo posible*
    - *Usando distintos criterios, incluyendo las fases en las que se divide el desarrollo (datos de **testing** distintos de datos de producción)*
    - *Y, además, proteger el acceso al código fuente en aquellos casos donde no sea público*
  - *Saber que en la fase de diseño debe eliminarse toda posibilidad de "hardcodear" información*
  - *Comprender qué es, quien realiza y para qué sirve el threat modeling*
    - *Incluyendo entender qué, al final del proceso, tendremos una lista de riesgos identificados que debemos estudiar cómo tratar*

< Ir al Índice

Fuente: Bing Chat AI

AuthN y AuthZ >

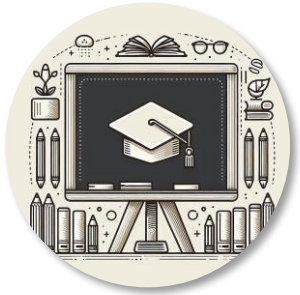
Log y errores >

Usabilidad >

CREACIÓN DE CÓDIGO SEGURO

Aspectos a tener en cuenta en la implementación

(ver Seminario 5 para ejemplos concretos)



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

*Iconos por Bing Chat AI

¿QUÉ TE VAS A ENCONTRAR EN ESTE BLOQUE?

- **En este bloque aprenderás**
  - Criterios para **seleccionar los frameworks** y lenguajes de desarrollo
  - Un **algoritmo** concreto **para validar datos**
  - Elementos importantes para una **implementación de una web segura**
  - Criterios para implementar una **gestión de identidades y autenticación** seguras
    - Incluyendo la importancia de **externalizar el servicio** y la mejor política de autorización
  - Criterios a la hora de **implementar el log** de una aplicación
    - Incluyendo su **protección**
  - Aspectos básicos de **seguridad y usabilidad**

SELECCIONAR EL FRAMEWORK Y EL LENGUAJE DE PROGRAMACIÓN

- **En muchos proyectos esto no es una elección real**
  - Ya viene dado por el entorno de desarrollo
  - Si estamos en un ecosistema Java, no vamos a tener mucha libertad de elección…
- **Pero si tenemos la opción de elegirlo, hay que seguir unas reglas**
  - Mirar **cuánto tiempo va a estar soportada** la versión elegida
    - Cuanto más, mejor (versiones LTS (Long-Term Support))
    - Nunca usar algo cerca del fin de su ciclo de mantenimiento
    - En general, usar la última o penúltima versión de soporte largo disponible
  - Mirar el **historial de vulnerabilidades** en un repositorio de **CVE** (**Tema 1**)
  - **No usar** "el ultimo framework o lenguaje de moda" (para proyectos "serios")
    - Hasta que no esté establecido y se sepa que tendrá un soporte constante y duradero
  - **Minimizar** la cantidad de librerías, componentes, frameworks, etc. usados (**superficie de ataque**)
  - Escanear la versión usada con una herramienta **SCA** para ver si tiene vulnerabilidades si es posible
    - Más sobre esto después
- **No uses lenguajes no memory-safe como C/C++, porque introducen problemas**
  - Una alternativa memory-safe es Rust

DATOS NO CONFIABLES: VALIDACIÓN

- Se basa en implementar lo dicho
- **Siempre validar del lado del servidor**
  - Hagamos o no validaciones en el cliente con JavaScript
  - Bueno para usabilidad, terrible para seguridad
- Antes de usar cualquier entrada de usuario
- Sigue este esquema como guía

VERBOS HTTP

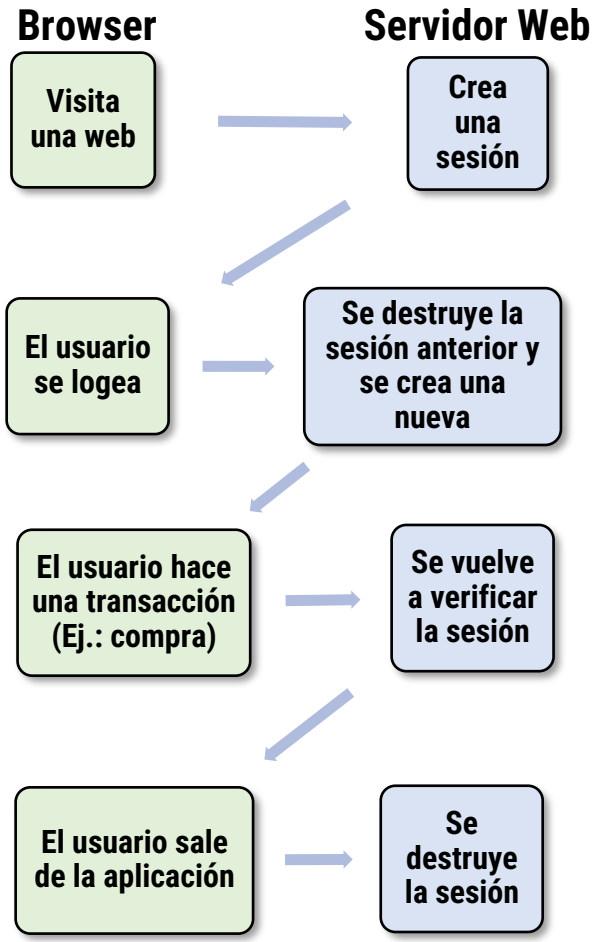
- **Si una aplicación usa el protocolo HTTP...**
  - Hay varias acciones que puede realizar y que se modelan a través de verbos HTTP (`GET`, `POST`, `PUT`, `DELETE`...ver SEW)
  - **Elimina el soporte de todo verbo que no se use en la aplicación**
    - Para evitar dar demasiada información y posibles ataques
  - Algunas extensiones del protocolo HTTP usan verbos adicionales
    - Debemos asegurarnos de no eliminar verbos legítimos
  - Una aplicación web típica normalmente sólo usa `GET`, `POST`, `OPTIONS` y `HEAD`
- **Se recomienda seguir los CIS Benchmarks para desactivarlos**
  - Podemos hacerlo a nivel servidor web o framework
  - ¡Acuérdate del **Tema 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

Cuidado sobre todo en aplicaciones REST.
Fuente: https://intmain.co/http-request-methods-rest-api-verbs

Manejo de sesiones

- **Las sesiones son un token transmitido entre el cliente y el servidor**
  - Que identifica de forma única a un usuario en un momento en el tiempo
- **Una correcta gestión de IDs de sesión requiere**
  - Deben tener **al menos 128 caracteres**
  - Deben ser **impredecibles** (aleatorios) para evitar **precalcularlos**
    - Se debe usar un generador de IDs aleatorios reconocido y no propio
  - El usuario debe recibir un nuevo ID **cada vez que se autentique**
  - **Usar, si existe, el manejador de sesiones del framework**
  - Debe tener una **fecha de expiración,** aunque el usuario siga en la aplicación
  - Solo debe **transmitirse por canales cifrados**
  - **Nunca aceptar** un ID de sesión que no se ha generado por la aplicación
    - Debe generar una alerta de log/SIEM y bloquear la IP (incidente de seguridad)
  - Deben **regenerarse** en cada autenticación, re-autenticación
    - U otro evento que cambie el nivel de privilegio del usuario
  - **Muchos frameworks ya tienen esto**: investígalo y **no lo programes tú**



Identidad y autenticación (AUTHN)

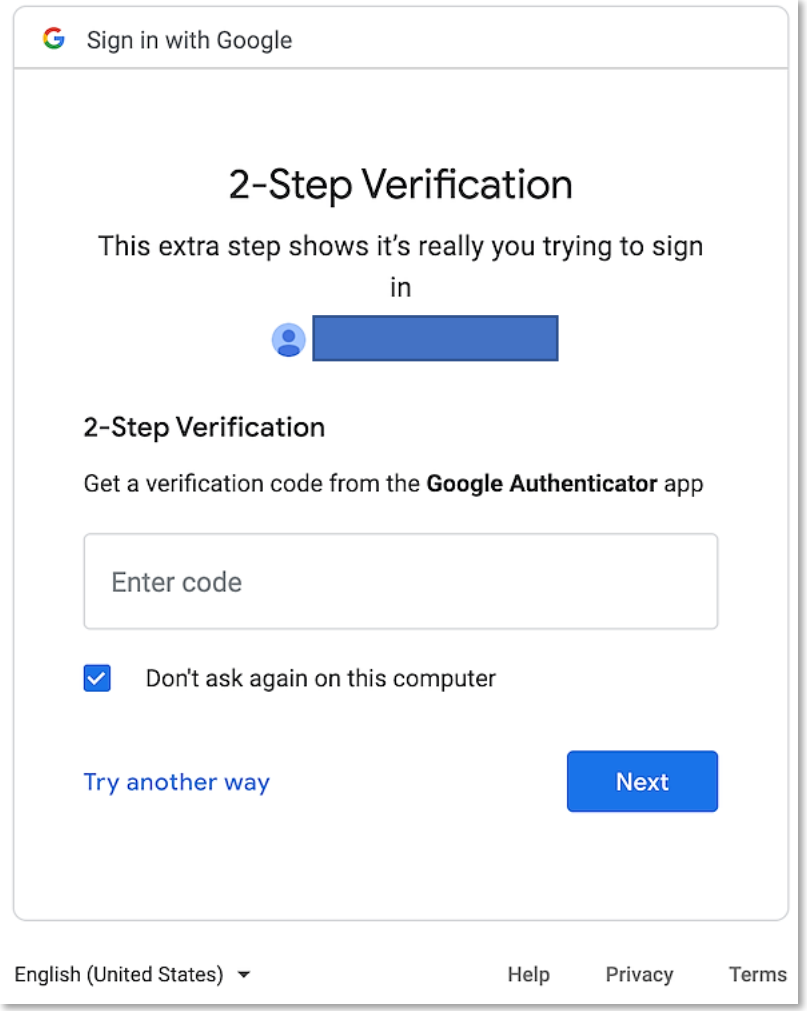
Como implementar la comprobación de identidad y los permisos de nuestros usuarios

Fuente: Bing Chat AI

*Iconos por Bing Chat AI

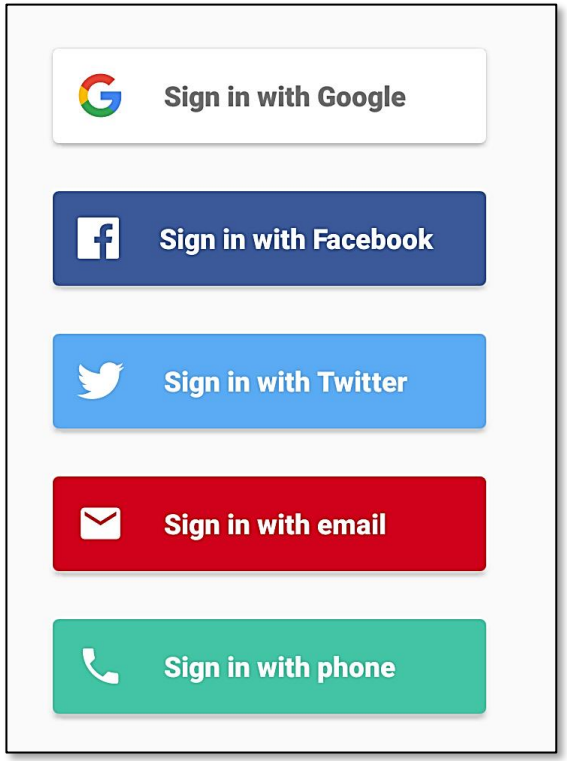
NO CREES TU PROPIO SISTEMA DE AUTENTICACIÓN

- Es común tener un sistema de identidades basado en **Active Directory** de **Microsoft**
- Otros proveedores de nube típicos tienen sus sistemas propios de autenticación. Ej.:
  - **Google**: https://cloud.google.com/docs/authentication
  - **Amazon**: https://aws.amazon.com/es/cognito/
- Debemos elegir y usar uno que sea compatible con nuestros requisitos
  - Y el negocio donde se va a implementar la aplicación
- Si esto no fuera una opción por tener unos requisitos muy particulares...
  - Lo mejor es usar un protocolo establecido como OAUTH: https://oauth.net/getting-started/



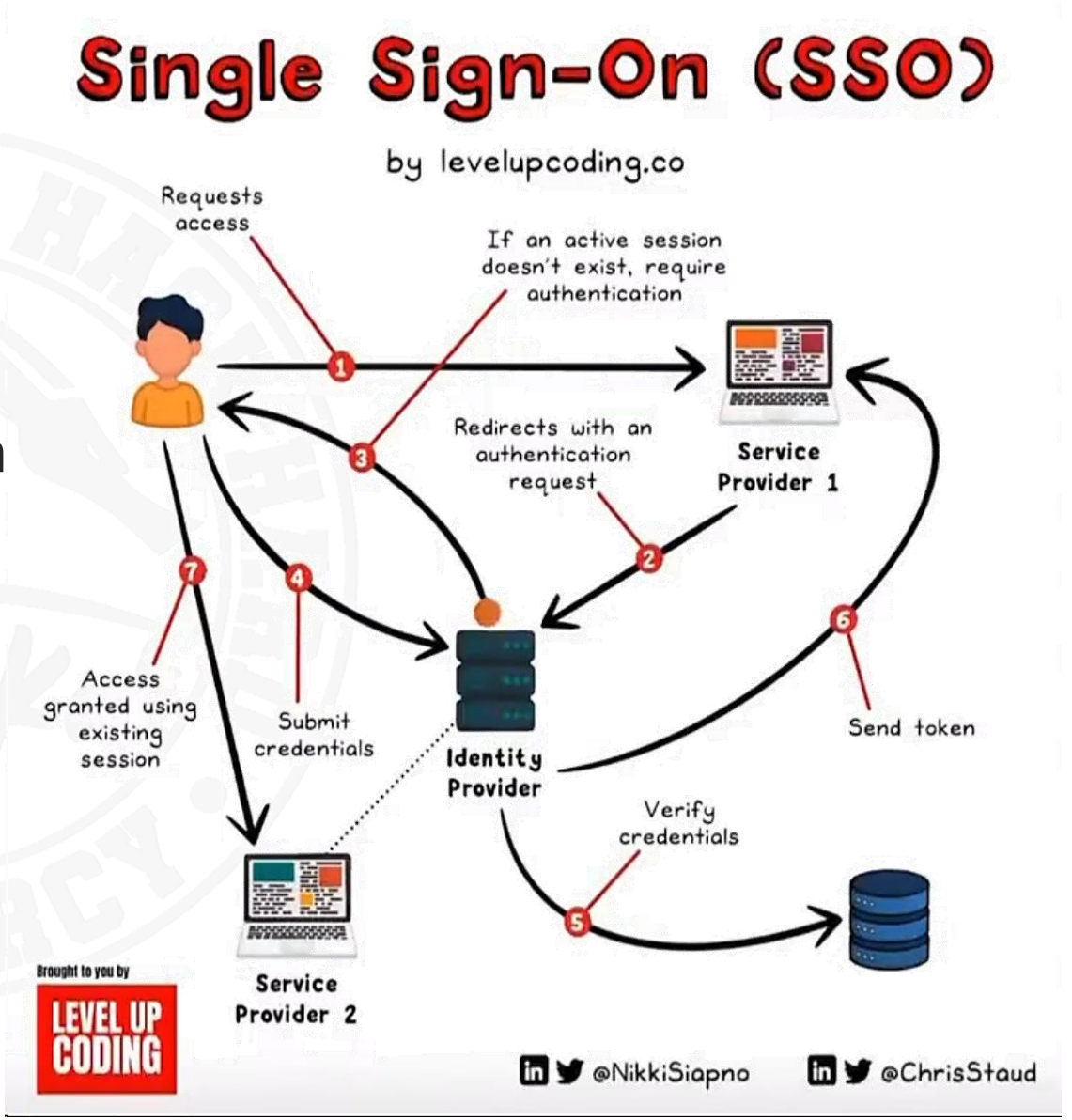
USAR SIEMPRE UN PROVEEDOR DE AUTENTICACIÓN DE TERCEROS

- **Google, Microsoft, LinkedIn, Twitter, Facebook…**
- **Ventajas**
  - Se implementa **rápidamente** (hay guías de implementación)
  - Nos **libramos de la responsabilidad** de mantener o testear ese código
  - El código de autenticación es **más seguro**, está más probado
  - Los usuarios pueden **confiar más** en estos sistemas
- **Desventajas**
  - Algunos usuarios no quieren compartir datos entre distintas empresas
    - Y EXIGEN tener logins separados para cada sitio
  - Algunos usuarios pueden no usar ninguno de estos proveedores
    - **Considera permitir usar varios** para contrarrestarlo
  - Podrían conllevar un coste
  - Estos servicios **pueden recoger información personal** de los usuarios
    - Inaceptable en ciertos contextos
  - ¡Una brecha en el sistema de 3ºs es automáticamente una **que TENEMOS que asumir**!



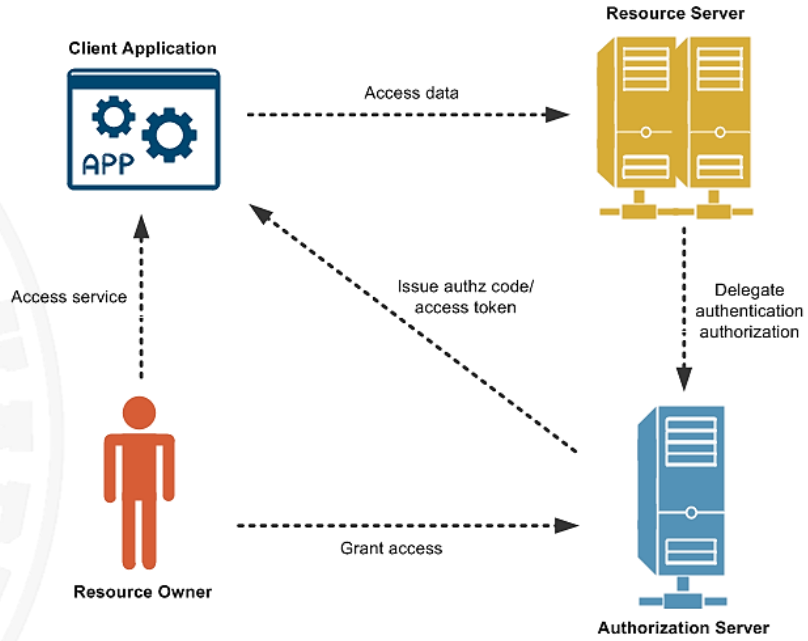
Usar siempre un proveedor de autenticación de terceros

- **Usar un proveedor de terceros también facilita el uso de Single Sign-On (SSO)**
  - El usuario introduce correctamente su usuario y contraseña (y su 2FA) en un servicio de autenticación
  - Al hacerlo, **tiene acceso a una o varias aplicaciones** vinculadas al dicho servicio de autenticación
- **Facilita la administración de cuentas**
  - Ya no es necesario tener varias, una para cada aplicación
  - Mejora la experiencia del usuario



LIBRERÍAS PARA LA AUTENTICACIÓN

- Comprar o usar una librería o sistema **software** para implementar autenticación
- Ventajas
  - **Más rápido y mucho más adecuado** que hacer una propia
  - La librería solo nos deja la responsabilidad de **implementar su uso**
    - No diseñar el sistema de autenticación
  - Si la librería tiene un soporte adecuado, puede ser muy segura
  - **Los datos de los usuarios están en nuestro poder**
    - Lo que puede ser conveniente en muchos escenarios
- Desventajas
  - Podría tener un **coste asociado**
  - Una **vulnerabilidad** en la librería es automáticamente nuestra hasta que no la actualicemos / mitiguemos
  - **Somos responsables de los datos de usuario**
    - Con los temas legales que conlleva y que debemos considerar (ver ASLEPI)



Fuente: docs.oracle.com

Autorización (AuthZ)

- **Determina lo que el usuario puede o no hacer en el sistema**
  - Una vez se sabe quién es (**AuthN**)
- **RBAC (Role-Based Access Control)**
  - **Se crean una serie de roles de usuario en el diseño**
  - Se les asignan los permisos mínimos para hacer un determinado trabajo
    - Ej.: Tester, desarrollador, administrador de base de datos….
  - A los usuarios se le asignan uno o varios roles
    - Sólo esos roles determinan lo que puede o no hacer
  - Cuando un usuario se va o promociona, se le quitan o cambian sus roles
    - Y con ello automáticamente cambian sus permisos (como los grupos de Windows, ASR)

- **DAC (Discretionary Access Control)**
  - **El propietario del recurso puede dar o quitar permisos como quiera**
  - Como el sistema de ficheros de Linux
  - Pero pueden aplicarse a entidades más abstractas, como “usuarios en la misma red”, etc.

*Iconos por Bing Chat AI

Autorización (AuthZ)

MAC (Mandatory Access Control)

- **El acceso a un recurso se basa en lo privado que es**
- Y si el usuario tiene el suficiente privilegio para acceder a recursos con ese nivel de privacidad
- Idéntico al descrito en el tema 3: Seguridad de SO

PBAC (Permission-Based Access Control)

- **Se les otorgan a los usuarios ciertos permisos sobre la información o los sistemas**
- Que se comprueban al acceder al mismo
- No hay roles
- Se otorgan permisos a usuarios individuales o a varios a la vez

*Iconos por Bing Chat AI

Manejo de errores, log y monitorización

Qué hacer con la información que podemos obtener de nuestra aplicación mientras está en uso

*Iconos por Bing Chat AI

Fuente: Bing Chat AI

REGLAS DE MANEJO DE LA INFORMACIÓN DE ERROR Y LOG

- **TODOS** los errores de la aplicación deben ser capturados y manejados
  - Hacer un `catch` global (en el `main`) es buena idea como último recurso
  - Para capturar errores inesperados y tratarlos de manera adecuada (¡**nunca lo dejes vacío**!)
- La información interna del error (trazas de pila…) **NUNCA** debe revelarse al usuario
  - Los errores **no deben mostrar información técnica** a los usuarios (nunca versiones de productos, tecnologías…)
- Como ya hemos dicho, nunca decir si se ha introducido mal un usuario o una clave
  - Dar un error idéntico en ambos casos (**previene la enumeración**)
- Los errores de seguridad (de login, de acceso, validación…) **deben lanzar una alerta**
  - Idealmente los logs de la aplicación se mandan a un IDS/IPS o un SIEM que los analice
  - Pueden provocarse ejecutando alguna herramienta de escaneo automático de vulnerabilidades (**Seminario 3**)
- Cuando el error incluye información de otros sistemas…
  - Codificar el contenido, eliminar caracteres no imprimibles y limitar el tamaño de la información que se registra

Reglas de manejo de la información de error y log

- **Hay sistemas de monitorización que analizan los log que se les mandan buscando problemas (Ej.: los SIEM vistos en el Tema 5)**
  - Muchos de ellos los analizan usando distintas técnicas de ML/IA (Ej.: las reglas vistas en SI)
  - Debe usarse un formato compatible con nuestro SIEM (**Tema 5**)
  - **Reglas generales**
    - Registrar los login fallidos, correctos y errores
    - Registrar intentos de login por fuerza bruta (Ej.: más de 10 intentos de login fallidos en menos de 1 minuto)
    - Registrar en general todos los eventos relacionados con seguridad
      - Usuario autenticado, usuario bloqueado tras varios intentos fallidos, validaciones fallidas…
  - **Reglas sobre la información que contienen**
    - **Recuerda**: Los logs **no deben contener información privada o personalmente identificable**
    - El **tipo** de evento que ha ocurrido (de seguridad / otro)
    - **Cuándo** ha ocurrido (timestamp)
    - **Dónde** ha ocurrido, tanto a nivel de dirección como de sistema (sistema, IP, URL, subdominio….)
    - El **resultado** del evento
    - Si es posible, identificar al **usuario** u **origen** asociado al evento

PROTECCIÓN DE LOGS

- **Deben protegerse los logs de accesos no autorizados**
  - Y hacerles copias como parte de la estrategia de backup de la empresa
- **Los accesos a los logs deben registrarse y monitorizarse**
- **Deben guardarse cifrados y en un servidor seguro**
  - Ver **SNI**, **Tema 5**
- **Debe ser accesibles por los equipos de respuesta a incidentes**
- **Deben borrarse siguiendo periódicamente**
  - Siguiendo la misma política de la empresa para información sensible
  - Y, por supuesto, las normas legales de nuestro país
- **Esto enlaza con la optativa Informática Forense y Auditoría (IFA)**

*Imágenes por Bing Chat AI

Seguridad y Usabilidad

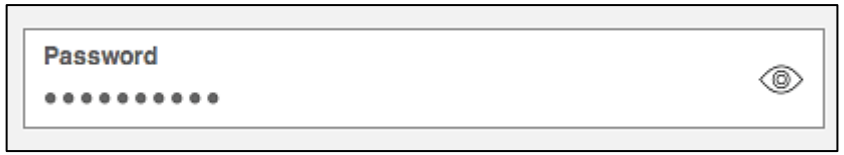
La seguridad debe ser usable, o nuestro producto podría fracasar

*Iconos por Bing Chat AI

Fuente: Bing Chat AI

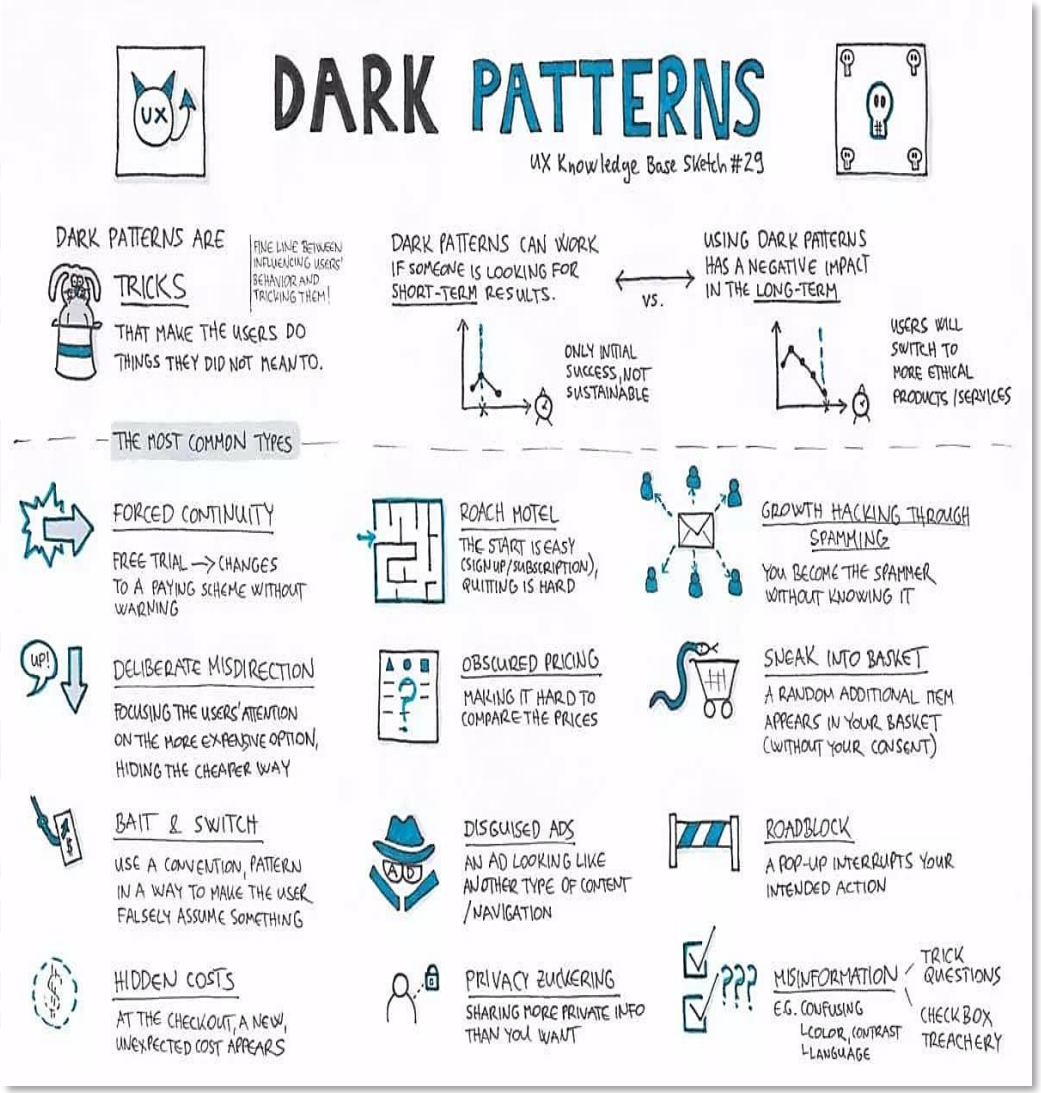
Seguridad y Usabilidad

- **Si las medidas de seguridad de una aplicación la hacen difícil de usar, los usuarios tenderán a tratar de evitarlas**
  - ¡O irse a la competencia!
- **Es necesario hacer un esfuerzo en hacer la seguridad usable. Ej.:**
  - Permitir desbloquear un dispositivo o app mediante huella o reconocimiento facial y no una password
  - Enseñar a los usuarios a crear claves a partir de frases y no las típicas reglas de complejidad
  - Enseñar a los usuarios a usar **Password Managers**
    - Para usarlos tu aplicación **no debe desactivar copiar y pegar** sobre los textfields de las claves
  - …
- **Este es un tema complejo que excede las competencias de esta asignatura**
  - Y está relacionado con lo que se impartió en CPM
- **Algunos materiales sobre diseño de UI desde la perspectiva de seguridad**
  - https://uxdesign.cc/designing-uis-with-security-in-mind-f4f6f265573a
  - https://www.w3.org/TR/UISecurity/#intro



Anti-Usabilidad: "Dark Patterns"

- **Diseñar programas para "engañar" a los usuarios y que hagan cosas que no quien hacer**
  - Comprar productos, suscribirse a un servicio… (Ej.: artículos de más en carrito, falsos "X personas están viendo esto"...)
  - ¡Algunas pueden comprometer la seguridad de los usuarios!
- **Se basan en que los usuarios no leen la página completa**
  - Y/o asumen que se va a comportar de una forma dada
  - **Las páginas se diseñan para engañar**; parece que dicen una cosa, pero realmente dicen otra
- **Como usuarios debemos conocer estas prácticas para defendernos de ellas**
  - Así **evitamos reproducirlas en nuestras aplicaciones**
- **Lista de "dark patterns":**
  - https://www.darkpatterns.org/types-of-dark-pattern



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

¿Crees que lo has entendido todo? ¡Autoevalúate!

?

Eres capaz de hacer lo siguiente

- *Entender los criterios para elegir un framework o lenguaje de desarrollo correctamente*
- *Comprender el algoritmo con el que debes implementar la validación de todo dato de entrada*
- *Entender que debes minimizar el nº de verbos HTTP que tu aplicación admite*
- *Comprender el algoritmo a implementar cuando se deban gestionar sesiones*
- ***Sobre identidad y autenticación***
  - *Entender por qué no se debe crear nunca nuestro propio sistema para autenticar usuarios*
  - *Comprender el concepto de Single Sign-On*
  - *Entender los distintos tipos de autorización que puedes implementar, sus ventajas y desventajas*
- ***Sobre log***
  - *Entender qué cosas no deben aparecer nunca en un log de tu aplicación*
  - *Comprender que tu aplicación seguramente esté en un ecosistema más grande, y que el formato de los logs debe adaptarse al mismo*
  - *Tener muy claro que la manipulación y alteración de los logs por parte de personas no autorizadas nos deja ciegos ante ataques que se hagan producido*
- ***Sobre seguridad y usabilidad***
  - *Comprender como implementar la aplicación para facilitar el uso de password managers*
  - *Entender qué es un dark pattern y para qué se usa*

< Ir al Índice

Fuente: Bing Chat AI

H. Estáticas >

H. Dinámicas >

AST (Application Security Testing)

Herramientas y técnicas para comprobar la seguridad del software

*Iconos por Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ TE VAS A ENCONTRAR EN ESTE BLOQUE?

- **En este bloque aprenderás**
  - Qué son las herramientas **SCA**
  - Qué son las herramientas **SAST**
    - Incluyendo como tener ambas fácilmente si usas GitHub
  - Qué son las herramientas **DAST**
  - Qué son las herramientas **IAST**
  - La importancia de la integración de todas estas herramientas en un ciclo de desarrollo **SecDevOps**

TESTING

- **Las actividades de testing son críticas para detectar problemas de seguridad**
  - Se usa el mismo método para hacer pruebas funcionales que se da en CVVS
  - Pero para los "abuse stories" o "*casos de uso negativos*" mencionados
- **Es posible que la seguridad de nuestros productos software sea evaluada por 3ºs**
  - Normalmente, para ser aceptado en varios países / contextos
  - Para ello se usan los Common Criteria: Pruebas funcionales para requisitos de seguridad
    - https://www.commoncriteriaportal.org/index.cfm
    - Cada país tiene un organismo certificador; el **Centro Criptológico Nacional** (CCN) es el de España
- **El testing implica introducir herramientas AST en el proceso de desarrollo**
  - Lo cual es especialmente relevante si seguimos un modelo de desarrollo DevOps con un esquema CI/CD (ASW)
    - Todas estas herramientas deben integrarse en nuestro pipeline
    - Lo convertiremos en SecDevOps
  - Algunas tareas de seguridad en cada fase pueden realizarse en paralelo con otras

Herramientas estáticas: SCA y SAST

Analizando el código antes de la ejecución

*Iconos por Bing Chat AI

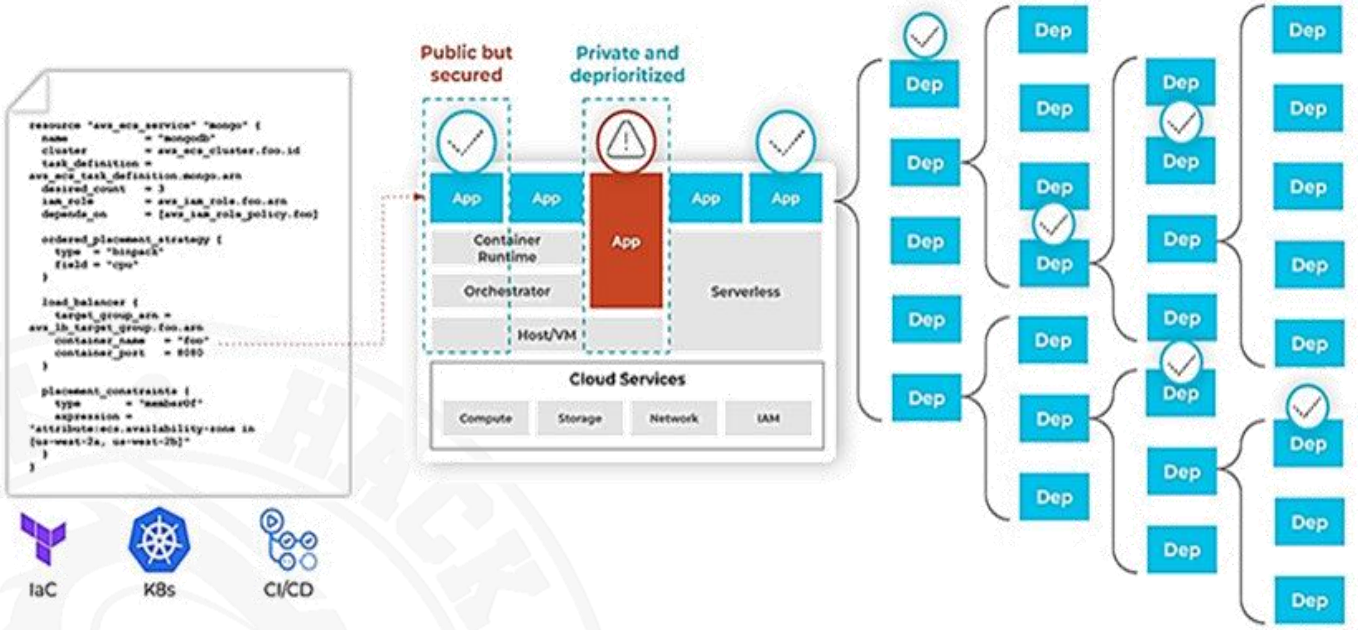
Herramientas SCA (Software Composition Analysis)

- **Gestionan el uso de componentes open source de una aplicación**
  - Escanean automáticamente el código de este tipo usado, incluyendo artefactos como contenedores y registros
- **Tienen estos objetivos**
  - **Inventario**: Recopilan todos los componentes open source de la aplicación, dependencias directas e indirectas
    - Así es posible saber todo lo que se está usando, lo que es clave para controlar su seguridad
  - **Cumplimiento de licencias**: proporcionan información sobre cada uno de los componentes identificados
    - Incluye si el tipo de licencia es compatible con la política de la organización y la atribución de autoría
  - **Vulnerabilidades de seguridad**: identifica vulnerabilidades conocidas en estos componentes (**CVE**)
    - **Es su principal función** (ver transparencia siguiente)
    - Informan si la aplicación llama a código vulnerable, sugieren formas de mitigarlo, gestionan actualizaciones/parches
  - **Características avanzadas**: Obligar a todo componente open source a cumplir la política de la organización,
    - Con respuestas automatizadas (solicitar aprobación del componente, impedir la construcción de la aplicación...)
    - También avisan de las vulnerabilidades de un componente antes de hacerle un pull request y que entre al sistema
- **Ejemplos**
  - https://sourceforge.net/software/software-composition-analysis-sca/
  - https://www.paloaltonetworks.com/cyberpedia/what-is-sca

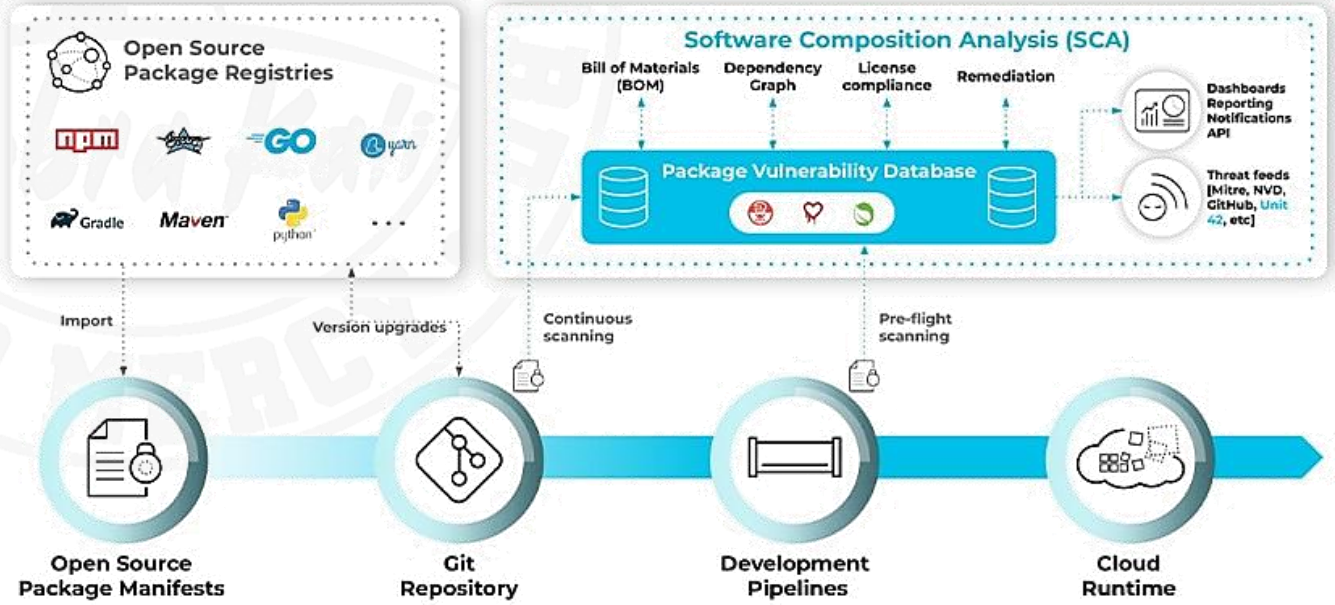
HERRAMIENTAS SCA

Seguridad de Sistemas Informáticos

- **Una aplicación moderna tiene una gran cantidad de dependencias**
  - Muchas de ellas son **open source**, por lo que podemos revisar su código
- **Las SCA analizan el código de esas dependencias open source**
  - Evitan que se introduzcan sus vulnerabilidades en nuestra aplicación
  - Si no usamos una SCA, alguna dependencia vulnerable "envenenaría" nuestro **software** tarde o temprano ☹
- **Todo el proceso de análisis se puede integrar automáticamente en la pipeline de desarrollo**



Una herramienta SCA analiza el código de todas las dependencias de código abierto de la aplicación dentro del pipeline de la misma. Fuente: https://www.paloaltonetworks.com/cyberpedia/what-is-sca



Herramientas SAST (Static Application Security Testing)

Seguridad de Sistemas Informáticos

- **Prueban estructuras internas o código de la aplicación**
  - No su funcionalidad (pruebas de caja blanca)
- **Ventajas**
  - **Soportan muchos lenguajes**: PHP, Java y .Net, C, C++, C#...
  - Descubren vulnerabilidades complejas **durante las 1ªs etapas de desarrollo**, que se pueden resolver rápidamente
  - **Detallan el problema**, incluida la línea de código
  - Se puede integrar en un entorno SDLC existente
- **Inconvenientes**
  - Escalar esta tecnología a grandes desarrollos puede ser un desafío
  - Suelen ser **inexactas**: muchos falsos positivos y falsos negativos
    - Aún peor con lenguajes de tipado dinámico
  - No pueden probar la aplicación en el entorno real
    - **No detectan vulnerabilidades en la lógica de la aplicación**
    - O configuraciones inseguras

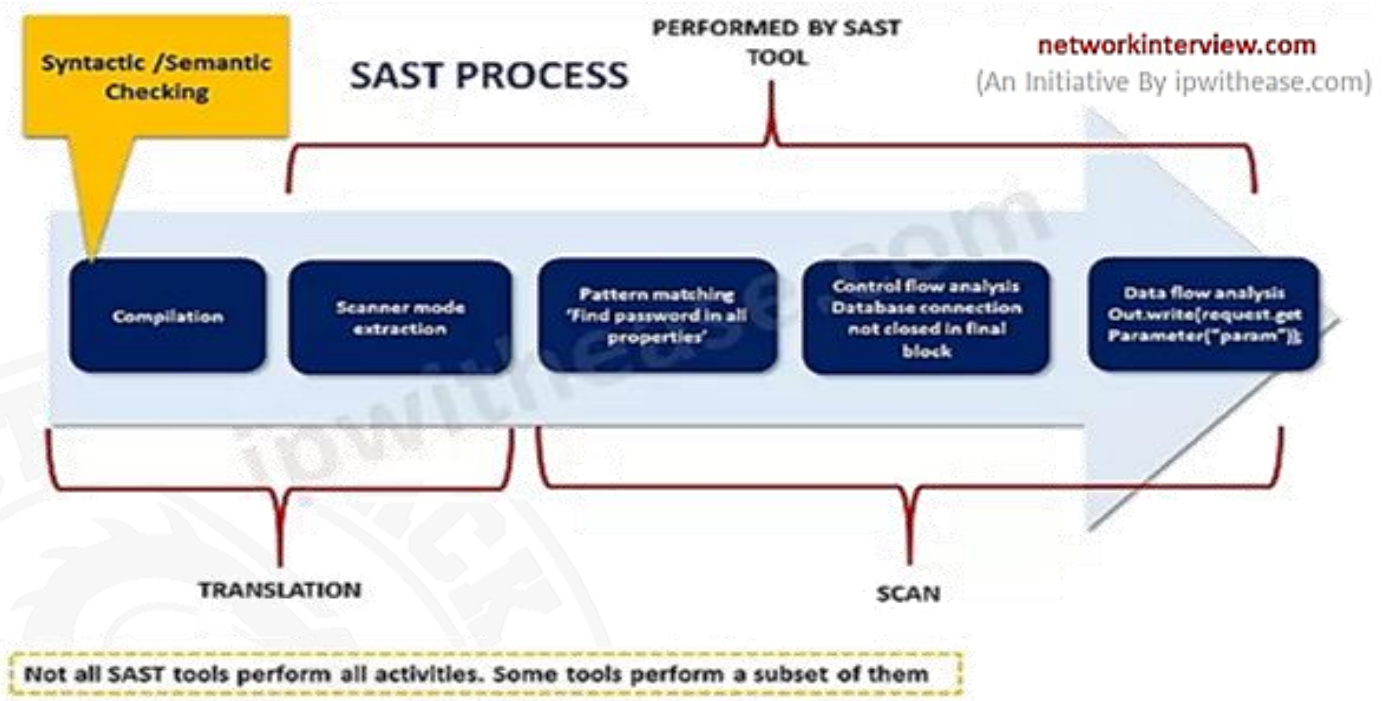
Fuente: https://www.mend.io/resources/blog/sast-vs-sca/

*Iconos por Bing Chat AI

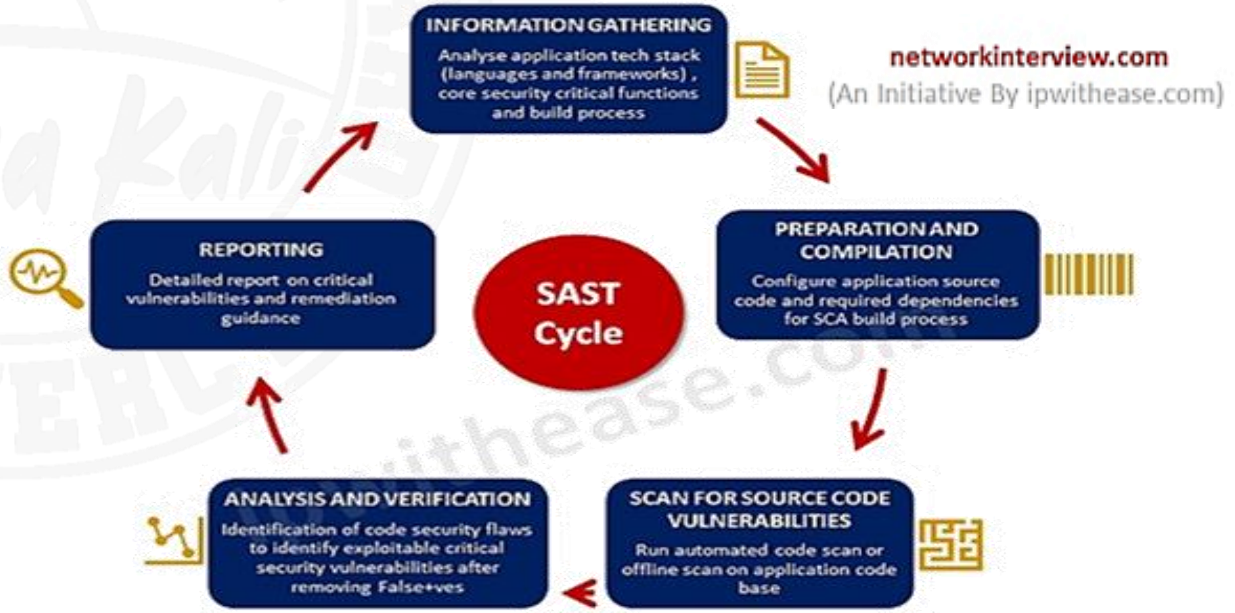
HERRAMIENTAS SAST

Seguridad de Sistemas Informáticos

- Puedes considerar una SAST como una extensión del proceso de compilación del **software** (DLP)
- En lugar de errores de tipo, busca patrones conocidos de malas prácticas de código
  - Es como un procesador de lenguaje especializado en “pifias”
- Tras acabar su análisis, da un informe independiente al del propio compilador
  - Con todos los detalles de los problemas encontrados y sugerencias de solución

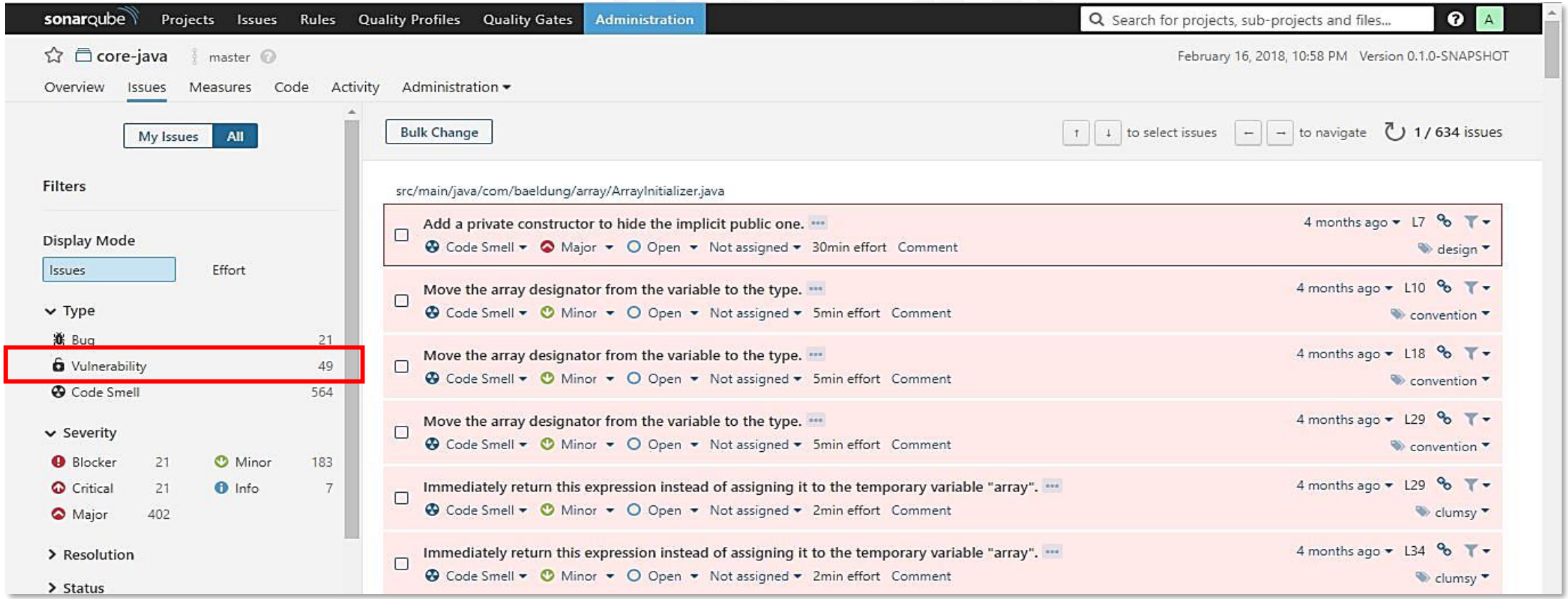


Una herramienta SAST actúa tras el análisis sintáctico y semántico (DLP).
Fuente: https://networkinterview.com/sast-static-application-security-testing/



HERRAMIENTAS SAST

- **Una herramienta SAST muy popular es SonarQube (vista en Lab 10)**
  - Más herramientas: https://www.g2.com/categories/static-application-security-testing-sast
  - Pasar una SAST y una SCA a vuestro **TFG** es un buen plus (guiño, guiño ☺)



Informe de SonarQube dejando en evidencia un desarrollo (presuntamente ☺): https://www.baeldung.com/sonar-qube

Fuente: Bing Chat AI

¡SAST y SCA fáciles y rápidos con GitHub!

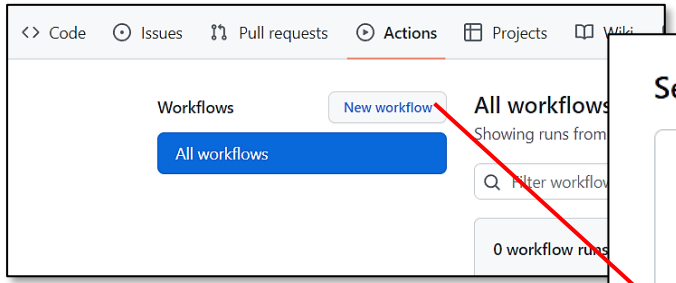
*Logo oficial de GitHub

SCA y SAST muy fácil con GitHub actions y workflows

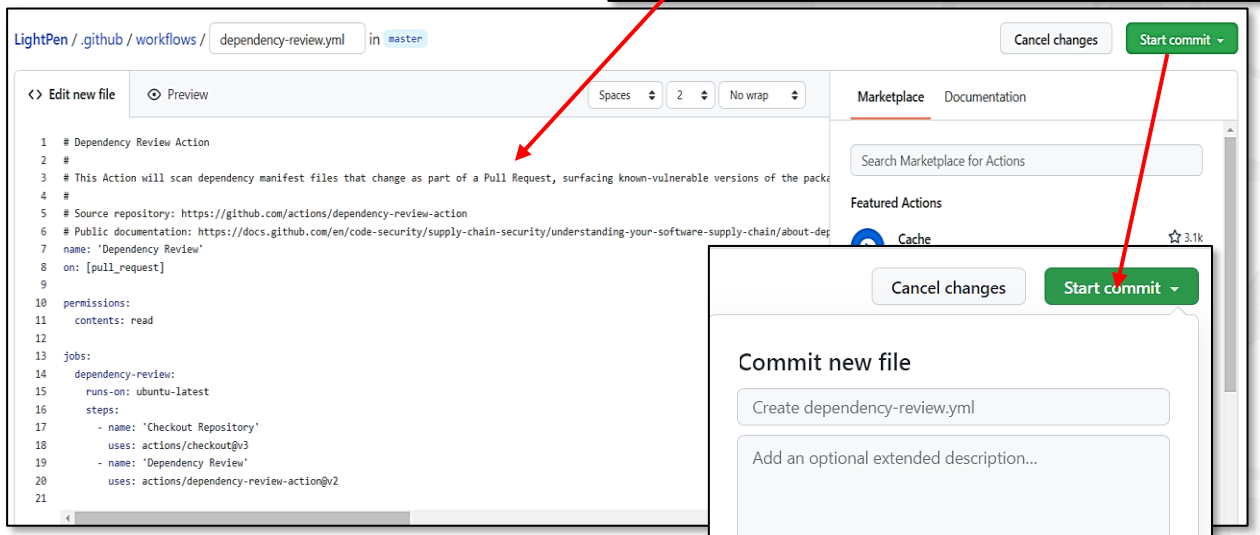
- **GitHub** te da la capacidad de usar las funciones SCA y SAST muy fácilmente
  - En principio sólo para repositorios públicos
- Integra escáneres SCA y SAST como parte de sus **GitHub Actions**
  - Entre **otras muchas funcionalidades**
  - Por lo tanto, puedes lanzar una gran cantidad de procesos automatizados para mejorar tú código
  - Uno de los gratuitos más populares es **CodeQL**
  - Sigue estas instrucciones: https://docs.github.com/es/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/setting-up-code-scanning-for-a-repository
  - Consulta otras **GitHub Actions** disponibles para encontrar más herramientas AST que se ajusten a tus desarrollos
    - ¡Y mejorar tu seguridad!

SCA y SAST muy fácil con GitHub Actions y workflows

Seguridad de Sistemas Informáticos

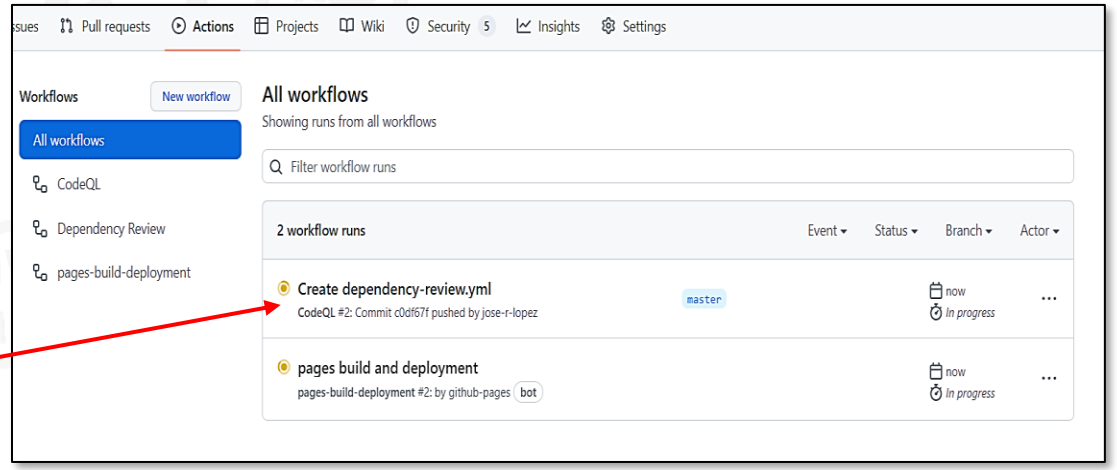


1. Crear un nuevo workflow



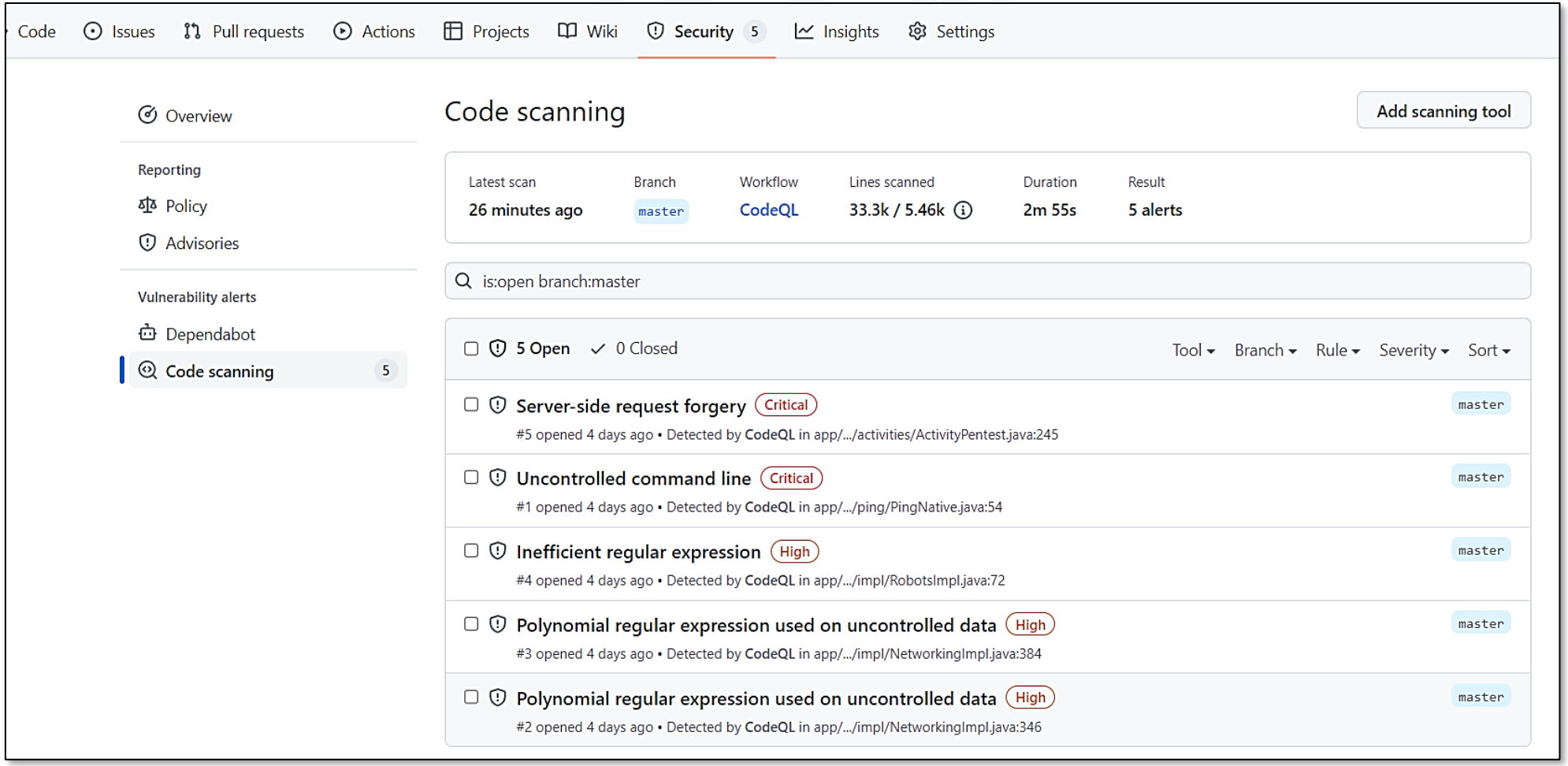
3. **Dependency review** hace un escaneo SCA y SAST con el software **CodeQL**. Acepta el fichero que se crea y hazle commit

4. Vete a "Actions" y espera a que la tarea termine



SCA y SAST muy fácil con GitHub Actions y workflows

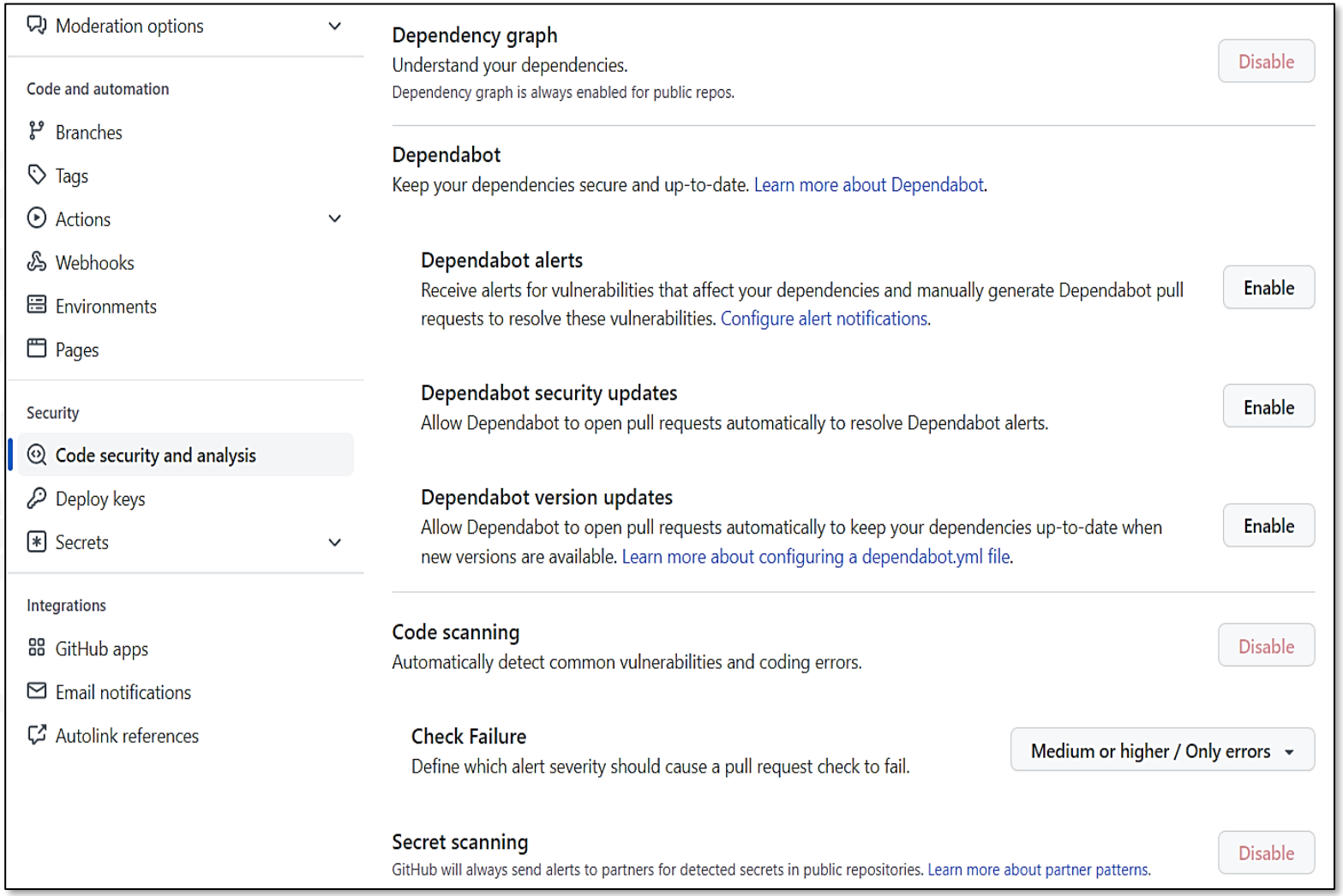
- ¡Y simplemente haciendo esto ya témenos características SCA (Dependabot) y SAST (Code scanning) en nuestro código!



SCA y SAST muy fácil con GitHub Actions y workflows

Seguridad de Sistemas Informáticos

- Las opciones de **Code security and analysis** también activa SCA y SAST automáticamente
- Y **secret scans**
  - ¡Si has hardcodeado secretos en tu código te lo detectará!
- ¡Estas opciones son **CLAVE** para implementar Código seguro!



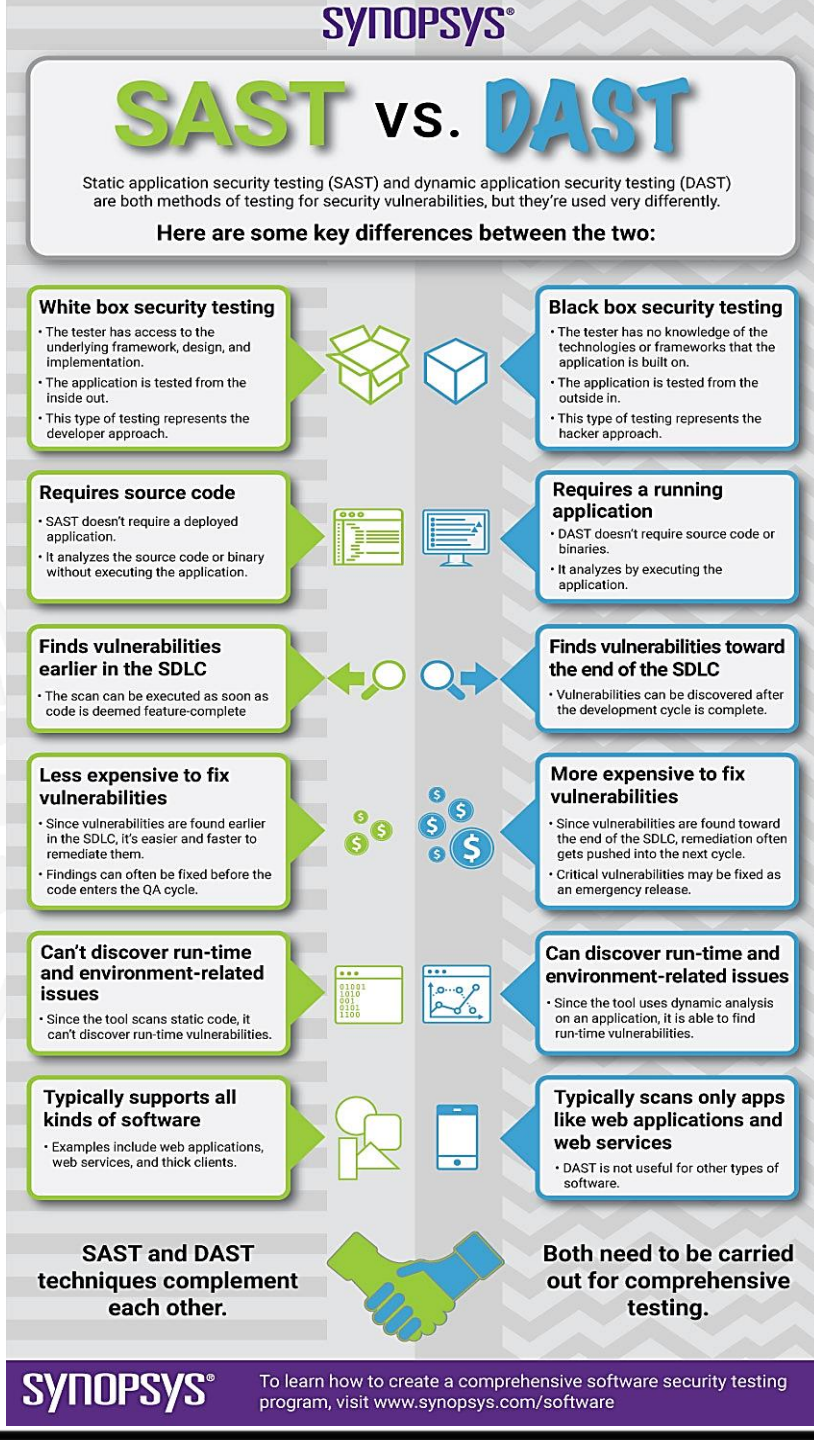
Herramientas dinámicas: DAST e IAST

Analizando el código tras su ejecución

*Iconos por Bing Chat AI

Herramientas DAST (Dynamic Application Security Testing)

- **Método de prueba de caja negra que introduce errores para probar las rutas de ejecución que estos provocan**
  - No requiere código fuente ni binarios; analiza la aplicación en ejecución
- **Ventajas**
  - A diferencia de SAST, permite detectar **problemas en ejecución**
    - Errores de autenticación y configuración de red, problemas tras logon...
  - Hay menos falsos positivos
  - Compatible con todos los lenguajes y/o frameworks personalizados
- **Inconvenientes**
  - **No dan información sobre las causas** de las vulnerabilidades
  - No adecuadas en las 1ªs etapas (necesitan la aplicación en ejecución)
  - No simula una parte de los posibles ataques
- **¿Quieres incorporar una gratuita a tu pipeline?**
  - Mira OWASP ZAP: https://owasp.org/www-project-zap/

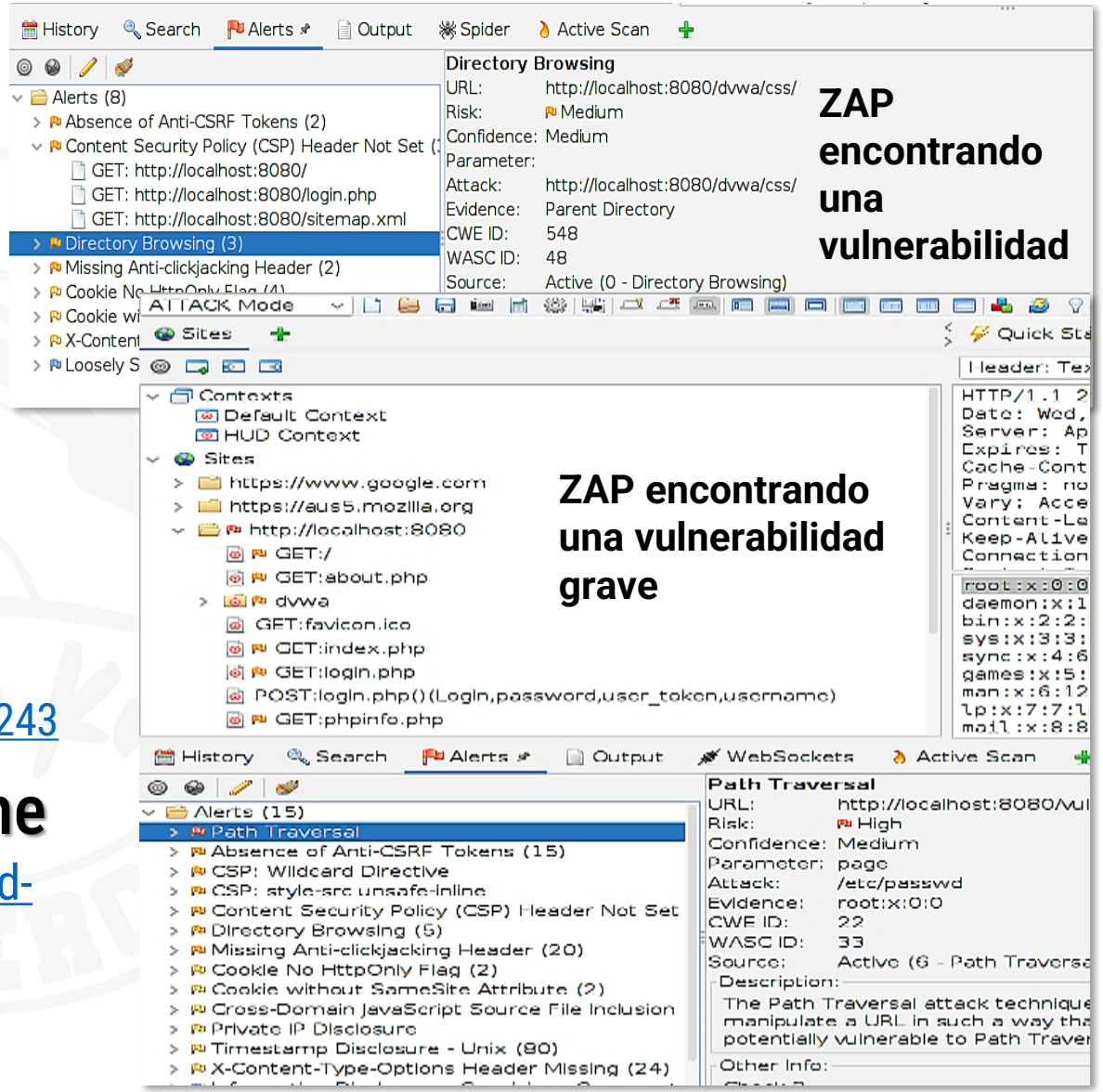


*Iconos por Bing Chat AI

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

Herramientas DAST (Dynamic Application Security Testing)

- **Una herramienta DAST muy popular es OWASP ZAP (en la imagen)**
  - https://www.zaproxy.org/
  - Listado de estas herramientas
    - https://www.comparitech.com/net-admin/dast-tools/
    - Algunas de ellas hacen más que solo DAST
- **DAST sencillo con ZAP manual**
  - Sencillo y recomendable para tu **TFG** ☺
  - https://medium.com/@renbe/dast-with-owasp-zap-89fb5e1fa243
- **DAST integrado automatizado en un pipeline**
  - https://www.everable.com/blog/automated-dast-in-ci/cd-pipeline-using-owasp-zap-a-comprehensive-guide



➢ Ejercicio **L10B3_DAST**

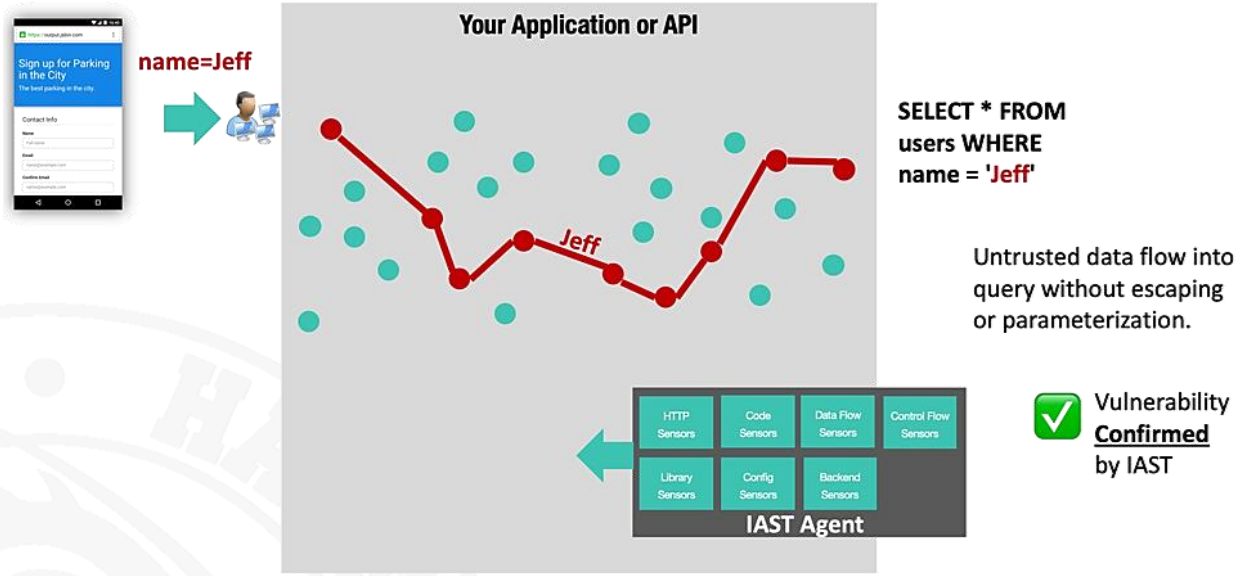
Herramientas IAST (Interactive Application Security Testing)

Seguridad de Sistemas Informáticos

- **Instrumentan el software para evaluar su rendimiento y detectar vulnerabilidades**
  - **Enfoque similar al de un agente**: Agentes y los sensores analizan continuamente el funcionamiento de la aplicación
    - **Durante las pruebas automatizadas y/o manuales o en ejecución…**
  - El proceso y la retroalimentación se hacen en tiempo real en el IDE, en el entorno de CI, en los procesos de garantía de calidad, o en producción
  - Los sensores tienen acceso al código fuente completo, flujo de datos y flujo de control, datos de configuración del sistema, componentes web y datos de conexión al back-end
    - Mayor cobertura y salida más precisa que las SAST y las DAST
  - **Ejemplos**: https://owasp.org/www-community/Source_Code_Analysis_Tools
- **Ventajas**
  - Los problemas potenciales se pueden detectar antes (“shift left”), minimizando costes y retrasos
  - Muestran datos más completos de líneas de código con problemas, y así es más fácil arreglarlos
- **Inconvenientes**
  - Pueden ralentizar el funcionamiento de la aplicación (por la instrumentación del código)

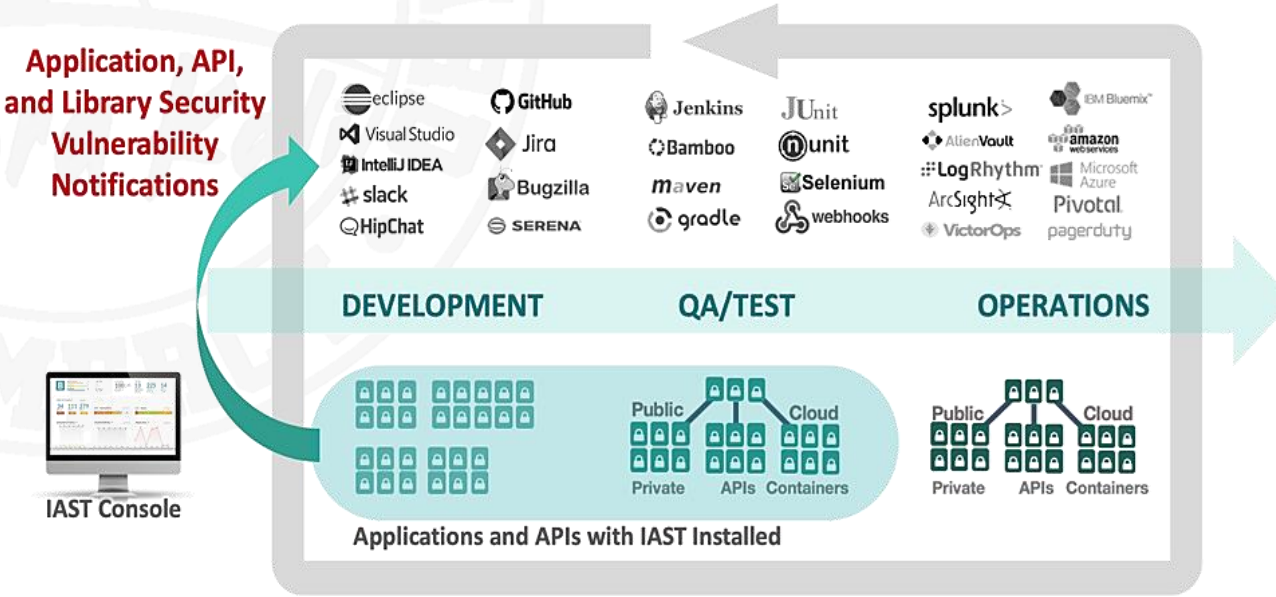
HERRAMIENTAS IAST

Seguridad de Sistemas Informáticos



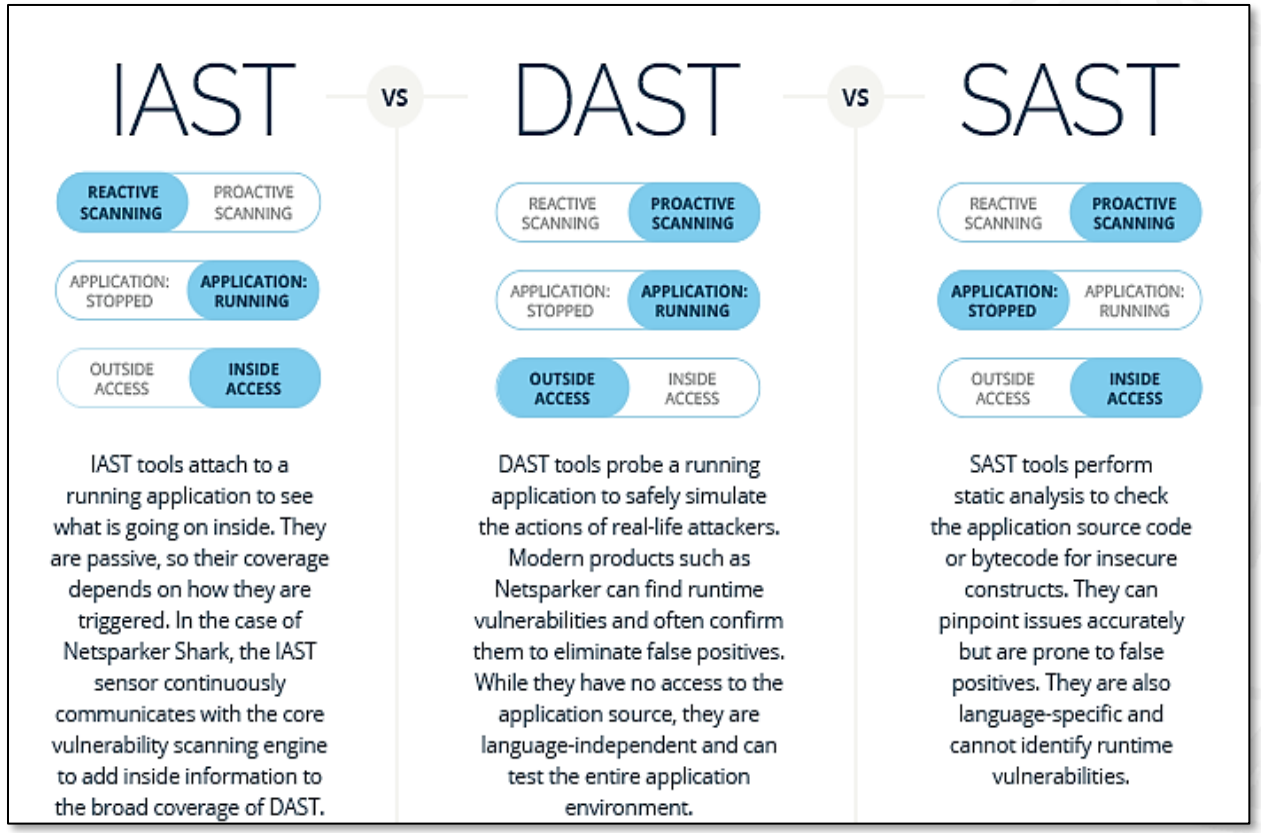
- **Con una IAST el proceso de construcción, test y despliegue típico no cambia**
  - Lo que pasa es que está constantemente "vigilando" la aplicación en segundo plano
- **Y notifica al elemento de la cadena de desarrollo correspondiente cuando se introduce código o librerías peligrosas**

Una herramienta IAST está integrada en tu aplicación y detecta entradas y acciones peligrosas, informando a otros elementos del pipeline. Fuente: https://dzone.com/refcardz/introduction-to-iast



HERRAMIENTAS IAST

- **Listado de estas herramientas y más de las categorías anteriores**
  - https://owasp.org/www-community/Source_Code_Analysis_Tools



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

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

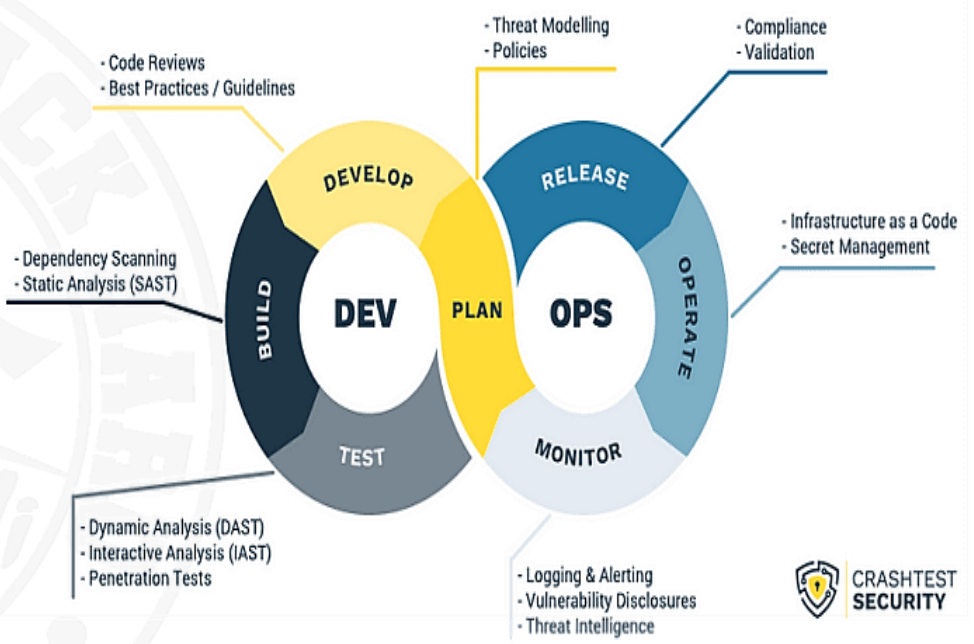
***Iconos por Bing Chat AI**

Herramientas de seguridad en DevOps

- **Las ventajas de integrar herramientas de seguridad en el pipeline CI/CD son mayores cuanto antes se haga en el proceso**
  - **¡"shift left" también se aplica aquí!**
  - Es vital que **los desarrolladores se sientan cómodos** usándolas como parte de su trabajo diario
  - Para asegurar una integración, entrega y despliegue continuos
  - Sino generará rechazo en los equipos de DevOps, y no las usarán adecuadamente
- **Las herramientas también deben ofrecer**
  - La capacidad de ser invocadas desde la línea de comandos
  - La integración con el panel de métricas que se use en el proyecto
  - El seguimiento de los defectos encontrados
  - La capacidad de interrumpir la compilación y enviar notificaciones automáticas por e-mail
- **Los lenguajes, los frameworks y las API cambian constantemente**
  - Las herramientas deben ofrecer compatibilidad rápida con las nuevas versiones de las mismas

Otras herramientas SecDevOps

- Otras herramientas
  - 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
- No podemos tampoco descartar que el propio framework de desarrollo que estamos usando incluya parte de estas características
  - Ej. Node.js, (SDI) Audita de manera automática la seguridad de las dependencias
  - https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities



El uso de estas herramientas en SecDevOps es cíclico.
Fuente: https://sudip-says-hi.medium.com/an-overview-of-security-testing-tools-in-devops-d955687f2913

¿Crees que lo has entendido todo? ¡Autoevalúate!

- **Eres capaz de hacer lo siguiente**
  - *Entender que el desarrollo seguro requiere la integración de herramientas de testing de seguridad en la cadena de desarrollo*
  - *Entender para que sirve una herramienta SCA y el tipo de problemas que previene*
    - *Especialmente su labor crítica a la hora de identificar y prevenir CVEs de software de terceros que estemos usando*
  - *Entender para que sirve una herramienta SAST y el tipo de problemas que previene*
    - *Y que son capaces de decirnos la línea de código exacta donde detectó un problema*
    - *Además de cómo poner en marcha un SAST (y un SCA) muy fácilmente en GitHub*
  - *Entender para que sirve una herramienta DAST y el tipo de problemas que previene*
    - *Y que al final tratan a la aplicación como una "caja negra" en ejecución*
  - *Entender para que sirve una herramienta IAST y el tipo de problemas que previene*

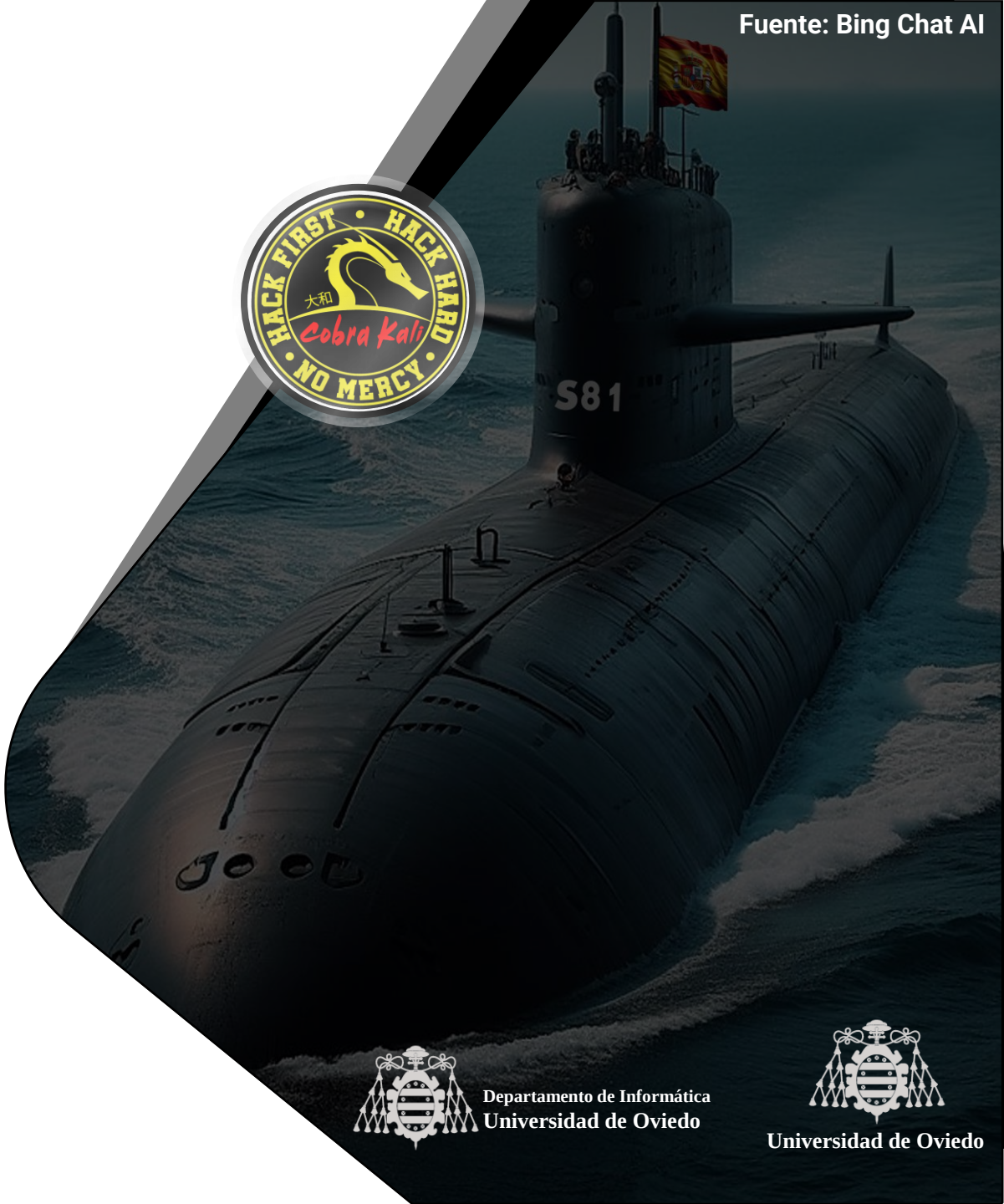
< Ir al Índice

Despliegue y Mantenimiento Seguros

Aspectos a tener en cuenta una vez la aplicación esté en producción

*Iconos por Bing Chat AI

Fuente: Bing Chat AI



Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

¿QUÉ TE VAS A ENCONTRAR EN ESTE BLOQUE?

- **En este bloque aprenderás**
  - Qué son las herramientas **WAF**
  - Qué son las herramientas **RASP**
  - Qué son las herramientas **UEBA**

SEGURIDAD EN EL DESPLIEGUE Y MANTENIMIENTO

- **Para mantener nuestra aplicación desplegada tenemos que tener en cuenta**
  - Lo que aprendisteis en ASR sobre **administración** de SSOO y nuestros **Temas 3, 4 y 5**
    - Especialmente los CIS Benchmarks
  - **Monitorización, log** y alertas de seguridad continuas (**Tema 5**)
  - Tener procedimientos de **respuesta a incidentes**
  - Si creamos un API, **usar un API Gateway**
    - Servicio administrado que facilita crear, publicar, mantener, monitorizar y proteger un API
    - Son la "puerta de entrada" para que las aplicaciones accedan a datos, la lógica de negocio o la funcionalidad de los servicios de backend. Ej.: https://aws.amazon.com/es/api-gateway/

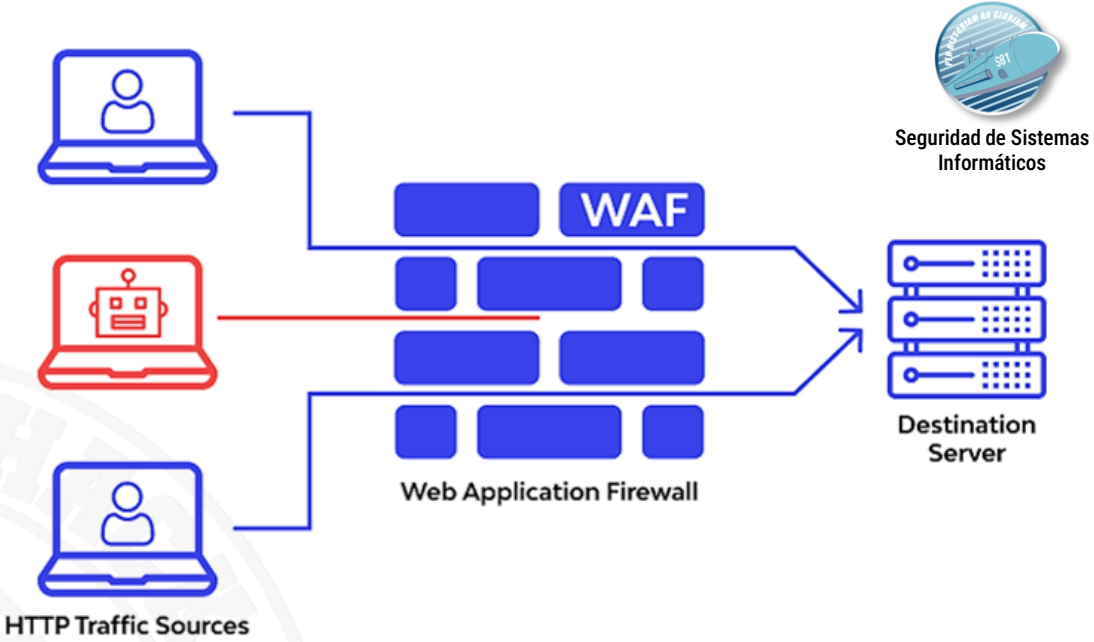
  - Usar aplicaciones de **protección activa** del software una vez desplegado: WAFs, RASP y UEBAs
  - Si hacemos un **despliegue en nube** tengamos que usar herramientas AST compatibles
    - Una herramienta SCA muy buena para entornos cloud es Snyk (https://snyk.io/)
      - https://snyk.io/blog/what-is-software-composition-analysis-sca-and-does-my-company-need-it/
    - Fortify WebInspect: combina DAST y SAST también en entornos de despliegue cloud
      - https://www.microfocus.com/es-es/products/webinspect-dynamic-analysis-dast/overview

*Iconos por Bing Chat AI

Web Application Firewalls (WAFs) (mod_security)

- **Módulos de los principales servidores web que protegen automáticamente cualquier aplicación web que alojen contra una amplia gama de ataques a aplicaciones: no despliegues sin uno**
  - Incorpora una capa de protección adicional para cualquier aplicación web (defense in depth)
  - `mod_security` es probablemente el WAF más conocido (visto en el **Lab 10**)
  - Necesitan reglas adaptadas a cada aplicación; las más conocidas son CSR (Core Security Rules)
    - https://modsecurity.org/crs/
  - *¿No puedes desplegar uno?* Alquílalo (Ej.: CloudFlare, https://www.cloudflare.com/es-es/lp/ppc/waf-x/)
- **Una vez activo puede realizar las siguientes operaciones automáticamente**
  - **Protección HTTP**: Detecta infracciones de protocolo
  - **Listas negras de IPs en tiempo real**: Usando servidores de reputación IP de terceros
  - **Detectar malware web, bots, crawlers, escáneres**: API Google Safe Browsing y otras integraciones
  - **Detección DoS y otros ataques web comunes:** XSS, Inyección SQL, CSRF...
  - **Identificación de datos confidenciales y prevenir el servir código fuente accidentalmente** (Ej. tarjetas de crédito)
  - **Detecta y oculta los mensajes de error y enmascara la identidad del servidor**
  - ... *(muchas más características)*

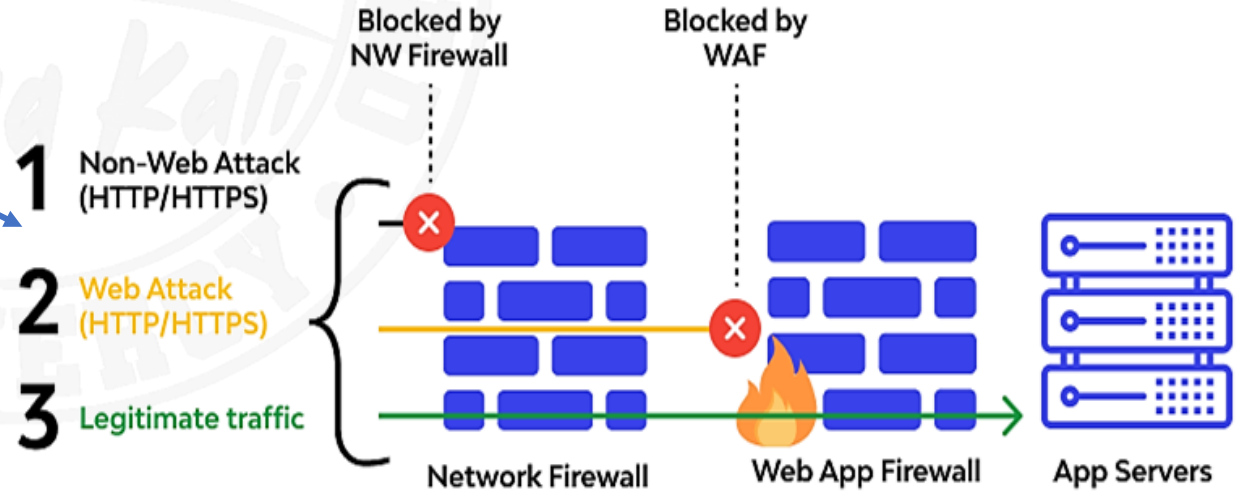
Web Application Firewalls



- Es importante que distingas
  - Un firewall (**Tema 5**) **analiza el tráfico de red**, pero no su contenido
    - Origen, destino, puerto...
  - Un WAF **analiza el contenido del mismo**, lo entiende y lo procesa
    - Por eso puede encontrar peticiones maliciosas en el tráfico de red
    - Suele encontrarse en un **reverse proxy** que controla el acceso a muchas aplicaciones

No te confundas, aunque ambos lleven la palabra "firewall" no son lo mismo. Es más, se suelen combinar. Fuente: https://www.wallarm.com/what/waf-meaning

- Por tanto, uno es de la capa de red y el otro de la de aplicación
  - **No son antagonistas**, de hecho, deberían ser complementarios
  - ¡Tener un WAF no te exime de tener un firewall ni viceversa!



➢ Ejercicio **L10B1_REVPROXY, L10B2_WAF, L10B2_WAFPROT**

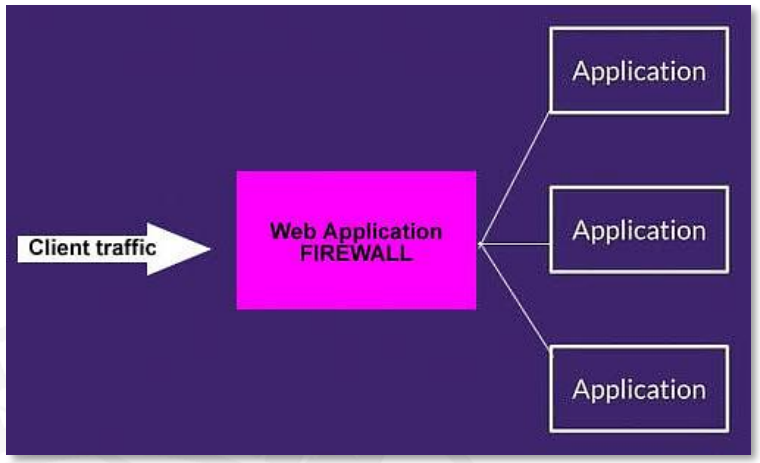
Herramientas RASP (Runtime Application Self-Protection)

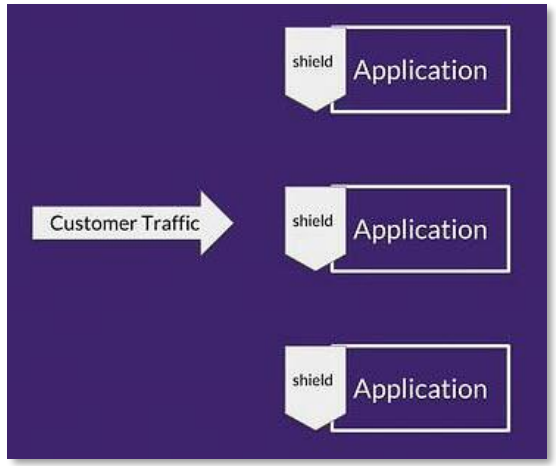
- **Inspeccionan el comportamiento de la aplicación y su contexto**
  - Capturan todas las peticiones para ver si son seguras y controlan la validación de solicitudes en la aplicación
  - Pueden funcionar en modo **diagnóstico** (alertas) o **protección** (deteniendo la aplicación y finalizando la sesión)
- **RASP e IAST tienen usos similares, pero RASP no realiza análisis completos**
  - Se ejecuta como parte de la aplicación inspeccionando su tráfico y actividad
  - Ambos informan de los ataques cuando ocurren, pero IAST en pruebas/despliegue y que RASP en producción
- **Ventajas**
  - Complementa a SAST y DAST siendo una capa adicional de protección en producción (defense in depth)
  - Permiten examinar y controlar entradas no previstas
  - Pueden responder rápidamente a un ataque al proporcionar un análisis exhaustivo y localizar vulnerabilidades
- **Inconvenientes**
  - Se ejecutan en el servidor de aplicaciones, pudiendo tener un impacto negativo en el rendimiento
  - **Dan una falsa sensación de seguridad**: los problemas detectados necesitan repararse y parar la aplicación
    - No son un sustituto de las pruebas de seguridad porque no dan protección completa (exceso de confianza)

HERRAMIENTAS RASP

Seguridad de Sistemas Informáticos

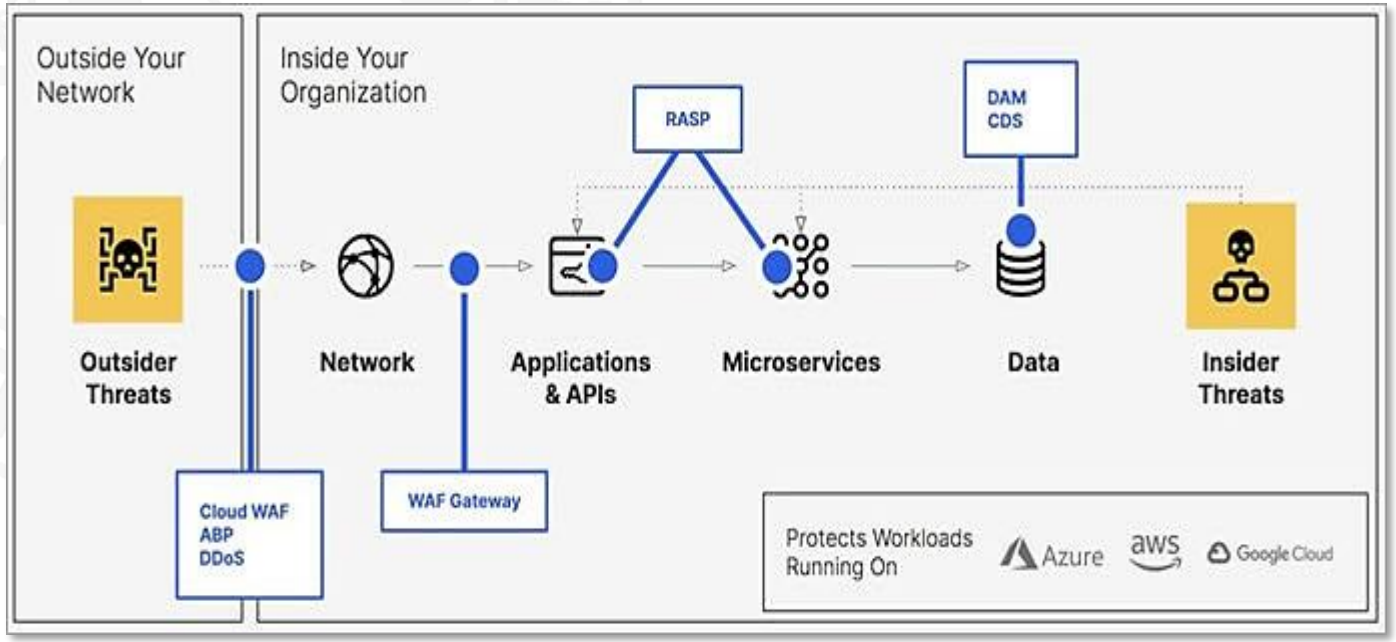
- **Una RASP es una capa que se sitúa en la aplicación principal**
  - Monitorizando todo el tráfico de entrada
  - Tanto al servidor como al API de la aplicación
- **Si se detecta cualquier tipo de vector de ataque, activa las protecciones que tiene embebidas**
  - Todo ello en tiempo de ejecución
- **Esto protege a la aplicación de ataques que puedan venir después**
  - **Funciona en tres modos**: Apagado (off), solo vigilancia (monitor) y bloqueo (block)





Diferencia entre la aproximación de un WAF y de un RASP. Fuente: https://www.softwaretestinghelp.com/rasp-tutorial/

Donde opera un RASP en una aplicación en ejecución. Fuente: https://dzone.com/refcardz/introduction-to-rasp



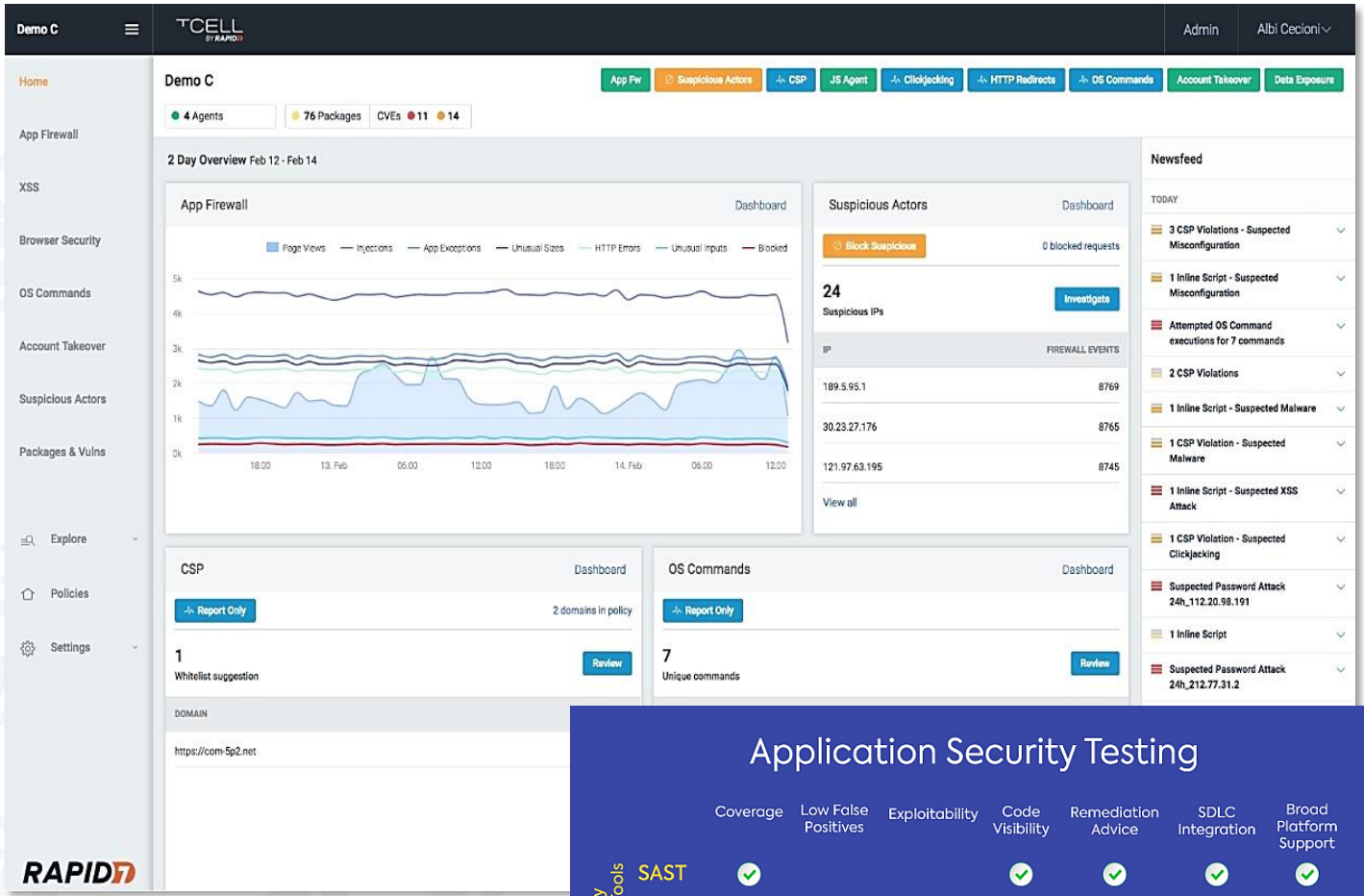
Herramientas RASP (Runtime Application Self-Protection)

Seguridad de Sistemas Informáticos

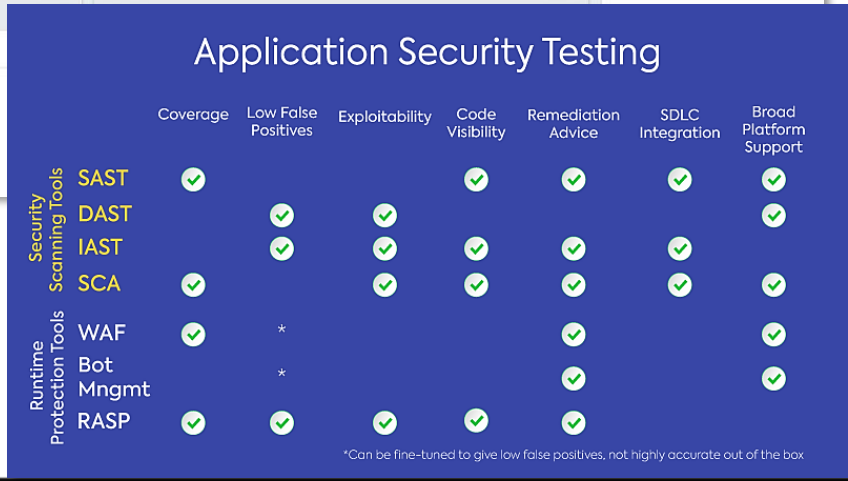
- Ejemplos de herramientas RASP
  - FreeRASP: https://github.com/talsec/Free-RASP-Community
  - Más ejemplos: https://www.g2.com/categories/runtime-application-self-protection-rasp

WAF	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

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



Fuente: 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)

Seguridad de Sistemas Informáticos

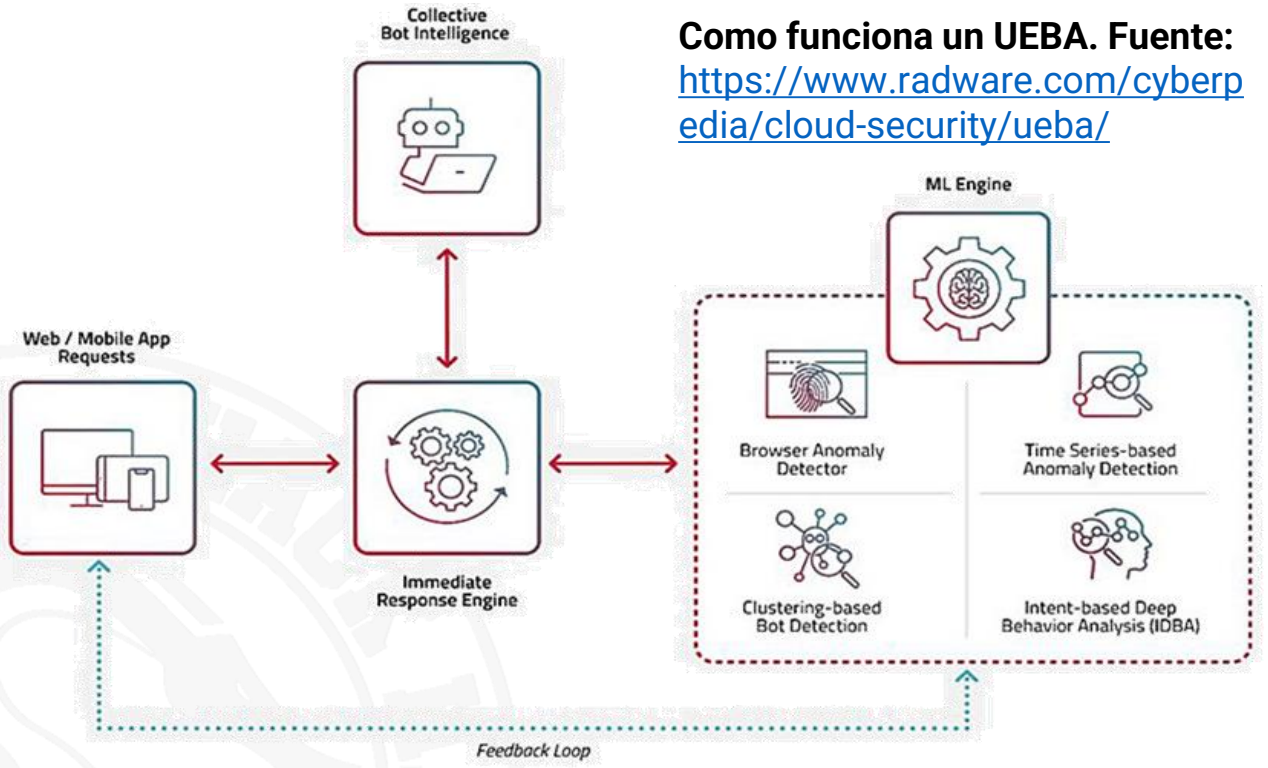
- **Herramientas de detección de anomalías basadas en IA**
  - **Analizan el comportamiento** de usuarios, servidores, cuentas, portátiles, aplicaciones, etc.
    - ¡Todo lo que esté conectado a la red de una organización!
  - Se usan para detectar amenazas, fallos externos e intrusos
    - Inside jobs, rogue users...suponen un riesgo mayor cada día
  - **Aprenden** lo que hacen habitualmente para fijar lo que se considera un **comportamiento "normal"**
    - *¿Desde dónde se conectan? ¿A qué servidores, archivos y aplicaciones acceden? ¿Desde qué dispositivos? ...*
  - Modelan un comportamiento "habitual" para distinguirlo de los "anómalos"
    - Cuando ocurre algo inusual el UEBA lo detectará y alertará a los responsables
- **Ejemplos son: IBM Security Qradar, Idaptive Next-Gen Access, ActivTrak, InsightIDR, Teramind o Exabeam Security Management Platform**
  - Un listado más completo: https://www.g2.com/categories/user-and-entity-behavior-analytics-ueba/free
- **Kassandra es un proyecto de investigación que sigue esta filosofía, desarrollado por Alba Cotarelo Tuñón, egresada de esta escuela**
  - https://github.com/Egida-Kassandra/kassandra

UEBA

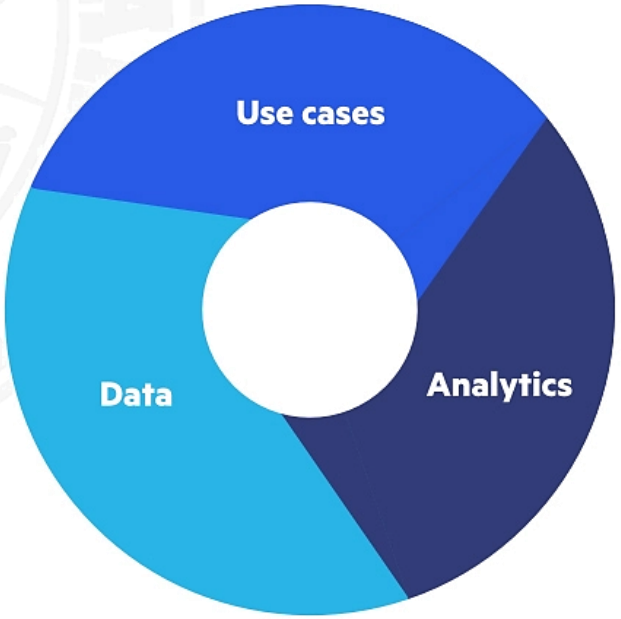
Seguridad de Sistemas Informáticos

- **Un UEBA es un sistema de Machine Learning (ML) que aprende de las acciones de los usuarios**
- **Analiza muchas fuentes de datos para identificar casos de uso conocidos**
  - Usuario malicioso o comprometido, etc.
- **Los datos incluyen fuentes externas e internas de la aplicación**
  - Ej.: Log de la aplicación
  - Es una de las aplicaciones de la IA más modernas e interesantes para ciberseguridad

Como funciona un UEBA. Fuente: https://www.radware.com/cyberpedia/cloud-security/ueba/



Datos que usa un UEBA. Fuente: 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

¿CREES QUE LO HAS ENTENDIDO TODO? ¡AUTOEVALÚATE!

?

- **Eres capaz de hacer lo siguiente**
  - *Comprender el concepto de API Gateway*
  - *Entender qué es un WAF y cómo protege en el contexto del desarrollo de una aplicación web*
  - *Entender qué es un RASP y cómo protege a una aplicación*
    - *Y que es un añadido que se sitúa dentro de la propia aplicación*
  - *Entender qué es un UEBA y cómo protege a una aplicación*
    - *Y que realmente es una forma de detectar desviaciones en el comportamiento de usuarios y entidades*

Referencias

- **Para leer más sobre DevSecOps**
  - https://blog.pentesteracademy.com/devsecops-learning-path-integrating-security-with-devops-1cc03670552f?gi=b60a95ce4fcb
- **Para aprender una aproximación alternativa a DevSecOps, es recomendable leer los OWASP Proactive Controls**
  - https://owasp.org/www-project-proactive-controls/
- **Para trabajar con un pipeline Jenkins (usado en Arquitectura del Software, ASW) con las distintas herramientas de testing vistas**
  - Instalación de Jenkins en un contenedor Docker (adecuado para pruebas):
    - https://www.jenkins.io/solutions/docker/
  - Plugin Fortify para Jenkins (**SCA**)
    - https://plugins.jenkins.io/fortify/
  - Plugin Sonarqube para Jenkins (**SAST**) (también para otros productos similares):
    - https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-jenkins/
  - **DAST** con OWASP ZAP y Jenkins
    - https://digitalvarys.com/devsecops-dynamic-analysis-dast-with-owasp-zap-and-jenkins/

TEMA 6
INTRODUCCIÓN A LA SEGURIDAD DE APLICACIONES

Fuente: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo