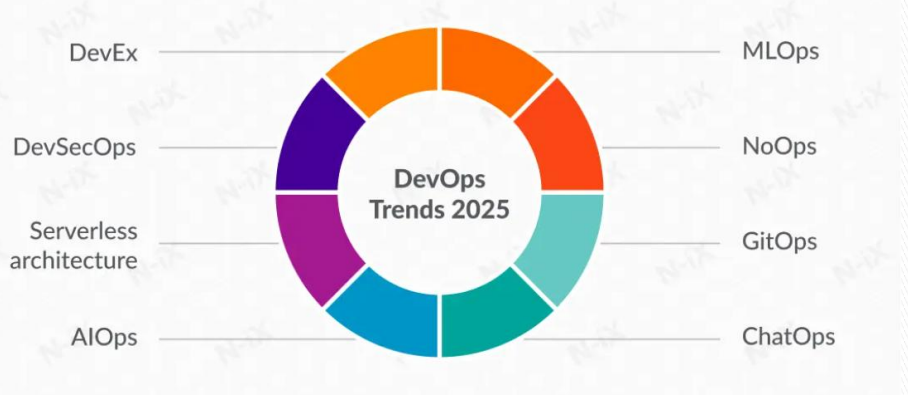
Desarrollo seguro basado en OWASP

1

Introducción

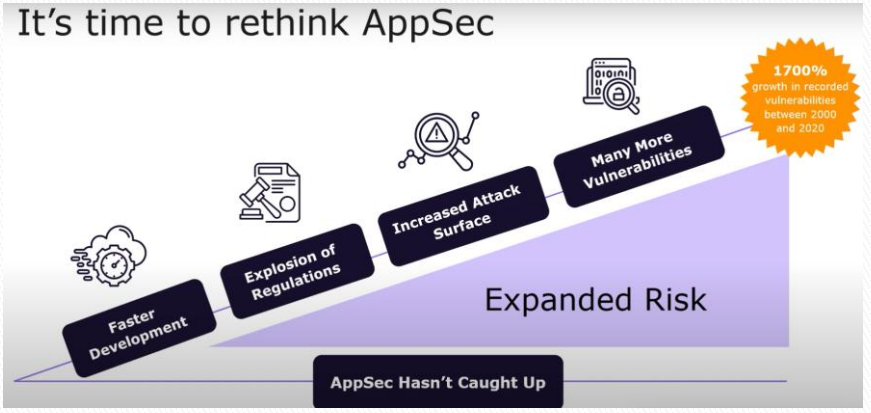
2

Tendencias en Desarrollo de Aplicaciones



4

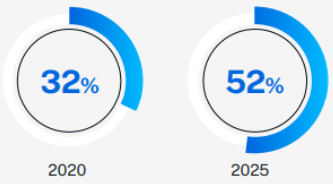
Tendencias en Desarrollo de Aplicaciones



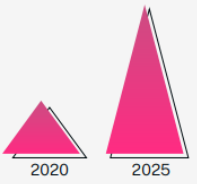
5

Seguridad de Aplicaciones

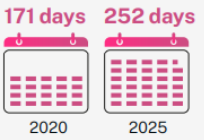
Good news first, the percentage of apps passing the OWASP Top 10 has increased 63% in 5 years (from 32% to 52%)



Now the bad news... the percentage of apps with high severity flaws has increased by 181%...



...and the average number of days to fix flaws has increased 47%.



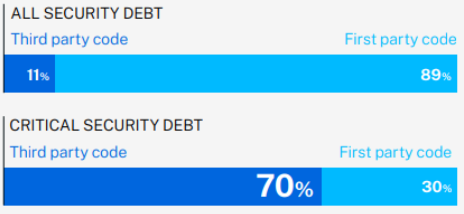
State of Software Security – Veracode 2025

6

Seguridad de Aplicaciones

Half of organizations have critical security debt (high severity, high exploitability)...

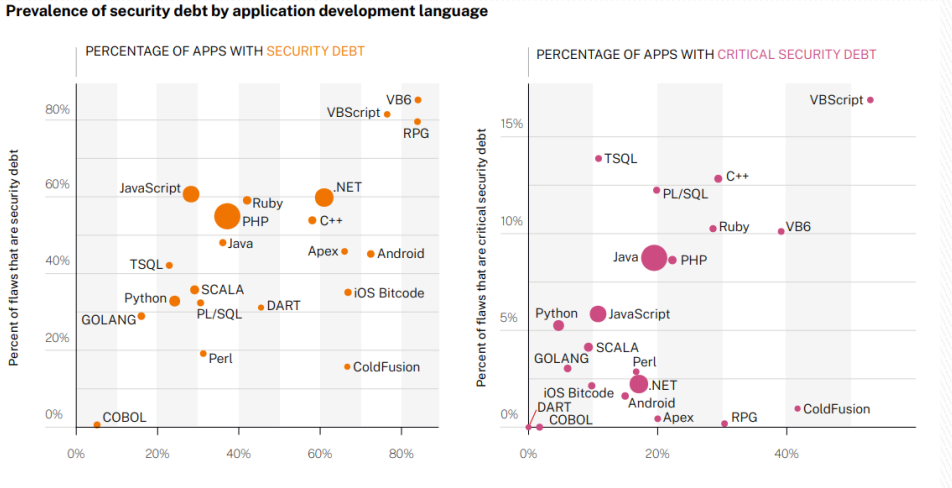
...and 70% of it comes from third party code and the software supply chain.



State of Software Security – Veracode 2025

7

Seguridad de Aplicaciones

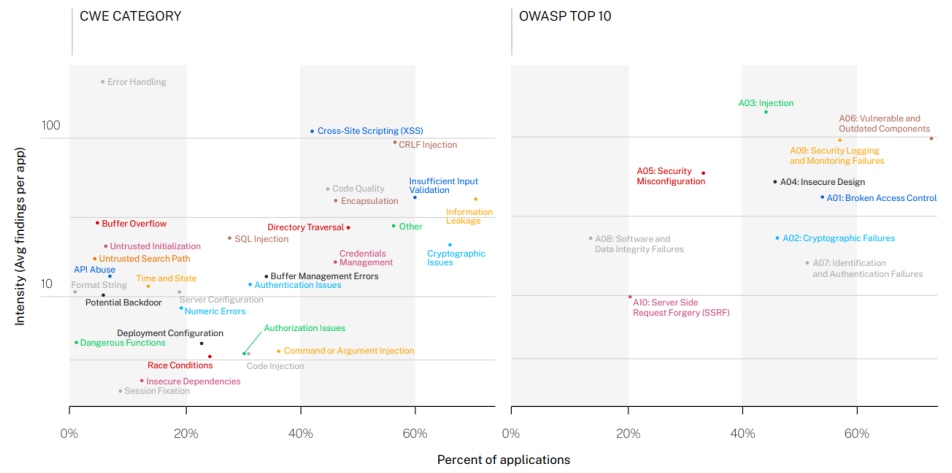


State of Software Security – Veracode 2025

8

8

Seguridad de Aplicaciones



State of Software Security – Veracode 2025

9

9

Seguridad de Aplicaciones

State of Software Security – Veracode 2025

10

Seguridad de Aplicaciones Web

11

Ataques en la actualidad

Chinese Hackers Exploit Ivanti CSA Zero-Days in Attacks on French Government, Telecoms

Jul 03, 2025 Ravie Lakshmanan

Ataques en la actualidad

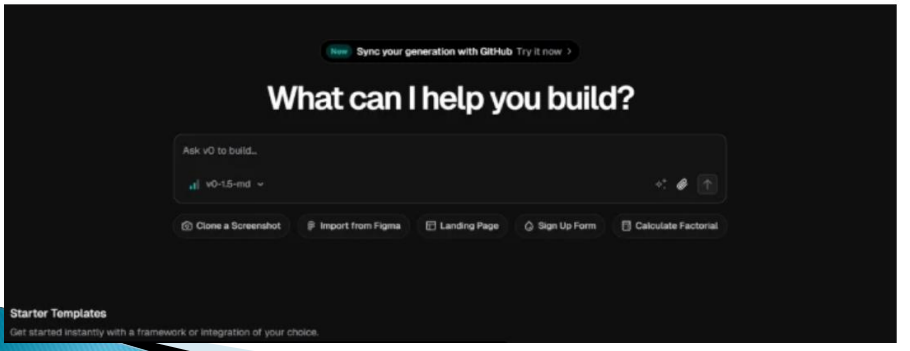
North Korean Hackers Target Web3 with Nim Malware and Use ClickFix in BabyShark Campaign

Jul 02, 2025 Ravie Lakshmanan

Ataques en la actualidad

Vercel's v0 AI Tool Weaponized by Cybercriminals to Rapidly Create Fake Login Pages at Scale

Jul 02, 2025 Ravie Lakshmanan



Ataques en la actualidad

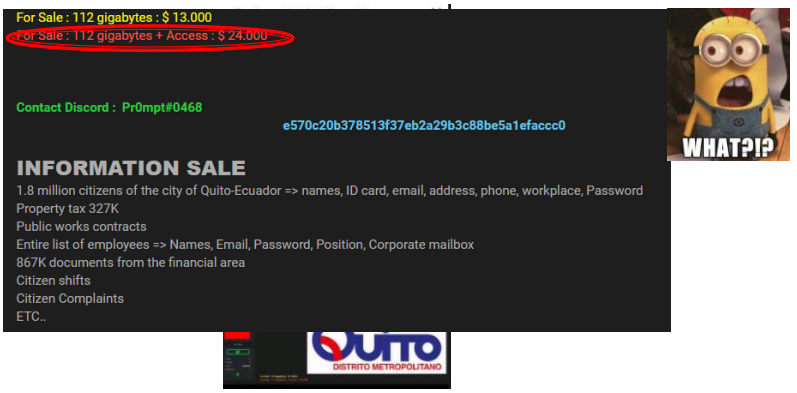
A hacker is threatening to leak 106GB of data allegedly stolen from Spanish telecommunications company Telefónica in a breach that the company did not acknowledge.

The threat actor has leaked a 2.6GB archive that unpacks into five gigabytes of data with a little over 20,000 files to prove that the breach occurred.

Index of /downloads/Telefonica_Partial

Name	Last modified	Size	Description
Parent Directory		-	
NOTE.txt	2025-07-03 23:59	283	
Partial_1.zip	2025-07-04 01:33	2.6G	

Costo de Ataque



16

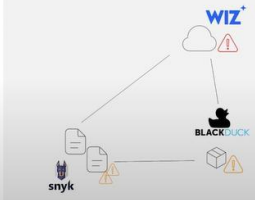
¿En que afecta al Desarrollador?

17

¿En que afecta al Desarrollador?

Incomplete view of risk

Lacking a clear, cohesive & current view of application risk

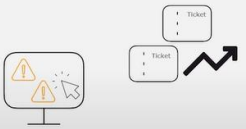


Too many vulnerabilities

Volume of findings makes it near-impossible to know what really matters

Manual & inefficient

Too much manual work slows progress, creates friction



18

18

Amenazas – Seguridad de las App.

1.- Script Kiddie

2.- Hacktivist

3.- Hacker

4.- Cyber Criminal

5.- Advanced Persistent Threat

20

20

Seguridad en el Desarrollo de Aplicaciones

Confidencialidad
Integridad
Disponibilidad

Conceptos guías que ayudan a tomar decisiones sobre seguridad

Técnicas de codificación segura

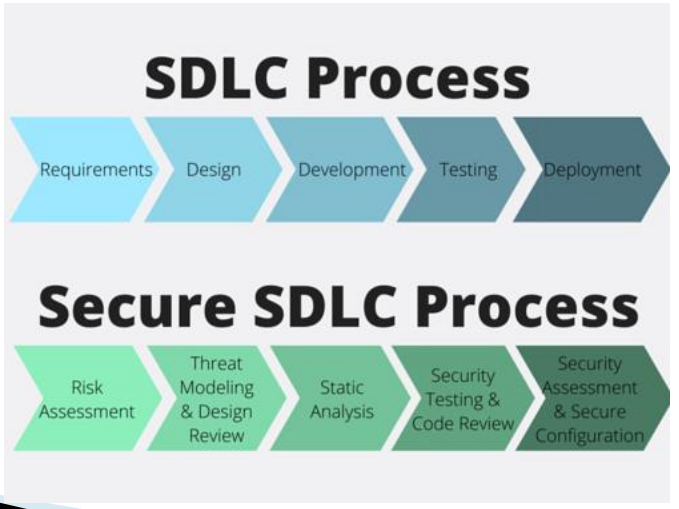
Objetivos –Seguridad de la Información

La información es confiable y solo ha sido alterada por personal autorizado

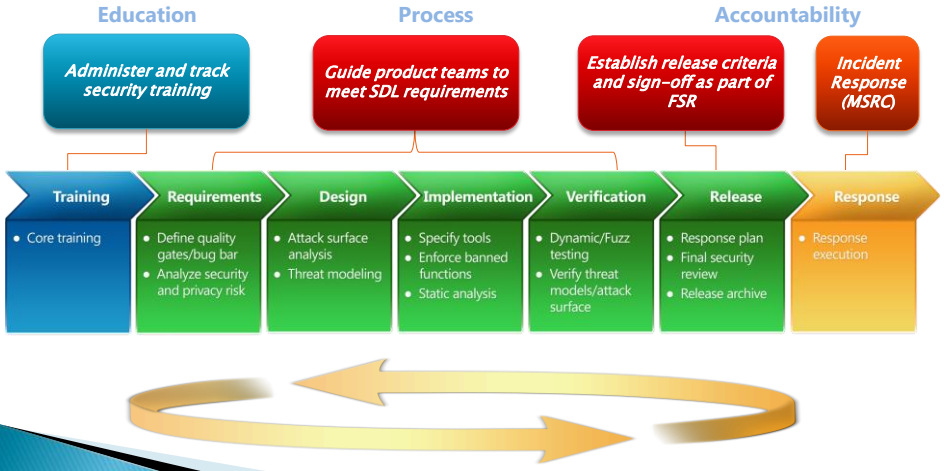
Solo accede a la información quienes están autorizados

Acceso a la información cuando sea requerido

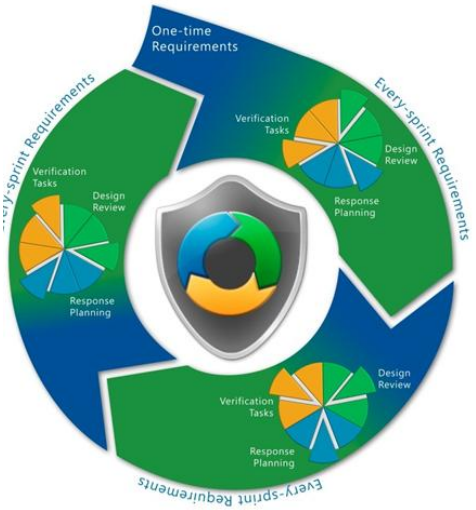
Security Development Lifecycle (SDL)



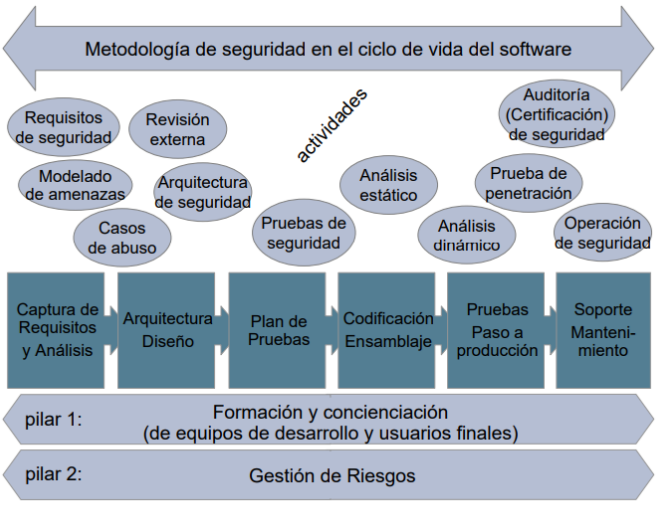
Security Development Lifecycle (SDL)



(SDL) – Metodologías Agiles



Security Development Lifecycle (SDL)



(*) Adaptado de *Software Security: Building Security In*. Gary McGraw. Addison Wesley, 200(

Requisitos de Seguridad – Checklist

1. Requerimientos de Seguridad Principales

- **Confidencialidad**: Controles para asegurar que la confidencialidad esté garantizada cuando los datos está en reposo, en tránsito y también cuando se procesa. *Ejemplo: algoritmo de cifrado de contraseña, de canal de comunicación*)
- **Integridad**: Controles firmas digitales ayudan a garantizar la integridad. *Ejemplo: que documentos requieren ser firmados digitalmente*
- **Disponibilidad**: Controles de protección contra la destrucción no deseada o la interrupción del servicio. *Ejemplo: definición de SLA, RTO*

Requisitos de Seguridad – Checklist

2. Requerimientos de Seguridad Principales

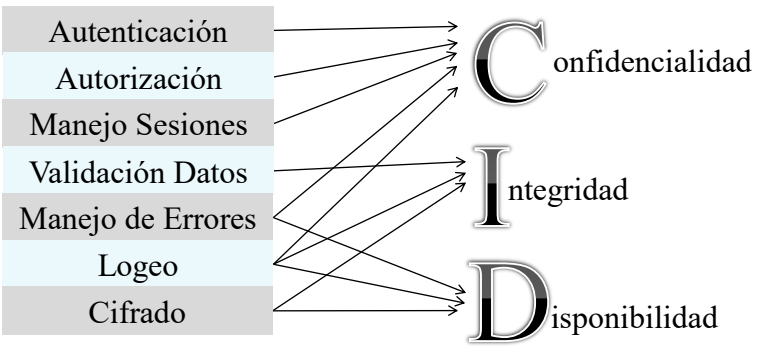
- **Autenticación**: Requisitos de autenticación, como qué método usar. *Ejemplo: Active Directory, SSO, Biométrico*
- **Autorización**: Permisos que se asignarán a todas las entidades autenticadas. *Ejemplo: Roles o Reglas de Negocios.*
- **Registro y Auditorias**: Requisitos para detectar cuándo un usuario no autorizado realiza un cambio o cuando un usuario autorizado realiza un cambio no autorizado. *Ejemplo: Accesos, transaccionales, parámetros de configuración*

Requisitos de Seguridad – Checklist

3. Requerimientos de Seguridad de Aplicación
   - Manejo de Sesiones.
   - Manejo de Errores.
   - Administración de Configuración.

4. Requerimientos de Seguridad de la Operación
   - Ambiente de despliegue asegurados.
   - Archiving, políticas de retención de acuerdo lo definido por el negocio.
   - Ofuscamiento de código
   - Regulaciones internacionales, PCI-DSS, ISO, etc.

Mecanismos para Alcanzar Objetivos



Tecnologías involucradas en el desarrollo de software

SEGTIUM
INGENIERIA EN CIBERSEGURIDAD

31

Lenguajes de Programación:

- Java, Python, C++, C#: Lenguajes de propósito general.
- JavaScript: Utilizado principalmente para desarrollo web.
- Ruby, PHP: Utilizados para desarrollo web y scripting.
- Swift, Objective-C: Para desarrollo de aplicaciones iOS.
- Kotlin: Lenguaje oficial para el desarrollo de aplicaciones Android.

32

Frameworks:

- Angular, React, Vue.js: Para desarrollo de aplicaciones web frontend.
- Spring, Django, Ruby on Rails: Para desarrollo web backend.
- Express.js: Framework para construir aplicaciones web con Node.js.
- Flask, FastAPI: Para construir aplicaciones web en Python.

33

Frameworks

34

34

Tools

35

35

Bases de Datos:

- MySQL, PostgreSQL, Oracle: Sistemas de gestión de bases de datos relacionales.
- MongoDB, Cassandra, Redis: Bases de datos NoSQL.
- SQLite: Ligera base de datos SQL incorporada.
- Firebase: Base de datos en la nube en tiempo real.

36

Control de Versiones:

- Git: Sistema de control de versiones distribuido ampliamente utilizado.

37

Entornos de Desarrollo Integrados (IDE):

- Visual Studio, IntelliJ, Eclipse: IDEs populares para varios lenguajes de programación.

38

Gestión de Proyectos y Colaboración:

- Jira, Trello, Asana: Herramientas de gestión de proyectos.
- GitLab, GitHub, Bitbucket: Plataformas para alojar y gestionar repositorios de código fuente.

39

Automatización de Construcción y Despliegue:

- Jenkins, Travis CI, CircleCI: Herramientas para la integración continua.
- Docker: Plataforma de contenedores.
- Kubernetes: Orquestación de contenedores.

40

Pruebas y QA (Aseguramiento de la Calidad):

- Selenium, JUnit, pytest: Herramientas de prueba.
- Jenkins, Travis CI: También utilizados para la automatización de pruebas.

41

Arquitecturas y Patrones de Diseño:

- REST, GraphQL: Para el diseño de interfaces de programación de aplicaciones (API).
- Microservicios, Monolitos: Diferentes arquitecturas de software.
- MVC, MVVM: Patrones de diseño arquitectónico.

42

Nube y Virtualización:

- AWS, Azure, Google Cloud Platform: Servicios en la nube.
- VirtualBox, VMware: Plataformas de virtualización.

43

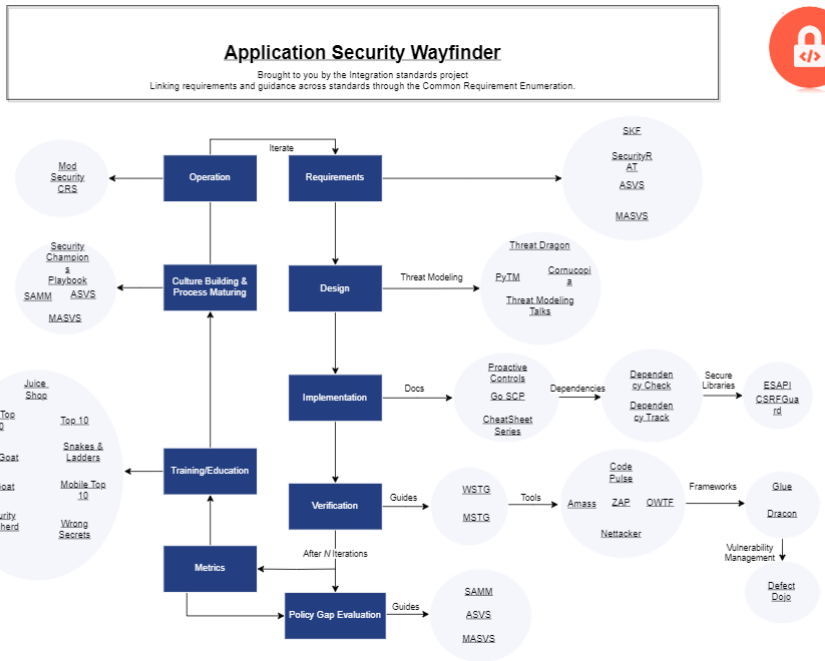
Mecanismos de defensa

44

OWASP

The Open Worldwide Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software. Our programming includes:

- Community-led open source software projects
- Over 250+ local chapters worldwide
- Tens of thousands of members
- Industry-leading educational and training conferences



https://owasp.org/projects/

OWASP Proyectos

47

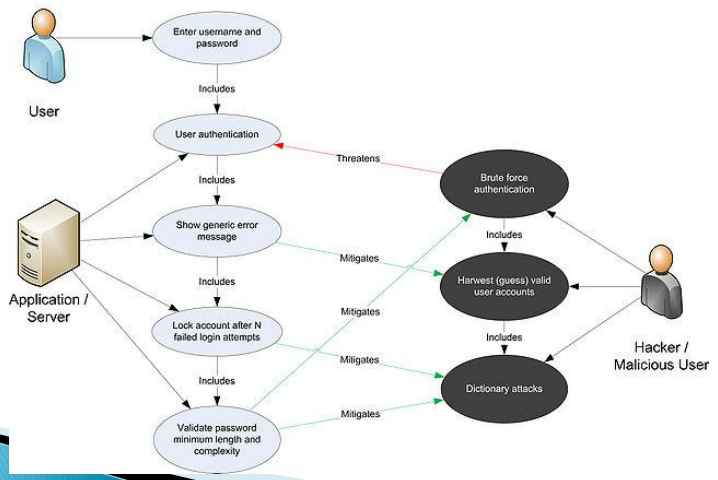
OWASP Proyectos

48

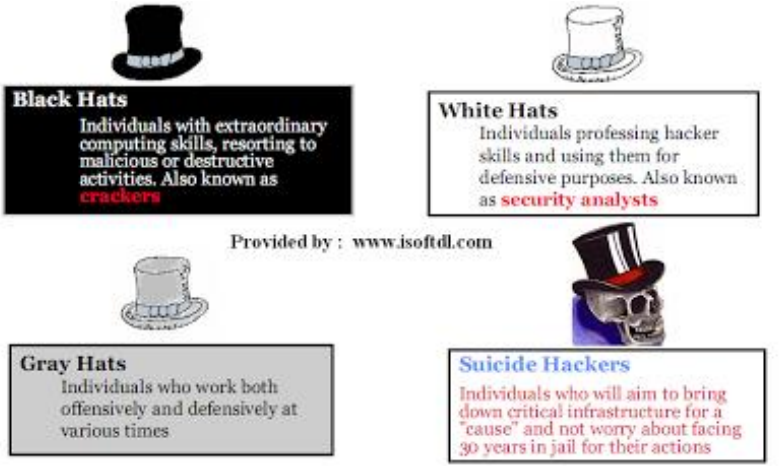
Análisis y Pruebas

Como debo empezar?

- **Casos de Abuso**
- **De donde genero estos casos?**



Mentalidad – Test de Seguridad

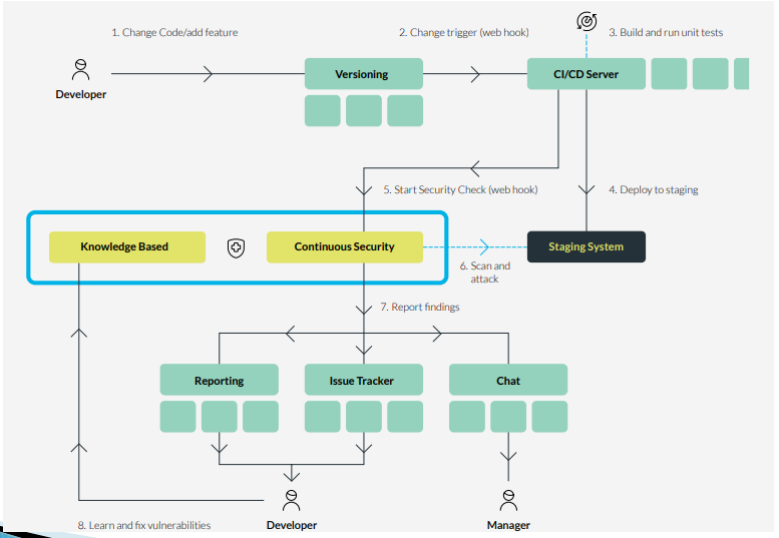


Tipos de Test de Seguridad

- **Análisis estático:** "El análisis estático del código es el proceso de evaluar el software sin ejecutarlo". Una técnica que se aplica directamente sobre el código fuente tal cual, sin transformaciones previas ni cambios.

- **Análisis dinámico:** es un tipo de análisis de software que supone la ejecución del programa y observar su comportamiento, para que el análisis dinámico resulte efectivo se debe ejecutar con los suficientes casos de prueba para producir resultados esperados.

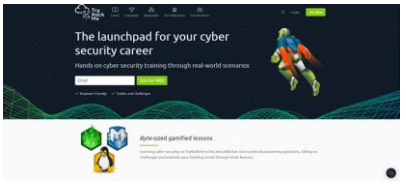
Análisis Estático



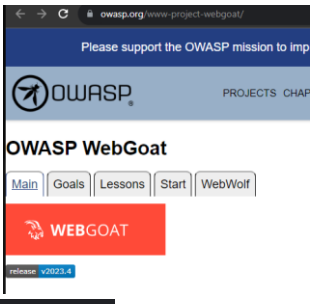
Herramientas de Aprendizaje

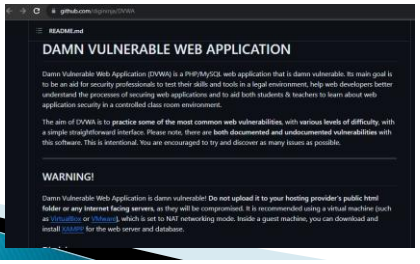
SEGTIUM
INGENIERIA EN CIBERSEGURIDAD

Simuladores – Elearning



Ambientes



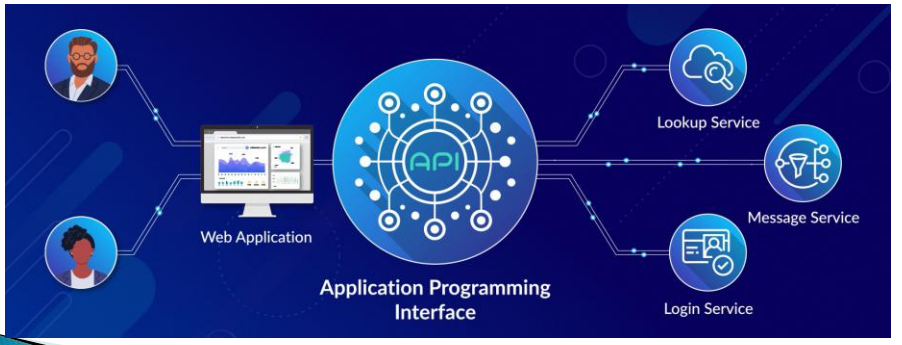


Seguridad Arquitectura de APIs

58

API

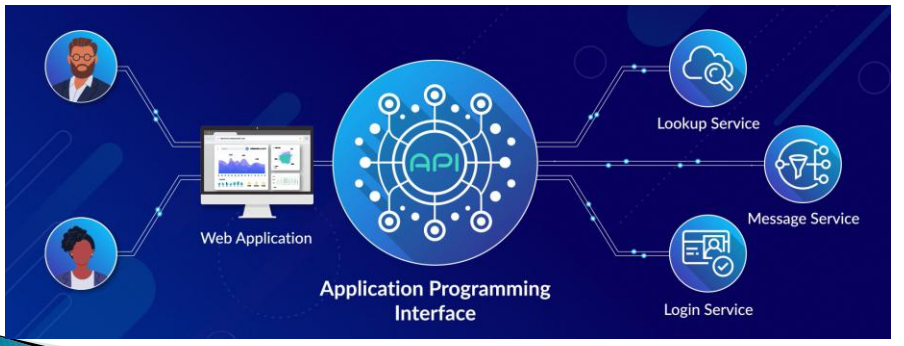
- Interfaz que conecta aplicaciones y servicios.
- Abstrae la complejidad interna del backend.
- Facilita la integración y el desarrollo.



59

API

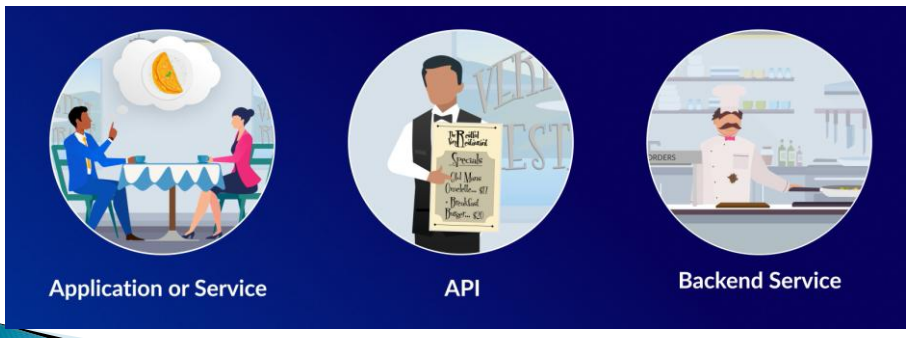
- Interfaz que conecta aplicaciones y servicios.
- Abstrae la complejidad interna del backend.
- Facilita la integración y el desarrollo.



60

API

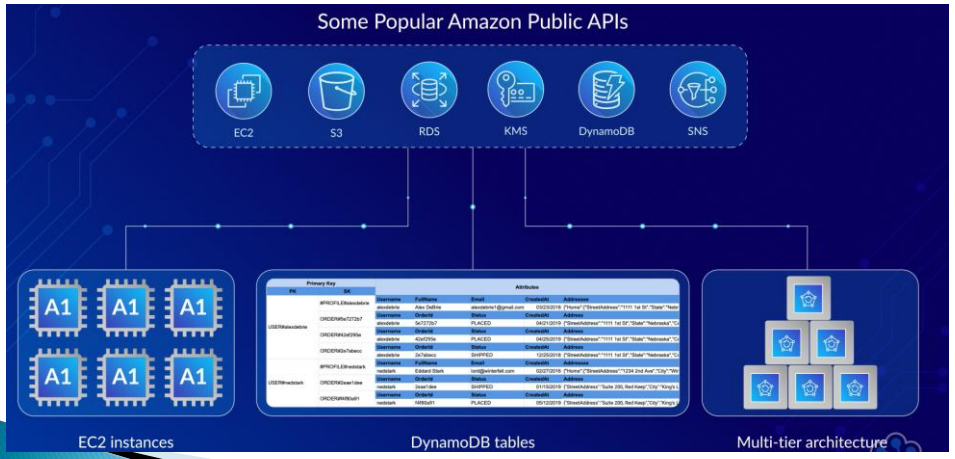
- Restaurante = servicio.
- Mesero y menú = API.
- El cliente pide sin conocer el proceso interno..



61

API

- Implementación Popular



62

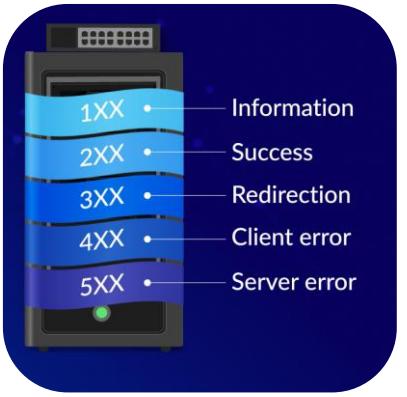
API

- Mecanismos de Comunicación

63

API

- Métodos del protocolo de comunicación



64

API

- Métodos del protocolo de comunicación

65

API – Tipos de Arquitectura

Aspecto	REST	GraphQL	SOAP
Paradigma	Estilo arquitectónico orientado a **recursos**. CRUD a través de métodos HTTP estándar.	Consulta declarativa a un **grafo** tipado; un único endpoint.	Protocolo basado en **mensajes XML** y operaciones definidas en WSDL.
Transporte habitual	HTTP/HTTPS.	HTTP/HTTPS (a veces WebSocket para suscripciones).	HTTP/HTTPS, SMTP, JMS, etc.
Formato de datos	JSON (predilecto), XML, CSV, imágenes…	JSON por defecto (respuesta adaptada al esquema).	XML estricto (SOAP Envelope).
Diseño de endpoint	Múltiples rutas significativas (/users/42/posts).	Un solo endpoint (/graphql) y consulta describe qué traer.	Un único endpoint por servicio con operaciones descritas en WSDL.
Operaciones/ verbs	GET, POST, PUT/PATCH, DELETE…	Query, Mutation, Subscription (definidas en el esquema).	<soap:Body> incluye acción; métodos SOAP Action.

66

API – Tipos de Arquitectura

Aspecto	REST	GraphQL	SOAP
Versionado	URL (/v2/), cabeceras o media-types.	Evolución de esquema sin romper clientes (campos deprecados).	Nuevo WSDL o namespace al cambiar versión.
Caching nativo	Sí, mediante cabeceras HTTP (ETag, Cache-Control).	Limitado (caching a nivel de proxy difícil sin libraries).	No (debe implementarse a nivel de aplicación).
Ventajas	Simple, ampliamente soportado, buen rendimiento con caché.	Recupera exactamente los datos necesarios, reduce chattiness.	Estandarizado para mensajes complejos y seguridad WS-*.
Desafíos	Over/Under-fetching; múltiples llamadas para vistas complejas.	Protección contra consultas anidadas abusivas (DoS); requiere gateway.	Verboso, acoplamiento fuerte, aprendizaje más lento.

67

API – Tipos de Arquitectura

- **Casos de uso típicos:** SOAP hoy se utiliza principalmente en **entornos empresariales** o integraciones B2B donde se requiere un contrato formal y alta seguridad. Por ejemplo, es común en **servicios financieros, pasarelas de pago, sistemas de banca, seguros**, telecomunicaciones, o integraciones con **sistemas heredados (legacy).**
- Su robustez en seguridad y cumplimiento lo hace preferido en operaciones de pago, *booking* de viajes, transacciones legales o financieras donde un acuerdo formal entre proveedor y consumidor de la API es necesario
- En resumen, organizaciones corporativas y gubernamentales a menudo optan por SOAP para garantizar transacciones seguras y confiables.

68

API – Tipos de Arquitectura

- **Casos de uso típicos:** REST es la elección predilecta para **APIs web públicas y servicios orientados a recursos**. Por ejemplo:
- **APIs de gestión de datos** (CRUD de entidades de negocio) expuestas a múltiples consumidores. REST brinda una interface estandarizada, con buena *discoverability* y documentación, ideal para servicios que terceros desarrolladores consumen abiertamente
- **Aplicaciones cliente-servidor sencillas basadas en recursos:** donde se intercambian entidades relativamente simples y no se requieren consultas muy especializadas. Un ejemplo serían aplicaciones de listado de contenidos, blogs, tiendas en línea, etc., en las que las operaciones se ajustan bien a crear/leer/actualizar/borrar objetos estándar
- En general, si se necesita una API fácil de entender, ampliamente compatible, REST suele ser la opción adecuada.

69

API – Tipos de Arquitectura

- **Casos de uso típicos:** GraphSql :

•**APIs para aplicaciones móviles o frontend con requisitos diversos:** GraphQL brilla cuando distintos clientes Muchas empresas (p.ej. Facebook, GitHub) ofrecen APIs GraphQL para permitir a los desarrolladores construir consultas a medida.

•**Sistemas complejos con muchos microservicios o fuentes de datos:** En escenarios donde la información está fragmentada en varios servicios (microservicios, APIs de terceros, bases de datos legacy), GraphQL actúa como **fachada unificadora**.

•Un solo query de GraphQL puede reunir datos que normalmente requerirían orquestar múltiples llamadas REST a diferentes servicios.

Esto es útil en aplicaciones con modelos de datos muy conectados (por ej., redes sociales, sistemas de gestión empresarial) donde reducir la cantidad de llamadas y centralizar la lógica de agregación en el servidor GraphQL mejora la eficiencia global.

70

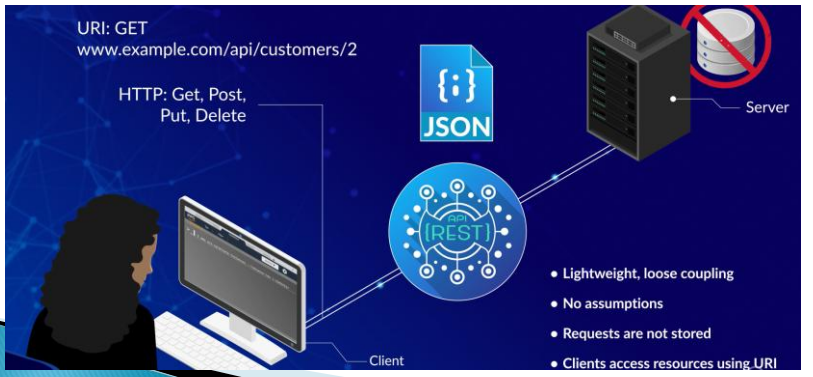
API – REST

- HTTP = protocolo de comunicación a nivel aplicación.
- API HTTP: simplemente usa HTTP para intercambiar mensajes.
- REST = estilo arquitectónico que impone restricciones sobre la comunicación

71

API - REST

- **Hypermedia-Driven**: las interacciones se describen con hipervínculos.
- **Loose Coupling**: cliente y servidor no asumen detalles internos
- **Stateless**: cada petición es independiente; el servidor no guarda contexto.
- **Recursos vía URI**: todo se accede por identificadores únicos.
- El servidor envía una **representación** (JSON, XML…) del estado del recurs



72

API - REST

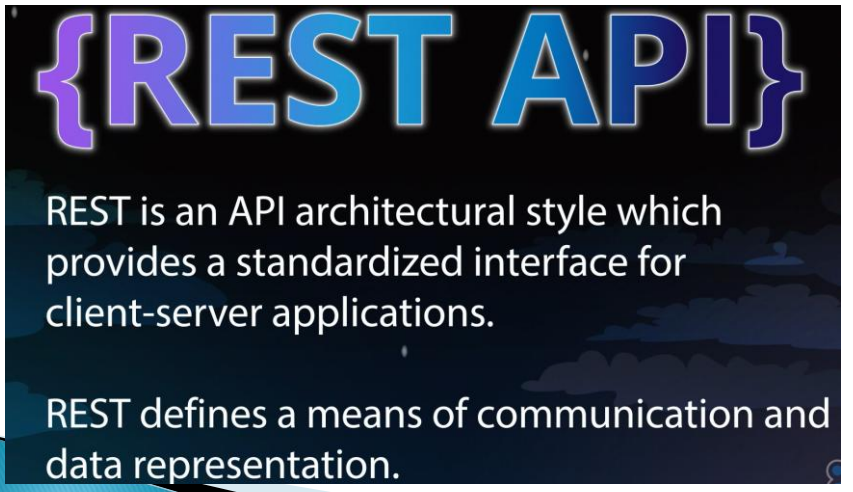
73

API – REST

74

API – REST

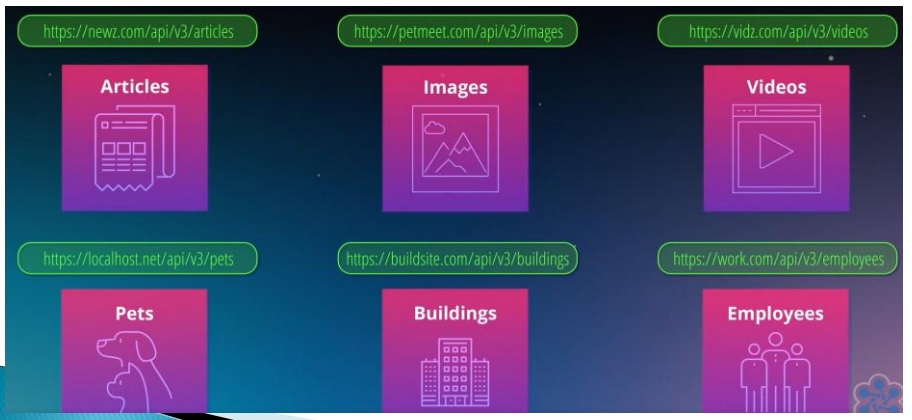
- REST = estilo arquitectónico (2000, Roy Fielding). PS.



75

API – REST

- Todo dato se modela como **recurso**: artículos, imágenes, empleados, etc.
- Cada recurso tiene un **identificador URI/URL**.
- Cliente usa operaciones genéricas (GET, POST, PUT, DELETE).



76

API – REST

- Todo dato se modela como **recurso**: artículos, imágenes, empleados, etc.
- Cada recurso tiene un **identificador URI/URL**.
- Cliente usa operaciones genéricas (GET, POST, PUT, DELETE).

77

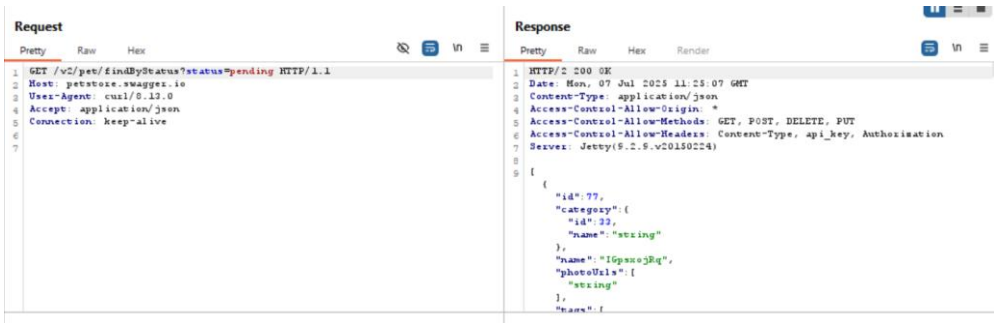
API - REST

- Todo dato se modela como **recurso**: artículos, imágenes, empleados, etc.
- Cada recurso tiene un **identificador URI/URL**.
- Cliente usa operaciones genéricas (GET, POST, PUT, DELETE).

78

Analizando API

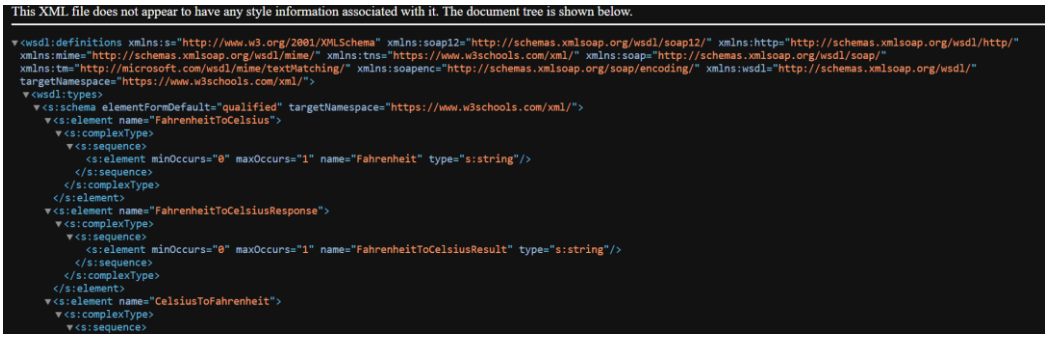
- REST https://petstore.swagger.io/
  - curl.exe -x http://127.0.0.1:8080 -k -H "Accept: application/json" "https://petstore.swagger.io/v2/pet/findByStatus?status=pending"



79

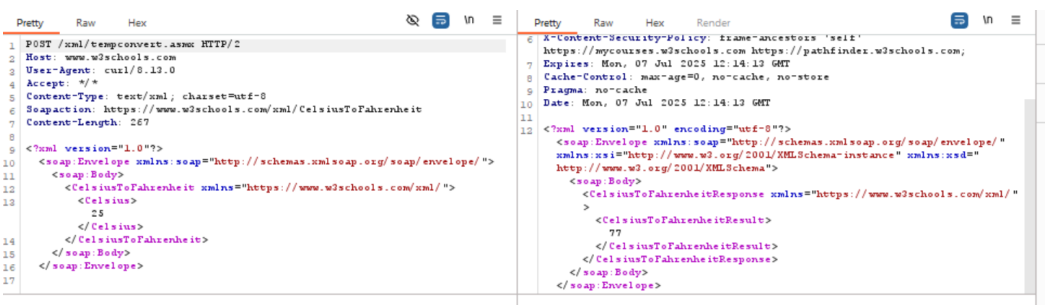
Analizando API

- SOAP https://www.w3schools.com/xml/tempconvert.asmx?WSDL



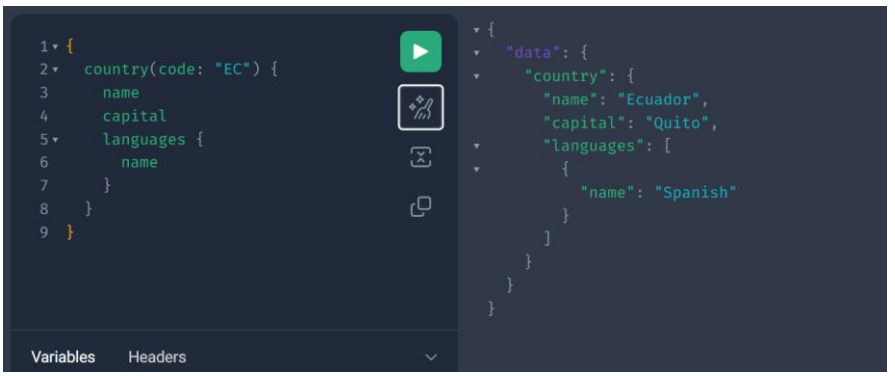
Analizando API

- SOAP https://www.w3schools.com/xml/tempconvert.asmx?WSDL



Analizando API

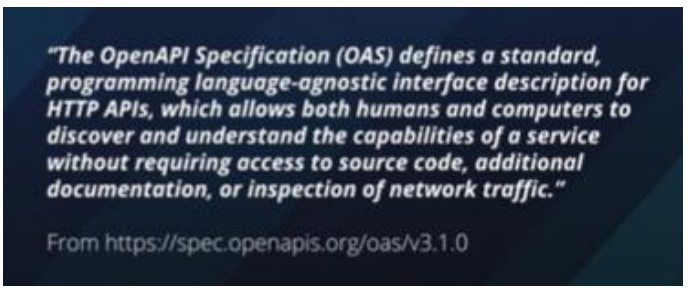
- GraphQL https://countries.trevorblades.com/
- https://studio.apollographql.com/public/countries/variant/current/explorer



82

Herramientas para API

- OPEN API



83

Herramientas para API

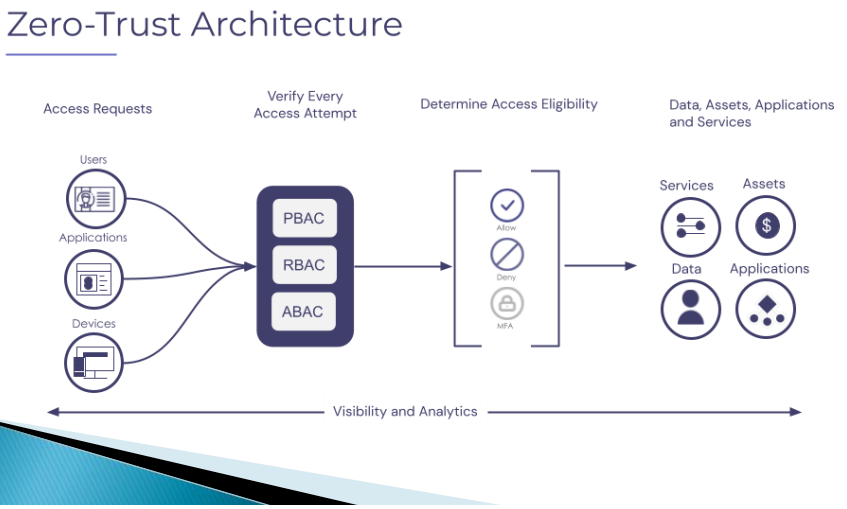
- OPEN API

```
openapi: 3.0.0
info:
 title: Pet Adoption
 description: Adopt 'em all
 version: 1.0.2
servers:
 - url: http://api.production.com/v1
   description: Production API
 - url: http://api.development.com/v1
   description: Development API
paths:
```

84

NIST SP 800-228 – Enfoque de ciclo de vida

Zero Trust en APIs



85

Diseño Seguro de APIS

- **Gobernanza & exposición.**– Publicar las APIs solo detrás de un API Gateway con TLS actualizados. Documentar con OpenAPI y usar atributos de parámetros.

```yaml
type: object
properties:
  id:
    # Returned by GET, not used in POST/PUT/PATCH
    type: integer
    readOnly: true
  username:
    type: string
  password:
    # Used in POST/PUT/PATCH, not returned by GET
    type: string
    writeOnly: true
```

86

Diseño Seguro de APIS

- **Gobernanza & exposición.**– Publicar las APIs solo detrás de un API Gateway con TLS actualizados. Documentar con OpenAPI y usar atributos de parámetros.

```yaml
/pet/findByTags:
  get:
    deprecated: true
```

87

Diseño Seguro de APIS

Descubrimiento y gestión de postura.

- Descubrimiento automático: detección de APIs shadow desde el código hasta runtime
- Evaluación de postura: validación de configuraciones seguras, exposición de datos, cumplimiento OWASP Top 10

88

Diseño Seguro de APIS

Protección en tiempo de ejecución

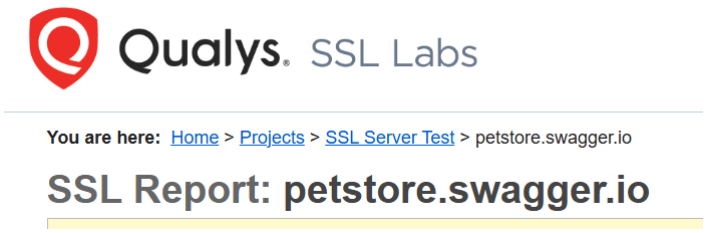
Monitorización y detección de anomalías: patrones que señalan manipulación o acceso indebido

Rate limiting

89

Diseño Seguro de APIS

- **Cifrado en tránsito.-** Toda comunicación entre cliente y API debe estar cifrada (TLS/HTTPS), sin excepciones, para garantizar confidencialidad e integridad



90

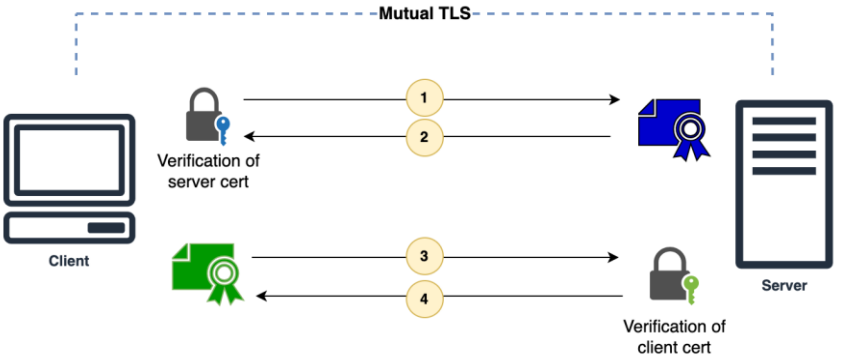
Diseño Seguro de APIS

- **Autenticación de servicio.-** Cada servicio (máquinas, aplicaciones, microservicios) debe probar su identidad antes de invocar la API. Esto puede realizarse mediante certificados mTLS, JWT firmados o API Keys.

91

Diseño Seguro de APIS

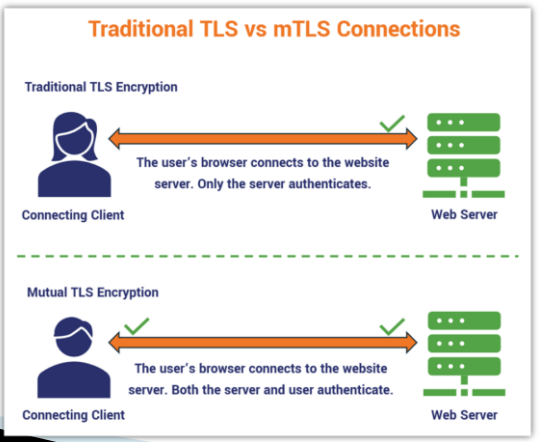
- Autenticación de servicio



92

Diseño Seguro de APIS

- Autenticación de servicio



93

Diseño Seguro de APIS

- **Autorización de Servicio.-** Después de autenticarse, se evalúan permisos específicos al servicio: ¿qué recursos y acciones le están permitidos? Esto aplica también al contexto de servicio

94

Diseño Seguro de APIS

- **Autenticación de Usuario.-** En caso de que la llamada represente una acción de un usuario final, se requiere autenticación robusta (MFA, OIDC/OAuth2, tokens JWT).

95

Diseño Seguro de APIS

- **Autenticación de Usuario.-** En caso de que la llamada represente una acción de un usuario final, se requiere autenticación robusta (MFA, OIDC/OAuth2, tokens JWT).

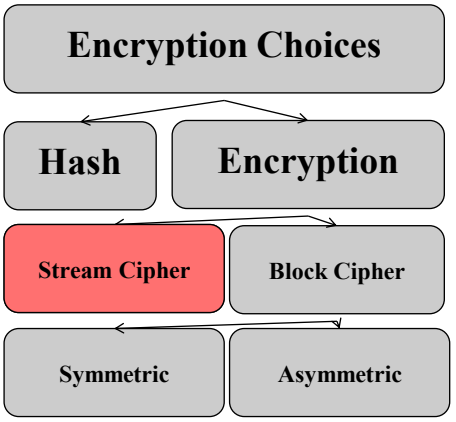
96

Diseño Seguro de APIS

Aspecto	Codificación	Criptografía	Hash
Propósito	Cambio de formato (legible)	Confidencialidad	Integridad / huellas
Ejemplo	Base64, URL encoding	AES-256, RSA	SHA-256, BLAKE3
Reversibilidad	☑ sin clave	☑ con clave	✗ irreversible
Escenario típico	Enviar datos codificados	Enviar datos sensibles	Verificar descargas

97

Diseño Seguro de APIS



No crear algoritmos, a menos que sea extrictamente necesario.

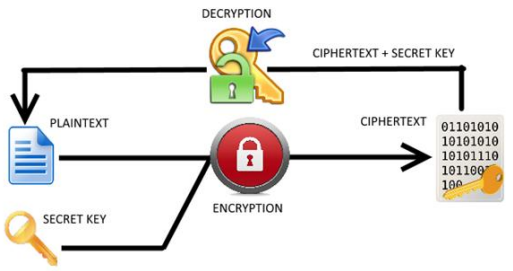
Una via o Reversible?
(e.g., SHA-256 vs. AES)

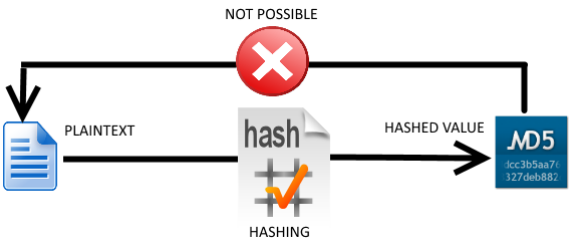
Bits vs. Blocks
(e.g., RC4 vs. AES)

Shared vs. Public & Private Keys
(e.g., AES vs. RSA)

98

Diseño Seguro de APIS





99

Diseño Seguro de APIS

- Encryption
https://codebeautify.org/encrypt-decrypt

- Hash
https://www.onlinehashcrack.com/hash-generator.php

100

Diseño Seguro de APIS

- Que tipo de algoritmo debo utilizar para almacenar un password en las bases de datos de las aplicaciones?

- ¿En qué casos debo utilizar "Encryption" en lugar de Hash? ¿Y cuáles son los riegos?

User	Password
Stephen	auhsoJ
Lisa	hsifdrowS
James	1010NO1Z
Harry	sinocarD tupaC
Sarah	auhsoJ

User	Password Hash
Stephen	39e717cd3f5c4be78d97090c69f4e655
Lisa	f567c40623df407ba980bfad6dff5982
James	711f1f88006a48859616c3a5cbcc0377
Harry	fb74376102a049b9a7c5529784763c53
Sarah	39e717cd3f5c4be78d97090c69f4e655

101

Diseño Seguro de APIS

- Técnicas para aprovechar el uso de hash

Free Rainbow Tables
Distributed Rainbow Table Project

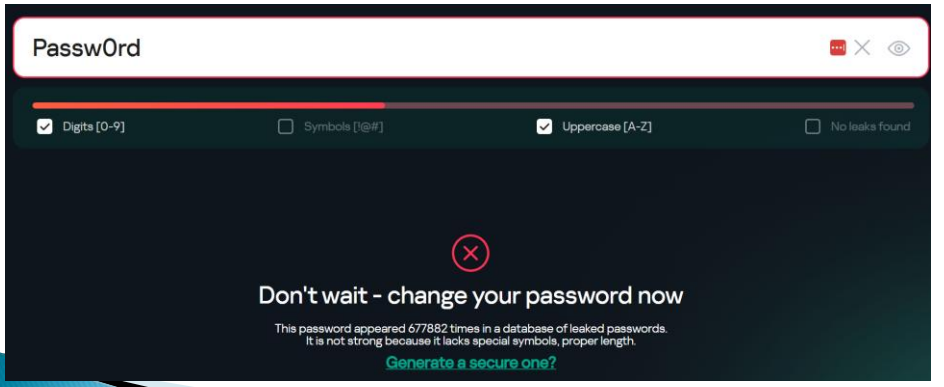
The goal of FreeRainbowTables.com is to prove the insecurity of using simple hash routines to protect valuable passwords, and force developers to use more secure methods.
By distributing the generation of rainbow chains, we can generate HUGE rainbow tables that are able to crack longer passwords than ever seen before.
Furthermore, we are also improving the rainbow table technology, making them even smaller and faster than rainbow tables found elsewhere, and the best thing is, those tables are freely available!

Character set and password length Hover your mouse over the below for more information	NTLM 4 TB	SHA-1 [1] and MySQLSHA1 3 TB	MD5 4.3 TB	LM 398 GB	Half LM challenge 18 GB
all-space#1-7 [2]				34 GB: 0 1 2 3	18 GB: 0 1 2 3
alpha#1-1,loweralpha#5-5,loweralpha-numeric#2-2,numeric#1-3	362 GB: 0 1 2 3		362 GB: 0 1 2 3		
alpha-space#1-9	35 GB: 0 1 2 3		23 GB: 0 1 2 3		
lm-frt-cp437-850#1-7				364 GB: 0 1 2 3	
loweralpha#1-10		179 GB: 0 1 2 3	179 GB: 0 1 2 3		
loweralpha#7-7,numeric#1-3	26 GB: 0 1 2 3		26 GB: 0 1 2 3		
loweralpha-numeric#1-10	587 GB: 0 8 16 24	587 GB: 0 8 16 24	588 GB: 0 8 16 24		
loweralpha-numeric-space#1-8	15 GB: 0 1 2 3	17 GB: 0 1 2 3	16 GB: 0 1 2 3		
loweralpha-numeric-space#1-9		108 GB: 0 1 2 3	108 GB: 0 1 2 3		
loweralpha-numeric-symbol32-space#1-7	33 GB: 0 1 2 3	33 GB: 0 1 2 3	33 GB: 0 1 2 3		
loweralpha-numeric-symbol32-space#1-8	428 GB: 0 1 2 3	427 GB: 0 1 2 3	425 GB: 0 1 2 3		

102

Diseño Seguro de APIS

- Para los casos de almacenamiento de password la seguridad depende también de la longitud



https://password.kaspersky.com/

103

Diseño Seguro de APIS

- Computo en nube hace la vida mas sencilla para un atacante

https://crackstation.net/

104

Diseño Seguro de APIS

- Veracode - Cryptography Storage Issues

105

Diseño Seguro de APIS

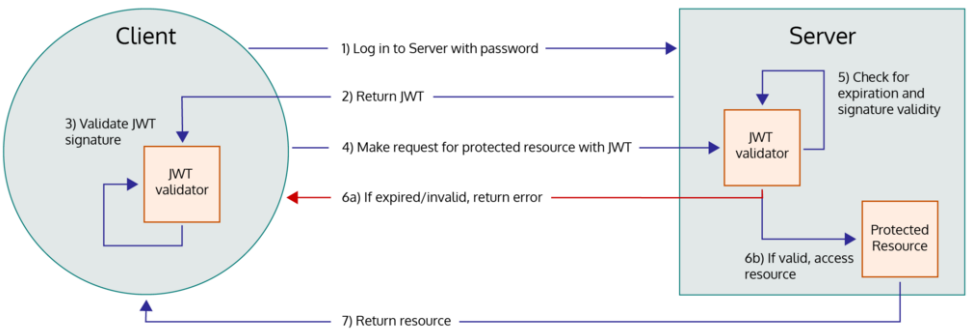
- Autenticación de Usuario.

Mecanismo	Objetivo principal	Referencia formal / *RFC*	Estructura del encabezado o token	Ejemplo
Basic Auth	Enviar credenciales (usuario + contraseña) codificadas en Base64 para autenticar cada petición. Muy simple; no cifra el canal.	RFC 7617 (actualiza 2617 y 1945)	Authorization: Basic <base64(user:pass)>	curl -u alice:s3cr3t https://api.ejemplo.com/v1/items
Bearer Token	Portar ("bear") un token opaco que representa una sesión o "scope" de acceso. No requiere firma por petición.	RFC 6750 (OAuth 2.0 Bearer)	Authorization: Bearer <token>	curl -H "Authorization: Bearer eyJhbGci..." https://api.ejemplo.com/v1/items
JWT Bearer	Usar un JSON Web Token auto-contenedor (claims + firma) como *bearer*; a menudo se emite en los flujos OAuth 2.0 o OpenID Connect.	RFC 7523 (JWT Profile) + RFC 7519 (JWT)	Authorization: Bearer <jwt-compact> JWT = header.payload.signature (Base64URL)	curl -H "Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...." https://api.ejemplo.com/v1/items

106

Diseño Seguro de APIS

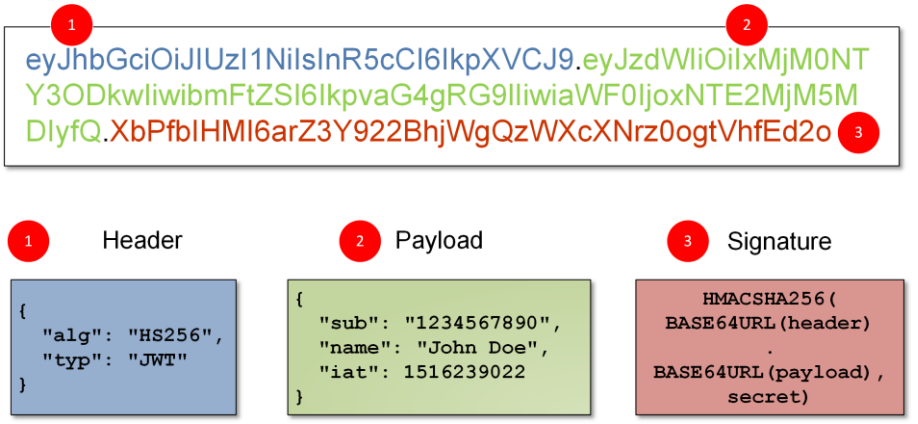
- Autenticación – JWT.



107

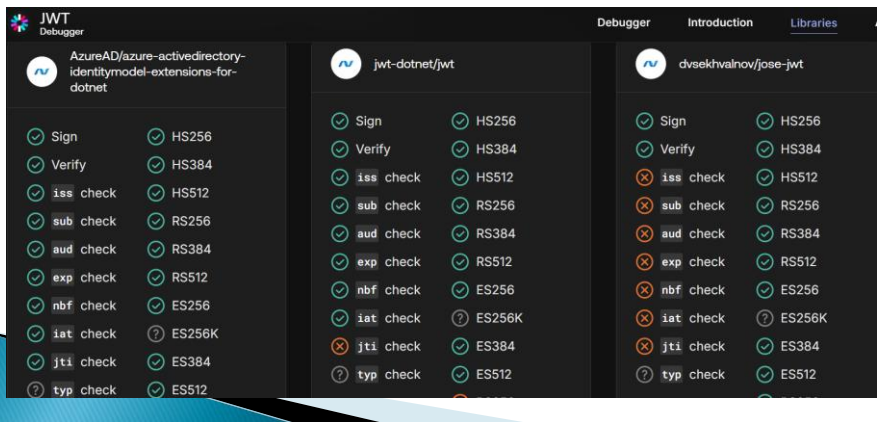
Diseño Seguro de APIS

- **Autenticación – JWT.**



Diseño Seguro de APIS

- **Autenticación – JWT.**
- https://jwt.io/libraries



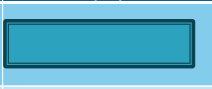
Diseño Seguro de APIS

- **Autenticación Test.**
- https://petstore.swagger.io/v2/swagger.json

110

Diseño Seguro de APIS

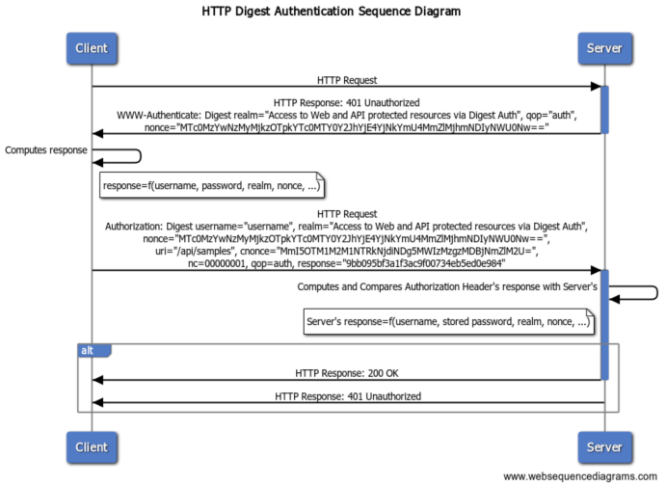
- **Autenticación de Usuario.**

Mecanismo	Objetivo principal	Referencia formal / *RFC*	Estructura del encabezado o token	Ejemplo
Digest Auth	Evitar envío directo de la contraseña mediante reto-respuesta con *nonce* y hash (MD5, SHA-256).	RFC 7616 (actualiza 2617)	Authorization: Digest username="alice", realm="api", nonce="abc", uri="/v1/items", response="6629fae49393a05397450978507c4ef1"	
OAuth 1.0	Delegar acceso con firmas HMAC-SHA1 en cada petición sin compartir la contraseña. Aún usado por Twitter v1, entre otros.	RFC 5849	Authorization: OAuth oauth_consumer_key="...", oauth_nonce="...", oauth_signature="...", ...	curl --oauth1.0 --user "ck:cs" --data "status=hola" https://api.twitter.com/1.1/statuses/update.json
OAuth 2.0	Delegación moderna basada en *flows* (Auth Code, Client Credentials, Device, etc.). Tokens de acceso suelen ser Bearer o JWT.	RFC 6749 (Framework) + 6750 (Bearer)	Flujo → intercambio de código por access_token; luego Authorization: Bearer <token>	bash # Paso 1: obtener código (redirección) # Paso 2: curl -X POST -d "grant_type=authorization_code&code=abcd...&client_id=123&client_secret=xyz" https://auth.ejemplo.com/token # Paso 3: usar token con Bearer

111

Diseño Seguro de APIS

- Autenticación de Usuario – Digest.



Diseño Seguro de APIS

- Autenticación de Usuario – Digest.

Supported Algorithms

The authentication process supports various hashing algorithms, including:

- **MD5**: The original algorithm specified in RFC 2069 and RFC 2617.
- **MD5-sess**: An enhancement introduced in RFC 2617.
- **SHA-256** and **SHA-256-sess**: Added in RFC 7616 to provide stronger security.
- **SHA-512-256** and **SHA-512-256-sess**: Also introduced in RFC 7616 for enhanced security. Wikipedia +

Diseño Seguro de APIS

- **Autorización granular**

SCOPES OAuth2/OpenID Connect. Representan permisos específicos en tokens

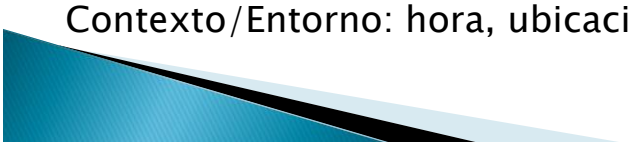
Funcionan como limitadores adicionales dentro de un rol (p. ej. un rol User con scopes read y/o write)

114

Diseño Seguro de APIS

ABAC (Attribute-Based Access Control)
El acceso se basa en la evaluación de una regla que combina atributos del sujeto, recurso, acción y contexto

Atributos típicos:
Sujeto/User: rol, departamento
Recurso/Objeto: tipo de documento, sensibilidad, propietario
Acción: CRUD (leer, editar, eliminar)
Contexto/Entorno: hora, ubicación, dispositivo



115

Diseño Seguro de APIS

Autorización granular

Alcance (scope)	Permisos que concede	Endpoints protegidos
invoice.read	Consultar facturas	GET /invoices, GET /invoices/{id}
invoice.write	Crear/actualizar	POST /invoices, PUT /invoices/{id}
invoice.admin	Borrar / ver auditoría	DELETE /invoices/{id}, GET /audit

1. **Crear cliente OIDC**
   Client ID: `invoice-web` | *Access Type*: **confidential** (genera `client_secret`).
2. **Definir scopes**
   - `invoice.read` → *Default*
   - `invoice.write` → *Optional*
   - `invoice.admin` → *Optional*

116

Diseño Seguro de APIS

Autorización granular– Request

```
GET /realms/acme/protocol/openid-connect/auth?
    client_id=invoice-web&
    response_type=code&
    redirect_uri=https://app.acme.com/callback&
    scope=openid%20profile%20invoice.read%20invoice.write
```

117

Diseño Seguro de APIS

Autorización granular – Request

```
curl -X POST https://id.acme.com/realms/acme/protocol/openid-connect/token \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "grant_type=authorization_code" \
  -d "client_id=invoice-web" \
  -d "client_secret=$CLIENT_SECRET" \
  -d "code=SpLXlOBeZQ..." \
  -d "redirect_uri=https://app.acme.com/callback"
```

118

Diseño Seguro de APIS

Autorización granular – Response

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "id_token": "eyJraWQiOiJhYmMxMjMiLCJhbGciOiJSUzI1NiJ9...",
  "refresh_token": "dXNlci0yMzQ1...",
  "token_type": "Bearer",
  "expires_in": 900,
  "scope": "openid profile invoice.read invoice.write"
}
```

119

Diseño Seguro de APIS

Autorización granular – Ejecución APIs

```
GET /invoices HTTP/1.1
Host: api.acme.com
Authorization: Bearer eyJhbGciOiJ...     ← scopes: openid profile invoice.read invoice.write
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Cache-Control: no-store, must-revalidate

{
  "data": [
    { "id": "INV-0001", "number": "2025-001", "amount": 1520.33, "currency": "USD", "status":
    { "id": "INV-0002", "number": "2025-002", "amount": 230.00,  "currency": "USD", "status":
  ],
  "links": {
    "self": "/invoices",
    "next": null
  }
}
```

120

Diseño Seguro de APIS

Autorización granular – Ejecución APIs

```
DELETE /invoices/123 HTTP/1.1
Host: api.acme.com
Authorization: Bearer eyJhbGciOiJ...     ← mismas scopes (sin invoice.admin)
Accept: application/json
```

```
HTTP/1.1 403 Forbidden
Cache-Control: no-store
                                                    voice.admin",
Content-Type: application/json


{
  "error": "access_denied"
}
```

121

Diseño Seguro de APIS

Autorización granular – Ejecución APIs

```
DELETE /invoices/123 HTTP/1.1
Host: api.acme.com
Authorization: Bearer eyJhbGciOiJ...     ← mismas scopes (sin invoice.admin)
Accept: application/json
```

```
HTTP/1.1 403 Forbidden
Cache-Control: no-store
Content-Type: application/json

{
  "error": "access_denied"
}
```

122

Diseño Seguro de APIS

Validación y Sanitización

Control	Requisito destacado	Por qué
"Positive validation"	Aceptar solo tipos, rangos y formatos esperados ("lista blanca").	Evita inyecciones / datos corruptos.
Entrada *-y-* salida	Validar *request* **y** *response* antes de exponerlos al cliente.	Minimiza *data leakage* y errores 500.
Límites duros	Tamaño máximo de JSON / profundidad / recursión.	Protege de DoS y abuso GraphQL.
Deserialización segura	Sólo clases permitidas; bloquear tipos polimórficos.	Previene RCE por gadget chains (Java, .NET).
Sanitización	Escape HTML/SQL/XML cuando no existan listas blancas.	Mitiga XSS o XXE en APIs inseguras.

123

OWASP Top 10 API Security Risks – 2023

124

OWASP Top 10 API

- Las API desempeñan un papel fundamental en la arquitectura de aplicaciones modernas. Sin embargo, dado que la innovación avanza a un ritmo diferente al de la concienciación sobre seguridad, creemos que es importante centrarse en concienciar sobre las debilidades comunes de seguridad de las API.
- El objetivo principal del Top 10 de Seguridad de API de OWASP es capacitar a quienes participan en el desarrollo y mantenimiento de API, como desarrolladores, diseñadores, arquitectos, administradores u organizaciones.

125

OWASP Top 10 API

APIs in the past		APIs today
Limited and non-sensitive data	----→	**Lots of very sensitive data exposed**
Small attack surface	----→	**Large attack surface**
Static attack surface	----→	**Dynamic attack surface**

Gartner

*"By 2022, API abuses will move from an infrequent to **the most-frequent attack** vector, resulting in data breaches for enterprise web applications."*

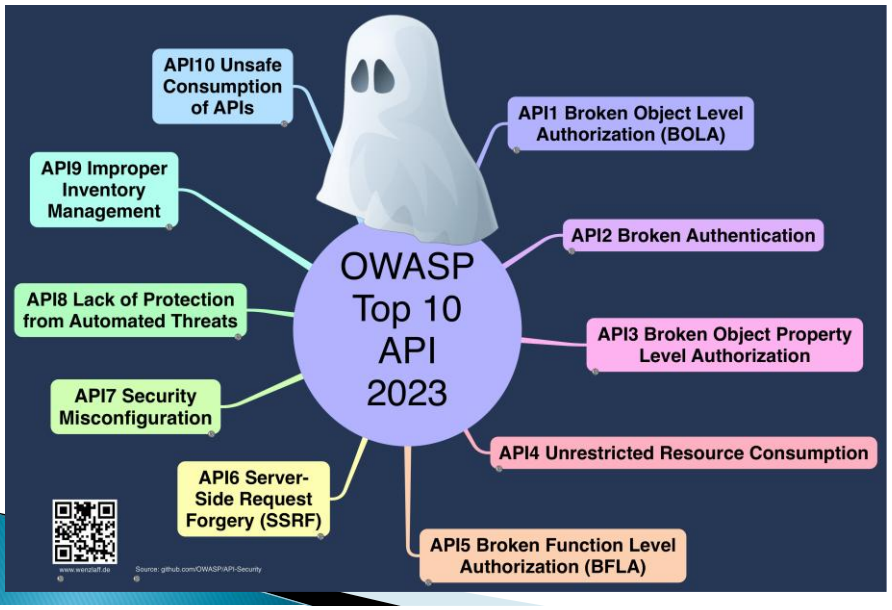
126

OWASP Top 10 API

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impacts
API Specific	Easy: **3**	Widespread **3**	Easy **3**	Severe **3**	Business Specific
API Specific	Average: **2**	Common **2**	Average **2**	Moderate **2**	Business Specific
API Specific	Difficult: **1**	Difficult **1**	Difficult **1**	Minor **1**	Business Specific

127

OWASP Top 10 API



128

OWASP Top 10 API

OWASP API Security Top-10 2019	OWASP API Security Top-10 2023	
API1 Broken Object Level Authorization	API1 Broken Object Level Authorization	Same
API2 Broken User Authentication	API2 Broken Authentication	Updated
API3 Excessive Data Exposure	API3 Broken Object Property Level Authorization	Updated
API4 Lack of Resources & Rate Limiting	API4 Unrestricted Resource Consumption	Updated
API5 Broken Function Level Authorization	API5 Broken Function Level Authorization	Same
API6 Mass Assignment	API6 Unrestricted Access to Sensitive Business Flows	New
API7 Security Misconfiguration	API7 Server-Side Request Forgery (SSRF)	New
API8 Injection	API8 Security Misconfiguration	Same
API9 Improper Assets Management	API9 Improper Inventory Management	Updated
API10 Insufficient Logging & Monitoring	API10 Unsafe Consumption of APIs	New

129

API1:2023 – Broken Object Level Authorization

APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface of Object Level Access Control issues. Object level authorization checks should be considered in every function that accesses a data source using an ID from the user

130

API1:2023 – Broken Object Level Authorization

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Widespread** : Detectability **Easy**	Technical **Moderate** : Business Specific
Attackers can exploit API endpoints that are vulnerable to broken object-level authorization by manipulating the ID of an object that is sent within the request. Object IDs can be anything from sequential integers, UUIDs, or generic strings. Regardless of the data type, they are easy to identify in the request target (path or query string parameters), request headers, or even as part of the request payload.	This issue is extremely common in API-based applications because the server component usually does not fully track the client's state, and instead, relies more on parameters like object IDs, that are sent from the client to decide which objects to access. The server response is usually enough to understand whether the request was successful.	Unauthorized access to other users' objects can result in data disclosure to unauthorized parties, data loss, or data manipulation. Under certain circumstances, unauthorized access to objects can also lead to full account takeover.

131

API1:2023 – Broken Object Level Authorization

Legitimate – *userId* matches in the query parameter and request

```
Request:
GET /v1/customers/15981?userId=207939055 HTTP/1.1

Authorization: Bearer gwwh1Y4epjv9Y

Cookie: _ga=GA1.3.630674023.1502871544;
_gid=GA1.2.1579405782.1502871544;userId=207939055
Host: payments-api.dnssf.com
X-Forwarded-For: 54.183.50.90

Response:
200 OK

{
     userId: 207939055,
     firstName: "John",
     lastName: "Smith",
     email: "john.smith@acme.com",
     phoneNumber: "+1650123123"
}
```

Attack – Attacker changes the *userId* in the query parameter

```
Request:
GET /v1/customers/15981?userId=207938044 HTTP/1.1

Authorization: Bearer gwwh1Y4epjv9Y

Cookie: _ga=GA1.3.630674023.1502871544;
_gid=GA1.2.1579405782.1502871544;userId=207939055
Host: payments-api.dnssf.com
X-Forwarded-For: 54.183.50.90

Response:
200 OK

{
    userId: 207938044,
     firstName: "David",
     lastName: "Miller",
     email: "david.miller@example.com",
     phoneNumber: "+1912456456"
}
```

132

API1:2023 – Broken Object Level Authorization

- Unauthorized access can result in unauthorized data access, data loss, or data manipulation. Unauthorized access to objects can also lead to full account takeover.
- In this example, the backend logic queries the database with the userId in the query parameter while verifying the authorization with the userId in the cookie – an attacker could simply enumerate it and extract data.

133

API1:2023 – Broken Object Level Authorization

What Happened

In September 2019, a critical bug was discovered on Uber API, which allows merchants, service providers and others to offer ride-sharing services to customers. Uber had exposed a vulnerable application programming interface (API) endpoint that allowed attackers to steal valuable data, including personally identifiable information (PII) records and authentication tokens of riders and drivers. The leaked authentication token could be used to perform a full account takeover.

Luckily for the company, the vulnerability was discovered before harm could be done. But the case is an example of where traditional security systems can fail to find potential threats because they lack the business context for the application's logic. Let's take a closer look at what happened and the implications.

134

API1:2023 – Broken Object Level Authorization

When a new Uber driver joins the platform through a referral link, their browser communicates with the API host "**bonjour.uber.com**". The registration process triggers an API call to the API endpoint of:

`POST /marketplace/\_rpc?rpc=getConsentScreenDetails`

```
POST /marketplace/\_rpc?rpc=getConsentScreenDetails HTTP/1.1
Host: bonjour.uber.com
Connection: close
Content-Length: 67
Accept: application/json
Origin: [https://bonjour.uber.com](https://bonjour.uber.com)
x-csrf-token: xxxx
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML,
DNT: 1
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: xxxxx
{"language":"en","userUuid":"xxxx-776-4xxxx1bd-861a-837xxx604ce"}
```

135

API1:2023 – Broken Object Level Authorization

When a new Uber driver joins the platform through a referral link, their browser communicates with the API host "**bonjour.uber.com**". The registration process triggers an API call to the API endpoint of:

`POST /marketplace/\_rpc?rpc=getConsentScreenDetails`

```
POST /marketplace/\_rpc?rpc=getConsentScreenDetails HTTP/1.1
Host: bonjour.uber.com
Connection: close
Content-Length: 67
Accept: application/json
Origin: [https://bonjour.uber.com](https://bonjour.uber.com)
x-csrf-token: xxxx
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML,
DNT: 1
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: xxxxx
{"language":"en","userUuid":"xxxx-776-4xxxx1bd-861a-837xxx604ce"}
```

136

API1:2023 – Broken Object Level Authorization


```
{
   "status":"success",
   "data":{
      "data":{
         "language":"en",
         "userUuid":"xxxxxx1e"
      },
      "getUser":{
         "uuid":"cxxxxxc5f7371e",
         "firstname":"Maxxxx",
         "lastname":"XXXX",
         "role":"PARTNER",
         "languageId":1,
         "countryId":77,
         "mobile":null,
         "mobileToken":1234,
         "mobileCountryId":77,
         "mobileCountryCode":"+91",
```


137

API1:2023 – Broken Object Level Authorization

- Implement a proper authorization mechanism that relies on the user policies and hierarchy.
- Use the authorization mechanism to check if the logged-in user has access to perform the requested action on the record in every function that uses an input from the client to access a record in the database.
- Prefer the use of random and unpredictable values as GUIDs for records' IDs.
- Write tests to evaluate the vulnerability of the authorization mechanism. Do not deploy changes that make the tests fail.

138

API2:2023 Broken Authentication

- Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently.
- Compromising a system's ability to identify the client/user, compromises API security overall.

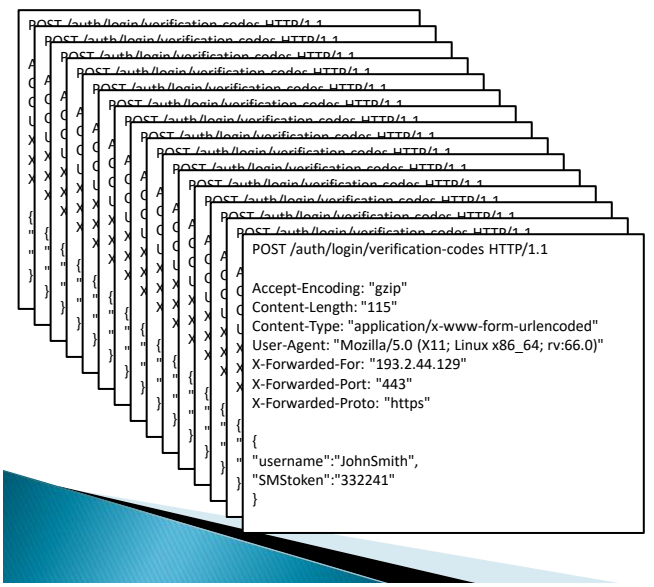
139

API2:2023 Broken Authentication

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Common** : Detectability **Easy**	Technical **Severe** : Business Specific
The authentication mechanism is an easy target for attackers since it's exposed to everyone. Although more advanced technical skills may be required to exploit some authentication issues, exploitation tools are generally available.	Software and security engineers' misconceptions regarding authentication boundaries and inherent implementation complexity make authentication issues prevalent. Methodologies of detecting broken authentication are available and easy to create.	Attackers can gain complete control of other users' accounts in the system, read their personal data, and perform sensitive actions on their behalf. Systems are unlikely to be able to distinguish attackers' actions from legitimate user ones.

140

API2:2023 Broken Authentication



- For example, password recovery often uses an SMS sent to the customer's phone. An attacker can try 999,999 codes within a few minutes, if the API does not implement a rate limiting policy.
- API protection solutions should profile the average usage for every API endpoint to detect abnormally excessive calling of a specific API endpoint.

141

API2:2023 Broken Authentication

What Happened

Parler improperly allowed mass collection of archived data (images, videos, information) that were posted onto their service. This was due to an unprotected API call that was sequentially numbered, therefore allowing any attacker to iterate continuously over the endpoint and take all information available – which is reaching upwards of 60TB now with over one million videos alone.

By having no security protections on who can iterate these endpoints, nor any rate-limiting protections, the internet was generally able to capture all data available. **This culminated in the gathering over 60TB of data with massive amounts of metadata, well over 1,400 unique types of data connected to the accessed data ranging from geolocation to the type of phone used.**

142

API2:2023 Broken Authentication

For example, if you have a corporate website and you store your PDFs numbered at `http://www.acme.com/pdfs/1.pdf`, that would allow an attacker to then guess the correct URL structure for 2.pdf, 3.pdf, and so on and so forth until they are detected and stopped, or they divulge all the information that they desire. In the case of Parler, their URLs looked like `https://par(dot)pw/v1/photo?id=` and the ID could be sequentially increased to gather information from the API without direct knowledge.

```
195   if allowed(url, nil) and status_code == 200 and not (
196       string.match(url, "^https?://images%.parler%.com/")
197       or string.match(url, "^https?://image%-cdn%.parler%.com/")
198       or string.match(url, "^https?://video%.parler%.com/")
199     ) then
200     html = read_file(file)
201     if string.match(url, "^https?://api%.parler%.com/v3/uuidConversion/") then
202       local id = string.match(html, "([0-9a-f]+)")
203       ids[id] = true
204       check("https://parler.com/post/" .. id)
205       check("https://share.par.pw/post/" .. id)
206     end
207     local match = string.match(url, "^https?://[^/]*parler%.com/post/([0-9a-f]+)$")
208     if match then
209       check("https://share.par.pw/post/" .. match)
210     end
```

143

API2:2023 Broken Authentication

- Make sure you know all the possible flows to authenticate to the API (mobile/ web/deep links that implement one-click authentication/etc.). Ask your engineers what flows you missed.
- Read about your authentication mechanisms. Make sure you understand what and how they are used. OAuth is not authentication, and neither are API keys.
- Don't reinvent the wheel in authentication, token generation, or password storage. Use the standards.

144

API2:2023 Broken Authentication

- Credential recovery/forgot password endpoints should be treated as login endpoints in terms of brute force, rate limiting, and lockout protections.
- Require re-authentication for sensitive operations (e.g. changing the account owner email address/2FA phone number).
- Where possible, implement multi-factor authentication.

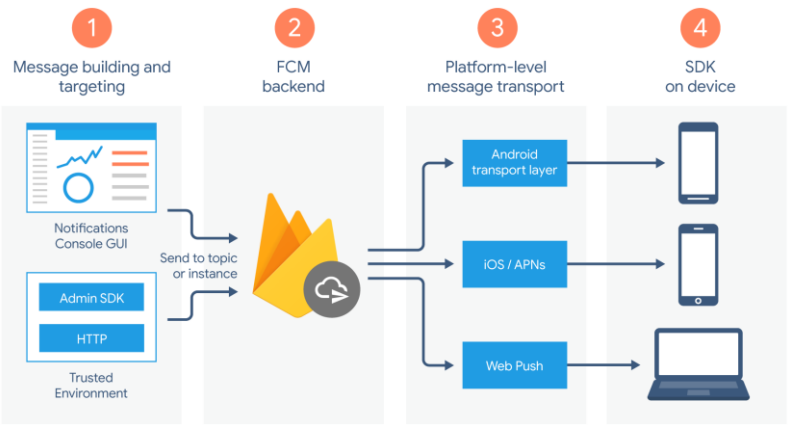
145

API2:2023 Broken Authentication

- Implement anti-brute force mechanisms to mitigate credential stuffing, dictionary attacks, and brute force attacks on your authentication endpoints. This mechanism should be stricter than the regular rate limiting mechanisms on your APIs.
- Implement account lockout/captcha mechanisms to prevent brute force attacks against specific users. Implement weak-password checks.
- API keys should not be used for user authentication. They should only be used for API clients authentication.

146

API2:2023 Broken Authentication



147

API3:2023 – Broken Object Property Level Authorization

This category combines API3:2019 Excessive Data Exposure and API6:2019 – Mass Assignment, focusing on the root cause: the lack of or improper authorization validation at the object property level. This leads to information exposure or manipulation by unauthorized parties.

148

API3:2023 – Broken Object Property Level Authorization

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Common** : Detectability **Easy**	Technical **Moderate** : Business Specific
APIs tend to expose endpoints that return all object's properties. This is particularly valid for REST APIs. For other protocols such as GraphQL, it may require crafted requests to specify which properties should be returned. Identifying these additional properties that can be manipulated requires more effort, but there are a few automated tools available to assist in this task.	Inspecting API responses is enough to identify sensitive information in returned objects' representations. Fuzzing is usually used to identify additional (hidden) properties. Whether they can be changed is a matter of crafting an API request and analyzing the response. Side-effect analysis may be required if the target property is not returned in the API response.	Unauthorized access to private/sensitive object properties may result in data disclosure, data loss, or data corruption. Under certain circumstances, unauthorized access to object properties can lead to privilege escalation or partial/full account takeover.

149

API3:2023 – Broken Object Property Level Authorization

Legitimate request – user retrieving stored credit card information	**Response with data exposure** - HTTP response to the API call contains sensitive data in the message body
Request: POST /payments/storedcard/json HTTP/1.1 Host: payments.host.com Connection: close Content-Length: 78 Cache-Control: max-age=0 Origin: https://payments.host.com Content-Type: application/x-www-form-urlencoded User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36	**Response:** HTTP/1.1 200 OK Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: Mon, 01 Jan 1990 00:00:00 GMT Date: Wed, 27 Jan 2021 15:43:39 GMT Content-Type: application/json; charset=utf-8 X-Frame-Options: SAMEORIGIN X-XSS-Protection: 1; mode=block Connection: close Content-Length: 55
(KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 Referer: https://payments.host.com/methods/ Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.9 Cookie: session=kjasdnfklnf01922wndlasdjkfnz-0f71k9013u18901u2mklsduhz12234d4D	[{"PAN":"4111111123454321","status":"ok","CVV":"1234"}]

150

API3:2023 – Broken Object Property Level Authorization

An online marketplace platform, that offers one type of users ("hosts") to rent out their apartment to another type of users ("guests"), requires the host to accept a booking made by a guest, before charging the guest for the stay.

As part of this flow, an API call is sent by the host to `POST /api/host/approve_booking` with the following legitimate payload:

```
{
  "approved": true,
  "comment": "Check-in is after 3pm"
}
```

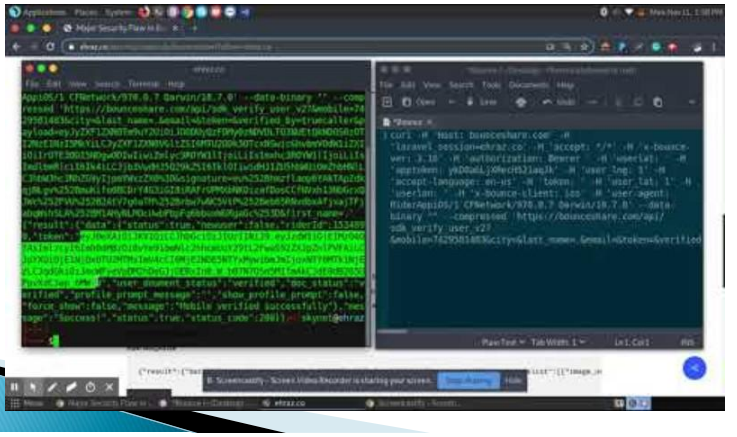
The host replays the legitimate request, and adds the following malicious payload:

```
{
  "approved": true,
  "comment": "Check-in is after 3pm",
  "total_stay_price": "$1,000,000"
}
```

151

API3:2023 – Broken Object Property Level Authorization

In 2019 a security researcher found that by passing a phone number in an API request the Bounceshare application would return an access token and RiderId associated with the account for that phone number.



152

API3:2023 – Broken Object Property Level Authorization

- When exposing an object using an API endpoint, always make sure that the user should have access to the object's properties you expose.
- Avoid using generic methods such as to_json() and to_string(). Instead, cherry-pick specific object properties you specifically want to return.
- If possible, avoid using functions that automatically bind a client's input into code variables, internal objects, or object properties ("Mass Assignment").

153

API3:2023 – Broken Object Property Level Authorization

- Allow changes only to the object's properties that should be updated by the client.
- Implement a schema-based response validation mechanism as an extra layer of security. As part of this mechanism, define and enforce data returned by all API methods.
- Keep returned data structures to the bare minimum, according to the business/functional requirements for the endpoint.

154

API4:2023 – Unrestricted Resource Consumption

- Satisfying API requests requires resources such as network bandwidth, CPU, memory, and storage. Other resources such as emails/SMS/phone calls or biometrics validation are made available by service providers via API integrations, and paid for per request. Successful attacks can lead to Denial of Service or an increase of operational costs.

155

API4:2023 – Unrestricted Resource Consumption

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Average**	Prevalence **Widespread** : Detectability **Easy**	Technical **Severe** : Business Specific
Exploitation requires simple API requests. Multiple concurrent requests can be performed from a single local computer or by using cloud computing resources. Most of the automated tools available are designed to cause DoS via high loads of traffic, impacting APIs' service rate.	It's common to find APIs that do not limit client interactions or resource consumption. Crafted API requests, such as those including parameters that control the number of resources to be returned and performing response status/time/length analysis should allow identification of the issue. The same is valid for batched operations. Although threat agents don't have visibility over costs impact, this can be inferred based on service providers' (e.g. cloud provider) business/pricing model.	Exploitation can lead to DoS due to resource starvation, but it can also lead to operational costs increase such as those related to the infrastructure due to higher CPU demand, increasing cloud storage needs, etc.

156

API4:2023 – Unrestricted Resource Consumption

A GraphQL API Endpoint allows the user to upload a profile picture.

```
POST /graphql

{
  "query": "mutation {
    uploadPic(name: \"pic1\", base64_pic: \"R0FOIEFOR0xJVA…\") {
      url
    }
  }"
}
```

Once the upload is complete, the API generates multiple thumbnails with different sizes based on the uploaded picture. This graphical operation takes a lot of memory from the server.

157

API4:2023 – Unrestricted Resource Consumption

The API implements a traditional rate limiting protection - a user can't access the GraphQL endpoint too many times in a short period of time. The API also checks for the uploaded picture's size before generating thumbnails to avoid processing pictures that are too large.

An attacker can easily bypass those mechanisms, by leveraging the flexible nature of GraphQL:

```
POST /graphql

[
  {"query": "mutation {uploadPic(name: \"pic1\", base64_pic: \"R0FOIEFOR0xJVA…\") {url}}
  {"query": "mutation {uploadPic(name: \"pic2\", base64_pic: \"R0FOIEFOR0xJVA…\") {url}}
  ...
  {"query": "mutation {uploadPic(name: \"pic999\", base64_pic: \"R0FOIEFOR0xJVA…\") {url
}
```

Because the API does not limit the number of times the `uploadPic` operation can be attempted, the call will lead to exhaustion of server memory and Denial of Service.

158

API4:2023 – Unrestricted Resource Consumption

Legitimate – max_return and page_size request attributes are normal

```
POST
/example/api/v1/provision/user/search
HTTP/1.1
User-Agent: AHC/1.0
Connection: keep-alive
Accept: */*
Content-Type: application/json;
charset=UTF-8
Content-Length: 131
X-Forwarded-For: 10.93.23.4

{
  "search_filter":
"user_id=exampleId_100",
  "max_return": "250",
  "page_size": "250",
  "return_attributes": [

  ]
}
```

Attack – Attackers modify the request to return an abnormally high response size

```
POST
/example/api/v1/provision/user/search
HTTP/1.1
User-Agent: AHC/1.0
Connection: keep-alive
Accept: */*
Content-Type: application/json;
charset=UTF-8
Content-Length: 131
X-Forwarded-For: 10.93.23.4

{
  "search_filter":
"user_id=exampleId_100",
  "max_return": “20000”,
  "page_size": "20000",
  "return_attributes": [

  ]
}
```

159

API4:2023 – Unrestricted Resource Consumption

In 2020 the Checkmarx research team found that SoundCloud had not properly implemented rate limiting for the /tracks endpoint of the api-v2.soundcloud.com API. Since no validation was performed for the number of track IDs in the ids list, an attacker could manipulate the list to retrieve an arbitrary number of tracks in a single request and overwhelm the server. Under normal conditions the request issued by the SoundCloud WebApp includes 16 track IDs in the ids query string parameter. The researcher was able to manipulate the list to retrieve up to 689 tracks in a single request causing the service response time to increase by almost 9x. According to Checkmarx "This vulnerability could be used to execute a Distributed Denial of Service (DDoS) attack by using a specially crafted list of track IDs to maximize the response size, and issuing requests from several sources at the same time to deplete resources in the application layer will make the target's system services unavailable."

160

API4:2023 – Unrestricted Resource Consumption

- Use a solution that makes it easy to limit memory, CPU, number of restarts, file descriptors, and processes such as Containers / Serverless code (e.g. Lambdas).
- Define and enforce a maximum size of data on all incoming parameters and payloads, such as maximum length for strings, maximum number of elements in arrays, and maximum upload file size (regardless of whether it is stored locally or in cloud storage).
- Implement a limit on how often a client can interact with the API within a defined timeframe (rate limiting).

161

API4:2023 – Unrestricted Resource Consumption

- Limit/throttle how many times or how often a single API client/user can execute a single operation (e.g. validate an OTP, or request password recovery without visiting the one-time URL).
- Add proper server-side validation for query string and request body parameters, specifically the one that controls the number of records to be returned in the response.
- Configure spending limits for all service providers/API integrations. When setting spending limits is not possible, billing alerts should be configured instead.

162

API4:2023 – Unrestricted Resource Consumption

- Limit/throttle how many times or how often a single API client/user can execute a single operation (e.g. validate an OTP, or request password recovery without visiting the one-time URL).
- Add proper server-side validation for query string and request body parameters, specifically the one that controls the number of records to be returned in the response.
- Configure spending limits for all service providers/API integrations. When setting spending limits is not possible, billing alerts should be configured instead.

163

API5:2023 – Broken Function Level Authorization (BFLA)

Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers can gain access to other users' resources and/or administrative functions.

164

API5:2023 – Broken Function Level Authorization

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Common** : Detectability **Easy**	Technical **Severe** : Business Specific
Exploitation requires the attacker to send legitimate API calls to an API endpoint that they should not have access to as anonymous users or regular, non-privileged users. Exposed endpoints will be easily exploited.	Authorization checks for a function or resource are usually managed via configuration or code level. Implementing proper checks can be a confusing task since modern applications can contain many types of roles, groups, and complex user hierarchies (e.g. sub-users, or users with more than one role). It's easier to discover these flaws in APIs since APIs are more structured, and accessing different functions is more predictable.	Such flaws allow attackers to access unauthorized functionality. Administrative functions are key targets for this type of attack and may lead to data disclosure, data loss, or data corruption. Ultimately, it may lead to service disruption.

165

API5:2023 – Broken Function Level Authorization

Legitimate – POST method is correctly requested

```
POST
/example/api/v1/provision/user/search
HTTP/1.1
User-Agent: AHC/1.0
Connection: keep-alive
Accept: */*
Content-Type: application/json;
charset=UTF-8
Content-Length: 131
X-Forwarded-For: 10.93.23.4

{
  "search_filter":
"user_id=exampleId_100",
  "max_return": "250",
  "page_size": "250",
  "return_attributes": [

  ]
}
```

Attack – Request is modified to send a DELETE method

```
DELETE
/example/api/v1/provision/user/search
HTTP/1.1
User-Agent: AHC/1.0
Connection: keep-alive
Accept: */*
Content-Type: application/json;
charset=UTF-8
Content-Length: 131
X-Forwarded-For: 10.93.23.4

{
  "search_filter":
"user_id=exampleId_100",
  "max_return": “250",
  "page_size": “250",
  "return_attributes": [

  ]
}
```

166

API5:2023 – Broken Function Level Authorization (BFLA)

Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers can gain access to other users’ resources and/or administrative functions.

167

API5:2023 – Broken Function Level Authorization (BFLA)

Real World Example: New Relic Synthetics users can escalate privileges to add or modify alerts

In 2018 Jon Bottarini found that a restricted user could make changes to alerts on Synthetics monitors without the proper permissions to do so. In fact, they could make changes with no permissions at all as a result of the privilege escalation weakness that was present in the product at that time. Exploitation involved submitting a legitimate request to an API endpoint that was otherwise not visible to the restricted user.

As part of his security research, Jon captured traffic of a privileged session using an intercepting proxy tool, Portswigger Burp Suite. In particular, this traffic included a POST request to an API endpoint and function that creates alerts on Synthethics monitors. He found that you could trap a GET request from the non-privileged session, retain the tokens and cookies for that restricted user, and alter the remainder of the trapped request to resemble the privileged POST request. This manipulation of API traffic to access functionality not visible in the UI (at all or to that user and their permissions) is a common technique attackers use to exploit function level authorization weaknesses and escalate privileges.

168

API5:2023 – Broken Function Level Authorization (BFLA) – Prevent

- Traditional security controls like WAFs and API gateways lack context of API activity and therefore do not know that the attacker in the example above should not be able to send a DELETE method.
- This API call would be seen as legitimate and would pass through these security controls. WAFs and API gateways sometimes support explicit, statically defined message filters, often referred to as a positive security approach.

169

API5:2023 – Broken Function Level Authorization (BFLA) – Prevent

- Restricting HTTP methods is also an easier task than restricting API parameters and values, the latter of which requires deeper subject matter expertise on the design of the API.
- API protection solutions should baseline typical HTTP access patterns per API endpoint and per user to identify calls with unexpected HTTP methods to specific API endpoints in order to prevent attackers from accessing unauthorized functionality and/or admin level capabilities.

170

API5:2023 – Broken Function Level Authorization (BFLA) – Prevent

- The enforcement mechanism(s) should deny all access by default, requiring explicit grants to specific roles for access to every function.
- Review your API endpoints against function level authorization flaws, while keeping in mind the business logic of the application and groups hierarchy.
- Make sure that administrative functions inside a regular controller implement authorization checks based on the user's group and role.

171

API6:2023 – Unrestricted Access to Sensitive Business Flows

- APIs vulnerable to this risk expose a business flow – such as buying a ticket, or posting a comment – without compensating for how the functionality could harm the business if used excessively in an automated manner.
- This doesn't necessarily come from implementation bugs.

172

API6:2023 – Unrestricted Access to Sensitive Business Flows

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Widespread** : Detectability **Average**	Technical **Moderate** : Business Specific
Exploitation usually involves understanding the business model backed by the API, finding sensitive business flows, and automating access to these flows, causing harm to the business.	Lack of a holistic view of the API in order to fully support business requirements tends to contribute to the prevalence of this issue. Attackers manually identify what resources (e.g. endpoints) are involved in the target workflow and how they work together. If mitigation mechanisms are already in place, attackers need to find a way to bypass them.	In general technical impact is not expected. Exploitation might hurt the business in different ways, for example: prevent legitimate users from purchasing a product, or lead to inflation in the internal economy of a game.

173

API6:2023 – Unrestricted Access to Sensitive Business Flows

- APIs vulnerable to this risk expose a business flow – such as buying a ticket, or posting a comment – without compensating for how the functionality could harm the business if used excessively in an automated manner.
- This doesn't necessarily come from implementation bugs.

174

API6:2023 – Unrestricted Access to Sensitive Business Flows

Scenario #2

An airline company offers online ticket purchasing with no cancellation fee. A user with malicious intentions books 90% of the seats of a desired flight.

A few days before the flight the malicious user canceled all the tickets at once, which forced the airline to discount the ticket prices in order to fill the flight.

At this point, the user buys herself a single ticket that is much cheaper than the original one.

175

API6:2023 – Unrestricted Access to Sensitive Business Flows – Prevent

- Business – identify the business flows that might harm the business if they are excessively used.
- Engineering – choose the right protection mechanisms to mitigate the business risk.
- Secure and limit access to APIs that are consumed directly by machines (such as developer and B2B APIs). They tend to be an easy target for attackers because they often don't implement all the required protection mechanisms.

176

API6:2023 – Unrestricted Access to Sensitive Business Flows – Prevent

- Some of the protection mechanisms are more simple while others are more difficult to implement. The following methods are used to slow down automated threats:
  - Device fingerprinting: denying service to unexpected client devices (e.g headless browsers) tends to make threat actors use more sophisticated solutions, thus more costly for them
  - Non-human patterns: analyze the user flow to detect non-human patterns (e.g. the user accessed the "add to cart" and "complete purchase" functions in less than one second)
  - Consider blocking IP addresses of Tor exit nodes and well-known proxies

177

API7:2023 Server Side Request Forgery

- Server-Side Request Forgery (SSRF) flaws can occur when an API is fetching a remote resource without validating the user-supplied URI. This enables an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall or a VPN.

178

API7:2023 Server Side Request Forgery

As part of a creation of a new webhook, a GraphQL mutation is sent with the URL of the SIEM API.

```
POST /graphql

[
  {
    "variables": {},
    "query": "mutation {
      createNotificationChannel(input: {
        channelName: \"ch_piney\",
        notificationChannelConfig: {
          customWebhookChannelConfigs: [
            {
              url: \"http://www.siem-system.com/create_new_event\",
              send_test_req: true
            }
          ]
          }
      }){
        channelId
    }
    }"
  }
]
```

179

API7:2023 Server Side Request Forgery

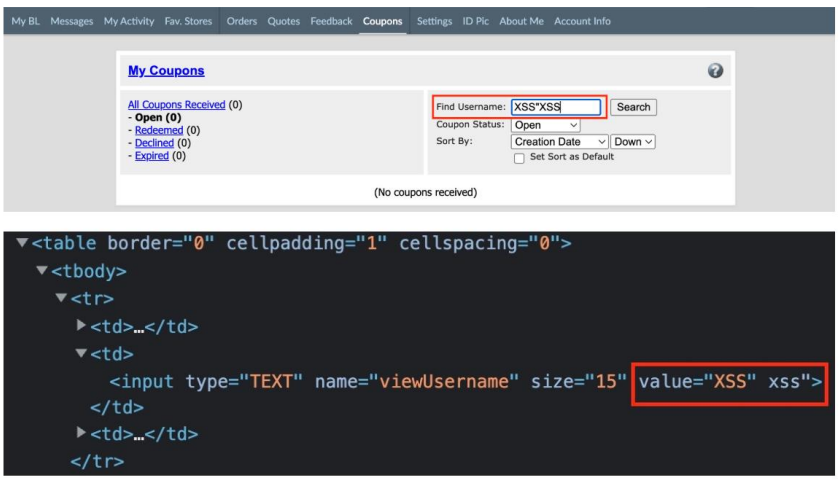
An attacker can leverage this flow, and make the API request a sensitive resource, such as an internal cloud metadata service that exposes credentials:

```
POST /graphql

[
  {
    "variables": {},
    "query": "mutation {
      createNotificationChannel(input: {
        channelName: \"ch_piney\",
        notificationChannelConfig: {
          customWebhookChannelConfigs: [
            {
              url: \"http://169.254.169.254/latest/meta-data/iam/security-credentials/ec
              send_test_req: true
            }
          ]
        }
      }) {
        channelId
      }
    }
  }
]
```

180

API7:2023 Server Side Request Forgery

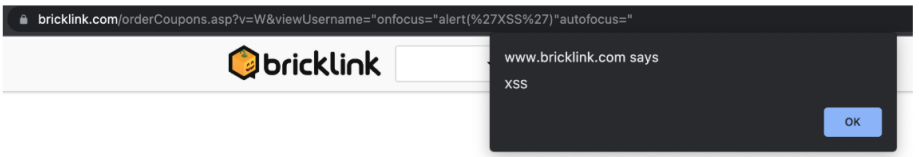



```
▼<table border="0" cellpadding="1" cellspacing="0">
  ▼<tbody>
    ▼<tr>
      ▶<td>…</td>
      ▼<td>
        <input type="TEXT" name="viewUsername" size="15" value="XSS" xss">
      </td>
      ▶<td>…</td>
    </tr>
```

181

API7:2023 Server Side Request Forgery

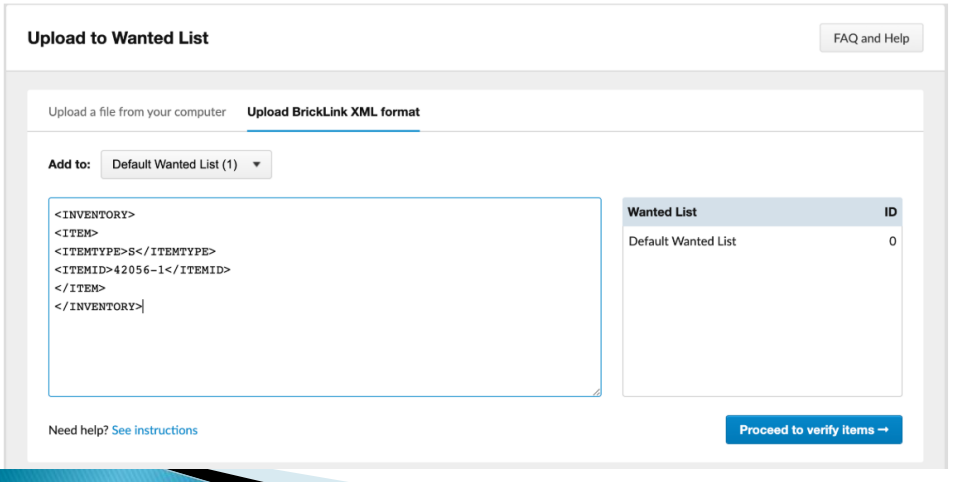
With the input **"onfocus="alert('XSS')"autofocus="** an alert popped, proving that the JavaScript code was indeed injected and executed on the page.



182

API7:2023 Server Side Request Forgery

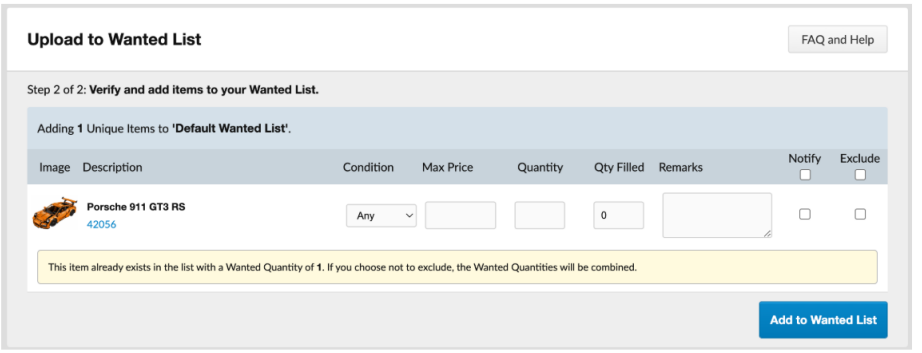
The user inputs the wanted set in the correct format (XML):



183

API7:2023 Server Side Request Forgery

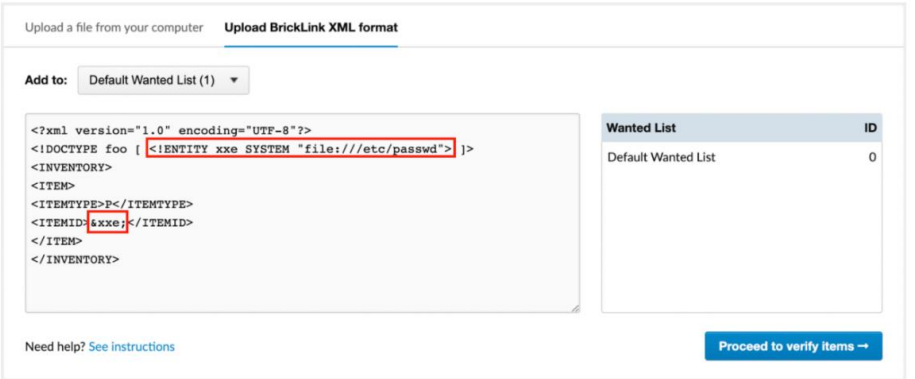
And after clicking **“Proceed”** the list is created:



184

API7:2023 Server Side Request Forgery

To test for a vulnerable condition, I added an XML External Entity that refers to **/etc/passwd/** and entered its content in the **Item ID** field.



185

API7:2023 Server Side Request Forgery

I clicked "**Proceed**" and got the following error back from the server:

Boom! We can see some of the content of **/etc/passwd** in the error message.

186

API7:2023 Server Side Request Forgery Prevent

- Isolate the resource fetching mechanism in your network: usually these features are aimed to retrieve remote resources and not internal ones.
- Whenever possible, use allow lists of:
  - Remote origins users are expected to download resources from (e.g. Google Drive, Gravatar, etc.)
  - URL schemes and ports
  - Accepted media types for a given functionality

187

API7:2023 Server Side Request Forgery Prevent

- Disable HTTP redirections.
- Use a well-tested and maintained URL parser to avoid issues caused by URL parsing inconsistencies.
- Validate and sanitize all client-supplied input data.
- Do not send raw responses to clients.

188

API8:2023 – Security Misconfiguration

- APIs and the systems supporting them typically contain complex configurations, meant to make the APIs more customizable.
- Software and DevOps engineers can miss these configurations, or don't follow security best practices when it comes to configuration, opening the door for different types of attacks.
- Detailed errors can expose sensitive user data and system details that may lead to full server compromise

189

API8:2023 – Security Misconfiguration

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Widespread** : Detectability **Easy**	Technical **Severe** : Business Specific
Attackers will often attempt to find unpatched flaws, common endpoints, services running with insecure default configurations, or unprotected files and directories to gain unauthorized access or knowledge of the system. Most of this is public knowledge and exploits may be available.	Security misconfiguration can happen at any level of the API stack, from the network level to the application level. Automated tools are available to detect and exploit misconfigurations such as unnecessary services or legacy options.	Security misconfigurations not only expose sensitive user data, but also system details that can lead to full server compromise.

190

API8:2023 – Security Misconfiguration

Legitimate – Client sends a legitimate request

```
GET /api/v2/network/connections/593065
HTTP/1.1
Accept: application/json, text/plain, */*
Accept-Encoding: gzip

HTTP/1.1 200 OK
{
"status":”success",
}
```

Attack – Attackers modify the connectionId resulting in a detailed exception error

```
GET /api/v2/network/connections/5930aaaaa
HTTP/1.1
Accept: application/json, text/plain, */*
Accept-Encoding: gzip

HTTP/1.1 500 Server Error
{
"status":"failure",
"statusMessage":"An error occurred while
validating input: validation error: unexpected
content \"593065d1\"
({com.tibco.xml.validation}COMPLEX_E_UNEXPECTED_C
ONTENT) at
/{http://www.tibco.com/namespaces/tnt/plugins/jso
n}ActivityOutputClass[1]/searchSvcReqsByRepReq[1]
/search[1]/status[1]/aaaa[1]\ncom.tibco.xml.valid
ation.exception.UnexpectedElementException:
unexpected content \"aaaa\"&#xD;\n\tat
com.tibco.xml.validation.state.a.a.a(CMElementVal
idationState.java:476)&#xD;\n\tat
com.tibco.xml.validation.state.a.a.a(CMElementVal
idationState.java:270)&#xD;\n\tat
com.tibco.xml.validation.state.driver.ValidationJ
azz.c(ValidationJazz.java:993)&#xD;\n\tat
com.tibco.xml.validation.state.driver.ValidationJ
azz.b(ValidationJazz.java:898)&#xD;\n\tat ….
```

191

API8:2023 – Security Misconfiguration

A social network website offers a "Direct Message" feature that allows users to keep private conversations. To retrieve new messages for a specific conversation, the website issues the following API request (user interaction is not required):

```
GET /dm/user_updates.json?conversation_id=1234567&cursor=GRlFp7LCUAAAA
```

Because the API response does not include the `Cache-Control` HTTP response header, private conversations end-up cached by the web browser, allowing malicious actors to retrieve them from the browser cache files in the filesystem.

192

API8:2023 – Security Misconfiguration Prevent

- A repeatable hardening process leading to fast and easy deployment of a properly locked down environment
- A task to review and update configurations across the entire API stack. The review should include: orchestration files, API components, and cloud services (e.g. S3 bucket permissions)
- An automated process to continuously assess the effectiveness of the configuration and settings in all environments
- Restrict incoming content types/data formats to those that meet the business/ functional requirements

193

API9:2023 – Improper Inventory Management

- APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important.
- A proper inventory of hosts and deployed API versions also are important to mitigate issues such as deprecated API versions and exposed debug endpoints.

194

API9:2023 – Improper Inventory Management

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Widespread** : Detectability **Average**	Technical **Moderate** : Business Specific
Threat agents usually get unauthorized access through old API versions or endpoints left running unpatched and using weaker security requirements. In some cases exploits are available. Alternatively, they may get access to sensitive data through a 3rd party with whom there's no reason to share data with.	Outdated documentation makes it more difficult to find and/or fix vulnerabilities. Lack of assets inventory and retirement strategies leads to running unpatched systems, resulting in leakage of sensitive data. It's common to find unnecessarily exposed API hosts because of modern concepts like microservices, which make applications easy to deploy and independent (e.g. cloud computing, K8S). Simple Google Dorking, DNS enumeration, or using specialized search engines for various types of servers (webcams, routers, servers, etc.) connected to the internet will be enough to discover targets.	Attackers can gain access to sensitive data, or even take over the server. Sometimes different API versions/deployments are connected to the same database with real data. Threat agents may exploit deprecated endpoints available in old API versions to get access to administrative functions or exploit known vulnerabilities.

195

API9:2023 – Improper Inventory Management

The Optus Breach

In September 2022, media reports indicated that the second largest telecommunications company in Australia, Optus, had suffered a data breach caused by a security vulnerability in one of the company's APIs and resulting in the exposure of 11.2 million customer records, including personal identifiable information. The API targeted by the attackers was used to manage customer accounts, and the vulnerability allowed unauthorized access to sensitive information, including names, addresses, and phone numbers.

The Optus breach highlights the importance of proper API inventory management, specifically the need for regular auditing and risk assessments. When APIs are not properly managed, vulnerabilities can go undetected, and unauthorized access can occur. In this case, the vulnerability was not discovered until it had already been exploited, putting thousands of customers' personal information at risk.

Adding to the severe reputational damage caused by the high-profile breach, Optus has reportedly put aside A$140 million to cover the costs of the incident.

196

API9:2023 – Improper Inventory Management – Prevent

- Inventory all API hosts and document important aspects of each one of them, focusing on the API environment (e.g. production, staging, test, development), who should have network access to the host (e.g. public, internal, partners) and the API version.
- Inventory integrated services and document important aspects such as their role in the system, what data is exchanged (data flow), and their sensitivity.
- Document all aspects of your API such as authentication, errors, redirects, rate limiting, cross-origin resource sharing (CORS) policy, and endpoints, including their parameters, requests, and responses.

197

API9:2023 – Improper Inventory Management – Prevent

- Generate documentation automatically by adopting open standards. Include the documentation build in your CI/CD pipeline.
- Make API documentation available only to those authorized to use the API.
- Use external protection measures such as API security specific solutions for all exposed versions of your APIs, not just for the current production version.
- When newer versions of APIs include security improvements, perform a risk analysis to inform the mitigation actions required for the older versions.

198

API10:2023 – Unsafe Consumption of APIs

- Developers tend to trust data received from third-party APIs more than user input, and so tend to adopt weaker security standards.
- In order to compromise APIs, attackers go after integrated third-party services instead of trying to compromise the target API directly.

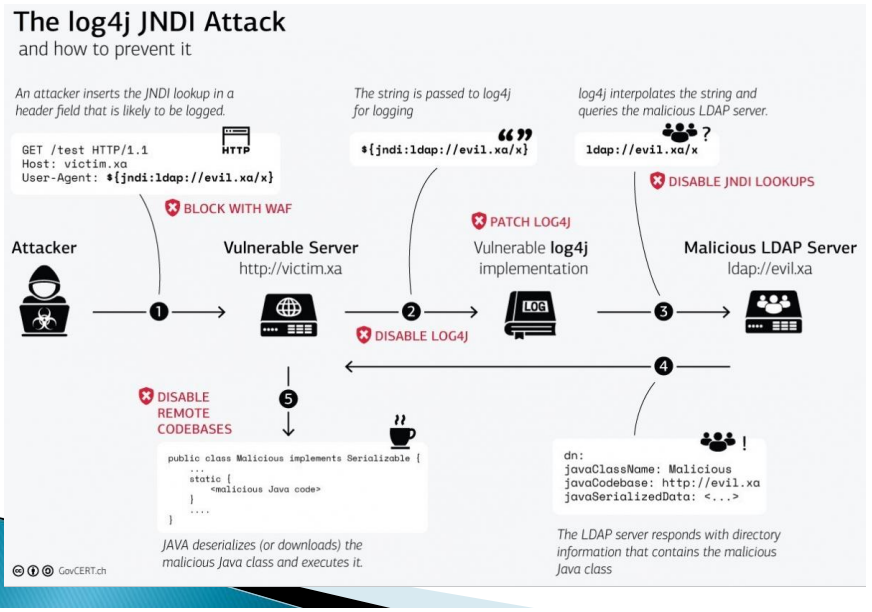
199

API10:2023 – Unsafe Consumption of APIs

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Common** : Detectability **Average**	Technical **Severe** : Business Specific
Exploiting this issue requires attackers to identify and potentially compromise other APIs/services the target API integrated with. Usually, this information is not publicly available or the integrated API/service is not easily exploitable.	Developers tend to trust and not verify the endpoints that interact with external or third-party APIs, relying on weaker security requirements such as those regarding transport security, authentication/authorization, and input validation and sanitization. Attackers need to identify services the target API integrates with (data sources) and, eventually, compromise them.	The impact varies according to what the target API does with pulled data. Successful exploitation may lead to sensitive information exposure to unauthorized actors, many kinds of injections, or denial of service.

200

API10:2023 – Unsafe Consumption of APIs



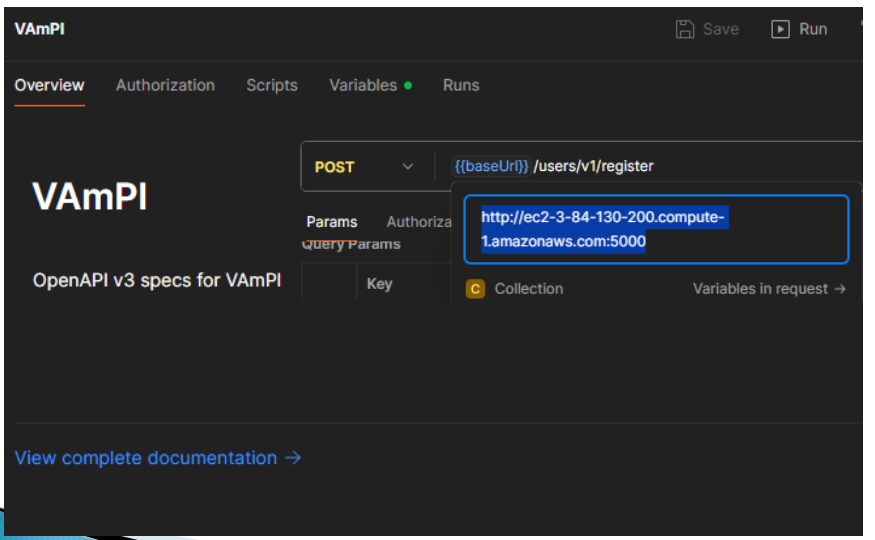
201

API10:2023 – Unsafe Consumption of APIs – Prevent

- When evaluating service providers, assess their API security posture.
- Ensure all API interactions happen over a secure communication channel (TLS).
- Always validate and properly sanitize data received from integrated APIs before using it.
- Maintain an allowlist of well-known locations integrated APIs may redirect yours to: do not blindly follow redirects.

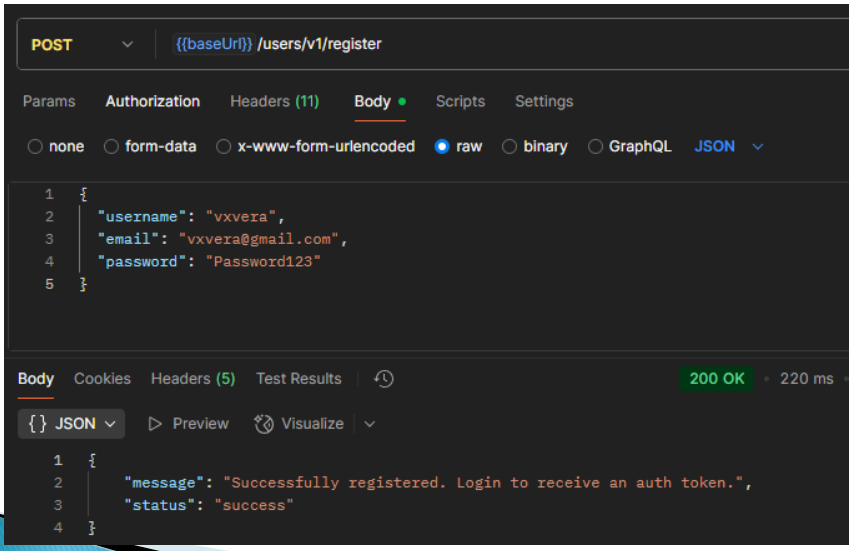
202

Testing



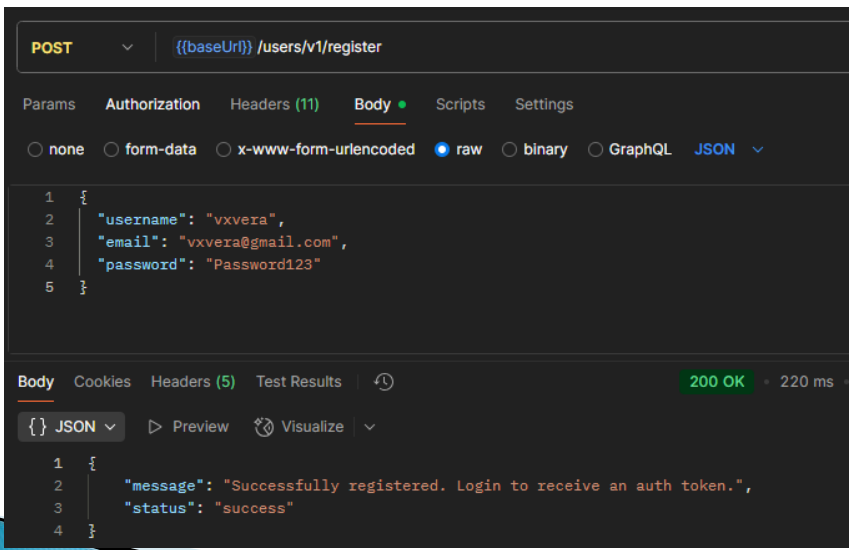
203

Testing



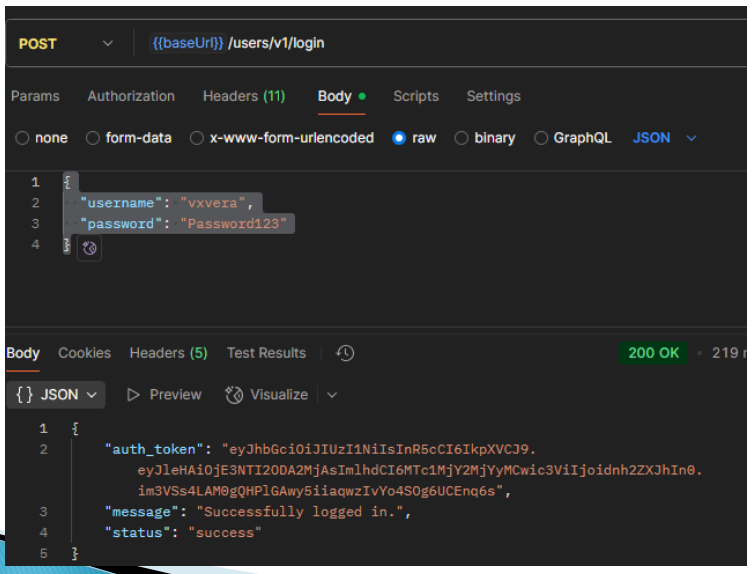
204

Testing



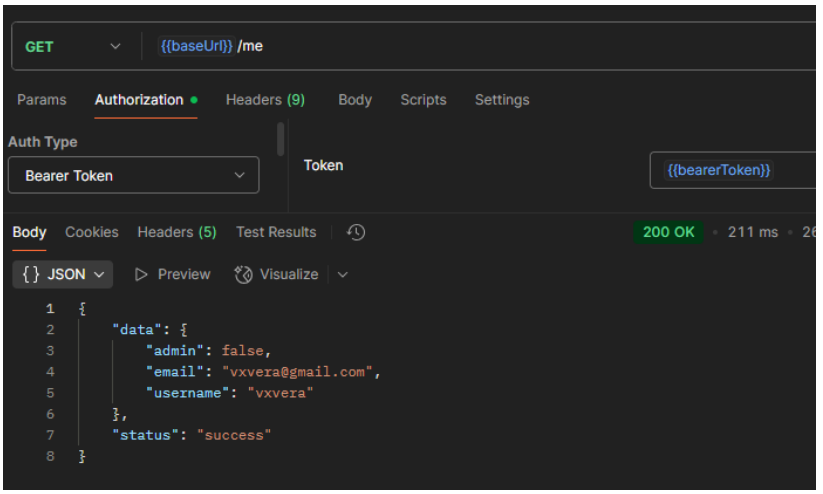
205

Testing



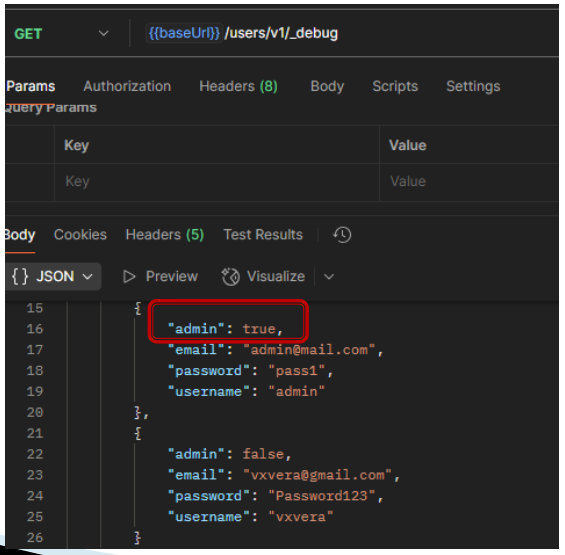
206

Testing



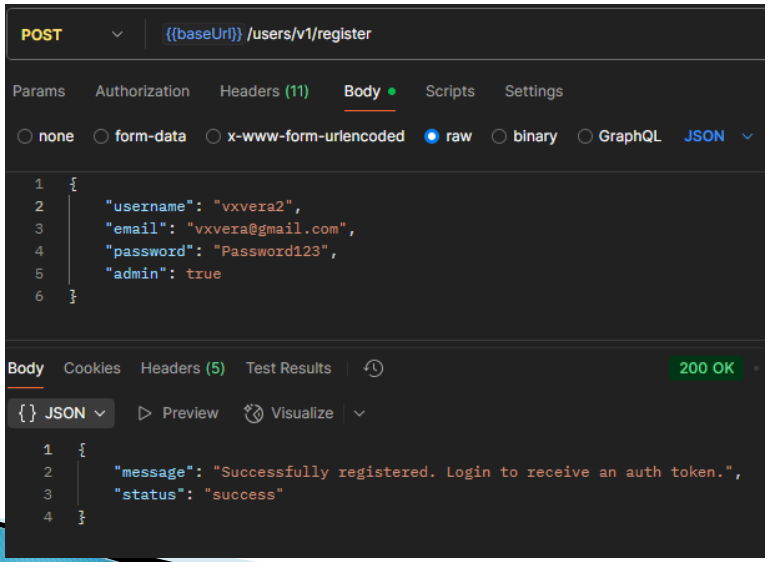
207

Testing – API3:2023 – Broken Object Property Level Authorization



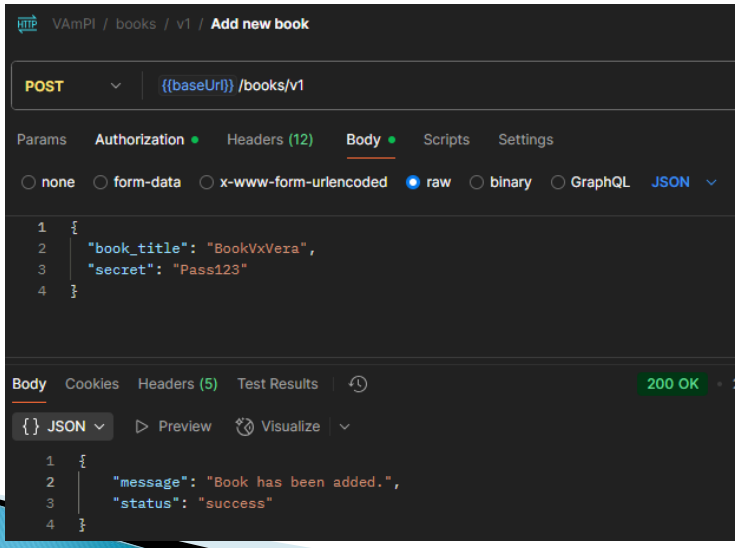
208

Testing – API3:2023 – Broken Object Property Level Authorization



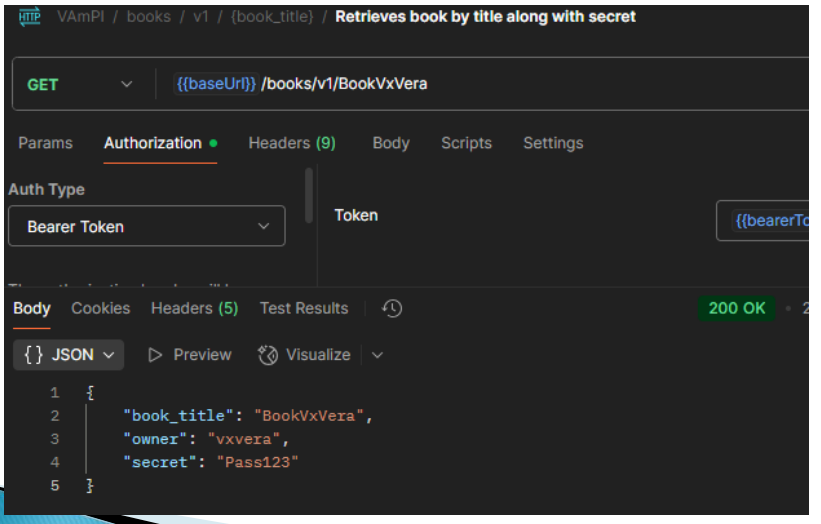
209

Testing – API1:2023 – Broken Object Level Authorization



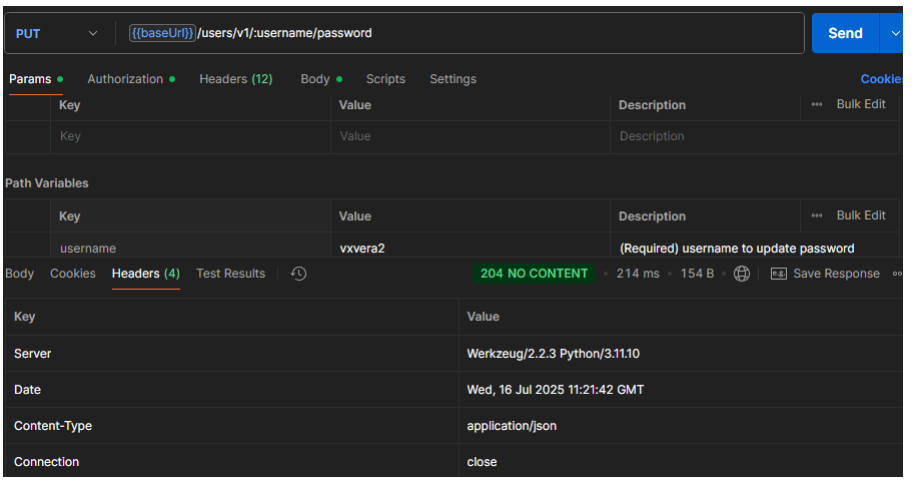
210

Testing – API1:2023 – Broken Object Level Authorization



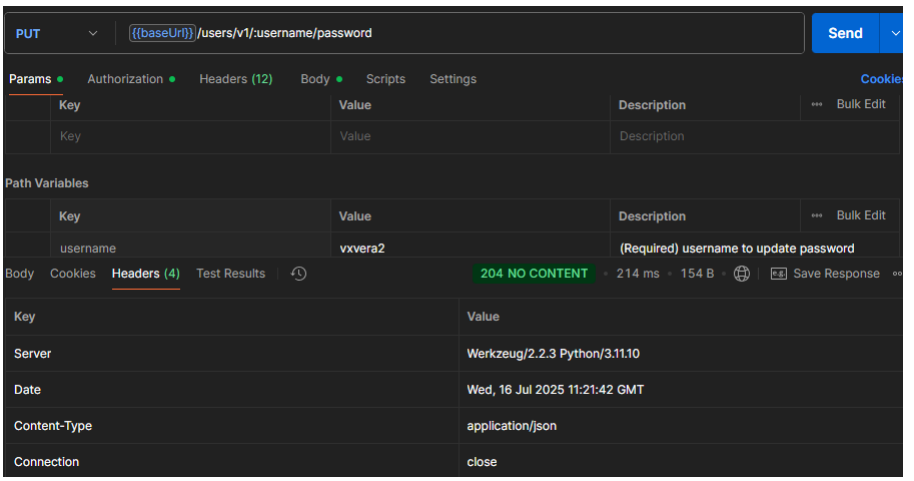
211

Testing – API5:2023 – Broken Function Level Authorization (BFLA)



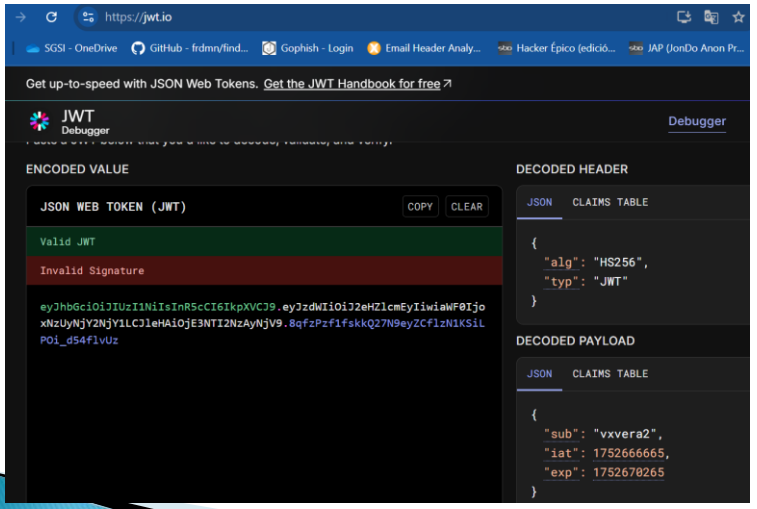
212

Testing – API5:2023 – Broken Function Level Authorization (BFLA)



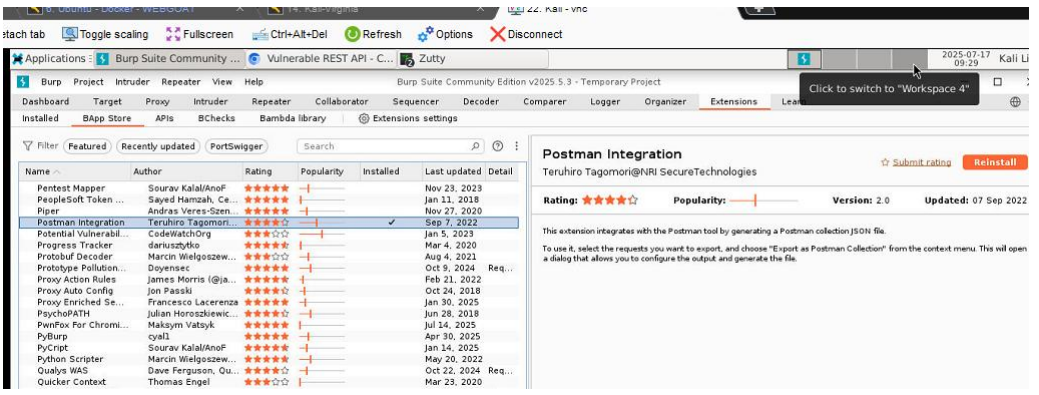
213

Testing – API5:2023 – Broken Function Level Authorization (BFLA)



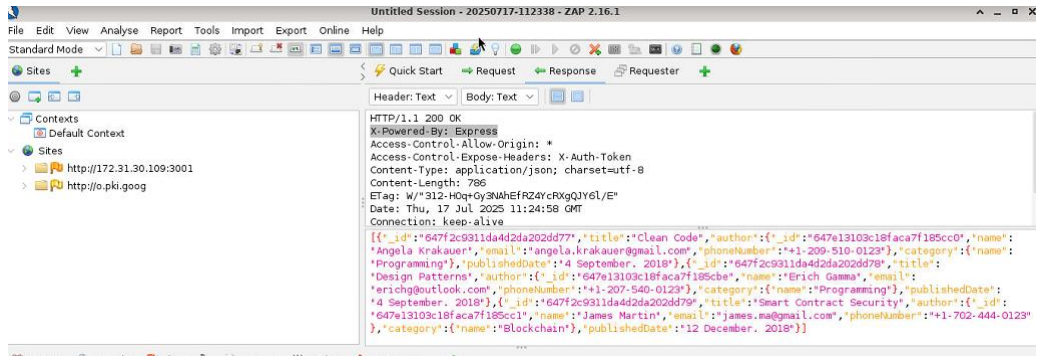
214

Testing

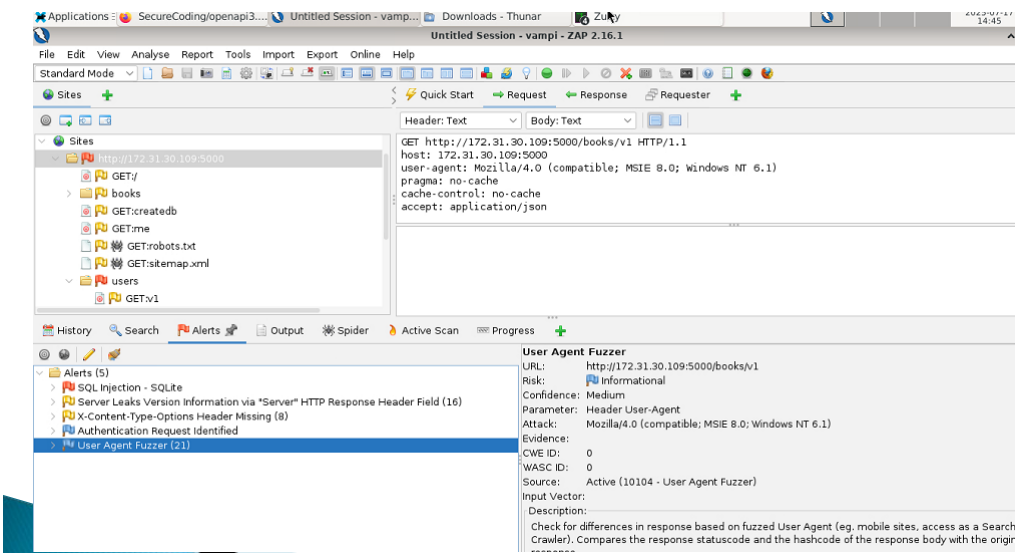


215

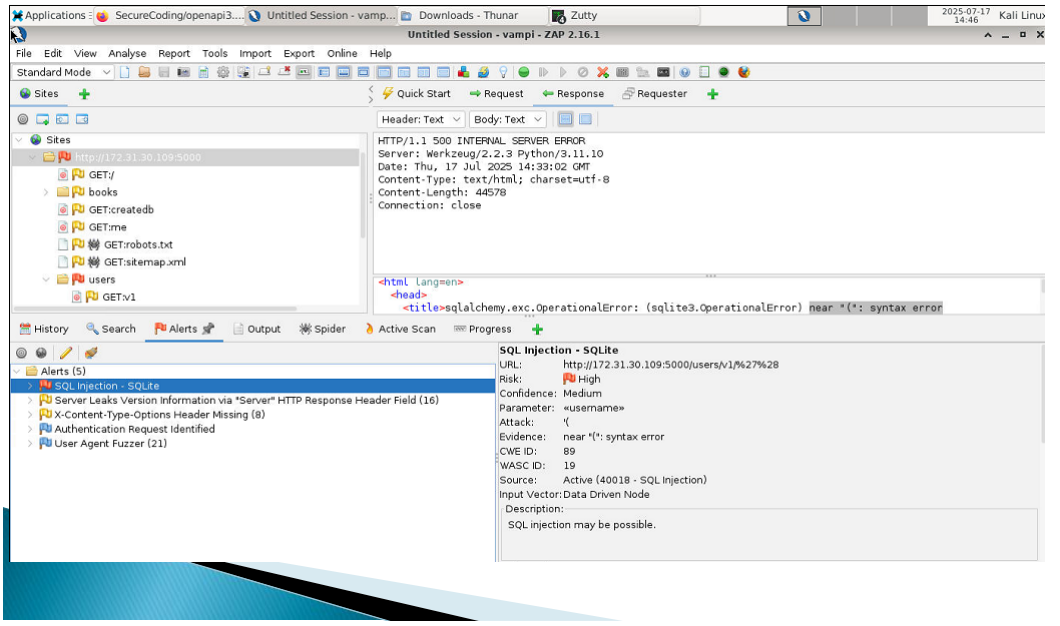
Testing



Testing



Testing



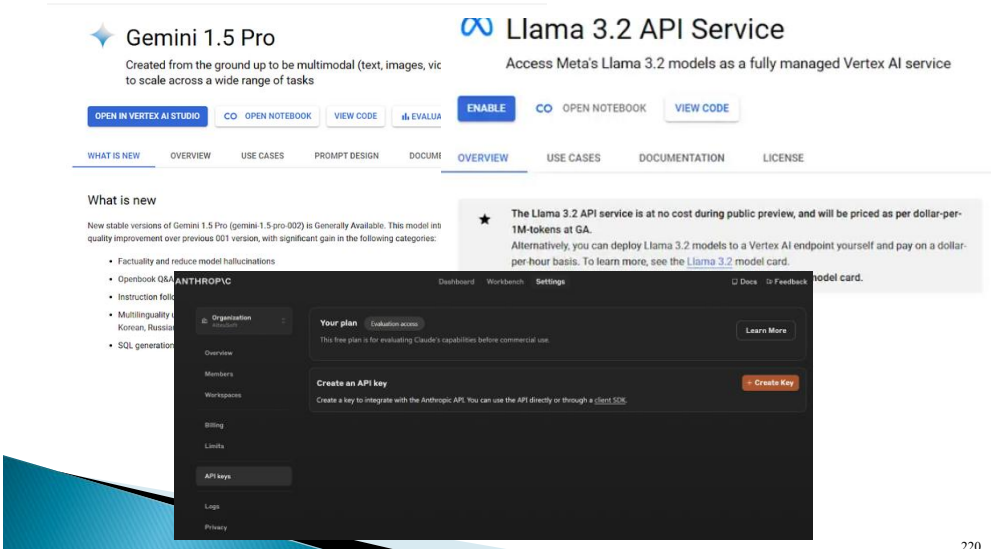
218

Secure Coding Test LLM

SEGTIUM
INGENIERIA EN CIBERSEGURIDAD

219

LLMs – APIS



220

LLMs – APIS

	GPT4-o	Gemini 1.5 Pro	Llama 3.2 herd	Claude 3.5 Sonnet	Grok beta	Mistral Large 2
General characteristics						
Last trained	Oct 2023	Undisclosed	Dec 2023	Apr 2024	Undisclosed	Undisclosed
Parameters (billion)	1,800*	1,560*	1, 3, 11, 90	500*	314+	123
Languages	50+	100+	8	10+	Undisclosed	10+
Max input (thousand tokens)	128	2,000	128	200	128	128

221

LLMs APIS

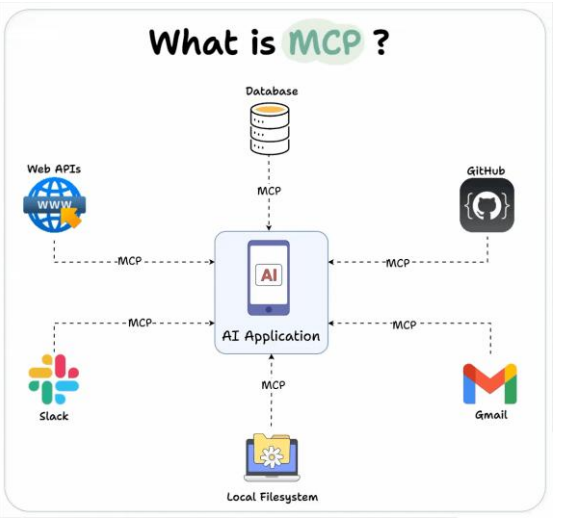
Model Context Protocol (MCP)

- Desarrollado por Anthropic e introducido en noviembre de 2024.
- Estándar abierto (JSON-RPC 2.0) que define cómo los LLMs pueden interactuar con fuentes de datos y herramientas externas (archivos, bases de datos, APIs, plugins)
- Adoptado por OpenAI, Google DeepMind, Microsoft Copilot Studio, Wix, Sourcegraph y otras plataforMAS

222

LLMs APIS

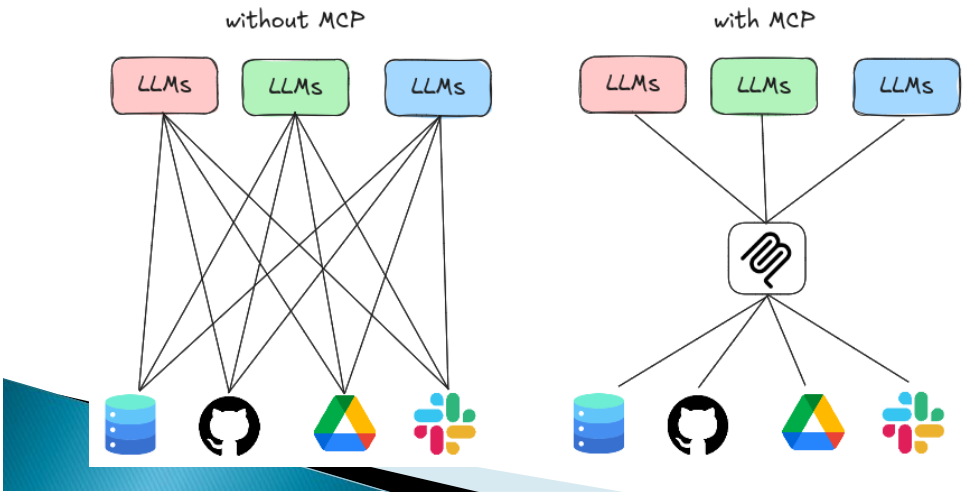
Model Context Protocol (MCP)



223

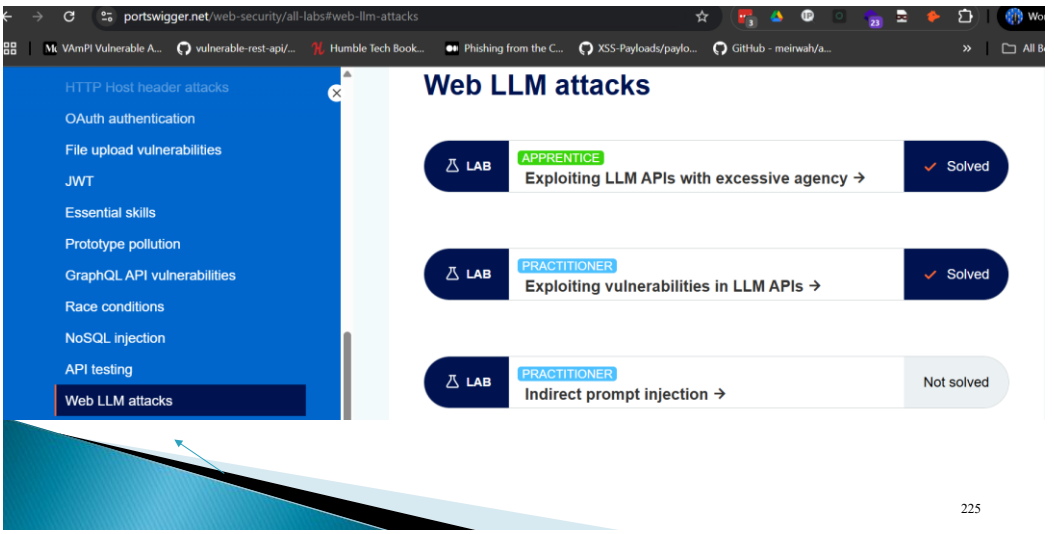
LLMs APIS

Model Context Protocol (MCP)



224

LLMs – APIS – Testing



225

Recursos OWASP

- C1: Valida los datos de entrada
- C2: Gestiona autenticación y contraseñas correctamente
- C3: Implementa control de acceso a nivel de función y objeto
- C4: Define políticas de cifrado y anonimización
- C5: Protege contra inyección (SQL, XSS, comandos)
- C6: Limpia y analiza las salidas del sistema
- C7: Gestiona sesiones y tokens de manera segura
- C8: Log y monitoreo de seguridad en tiempo real
- C9: Maneja errores correctamente sin filtrar información
- C10: Escribe pruebas de seguridad automatizadas (SAST, DAST)

226

GRACIAS