Seguridad en APIs

Por: Daniel Ojalvo Canedo

Datos personales

- ✓ Daniel Ojalvo Canedo
- ✓ Ingeniero Informático
- ✓ Magister Seguridad de Tecnologías de la información
- ✓ danielojalvo037gmail.com
- ✓ https://www.linkedin.com/in/daniel-ojalvo/
- ✓ 72464162

API

Las API son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. Por ejemplo, el sistema de software del instituto de meteorología contiene datos meteorológicos diarios. La aplicación meteorológica de su teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en su teléfono.

Por: Daniel Ojalvo Canedo

API

XML

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

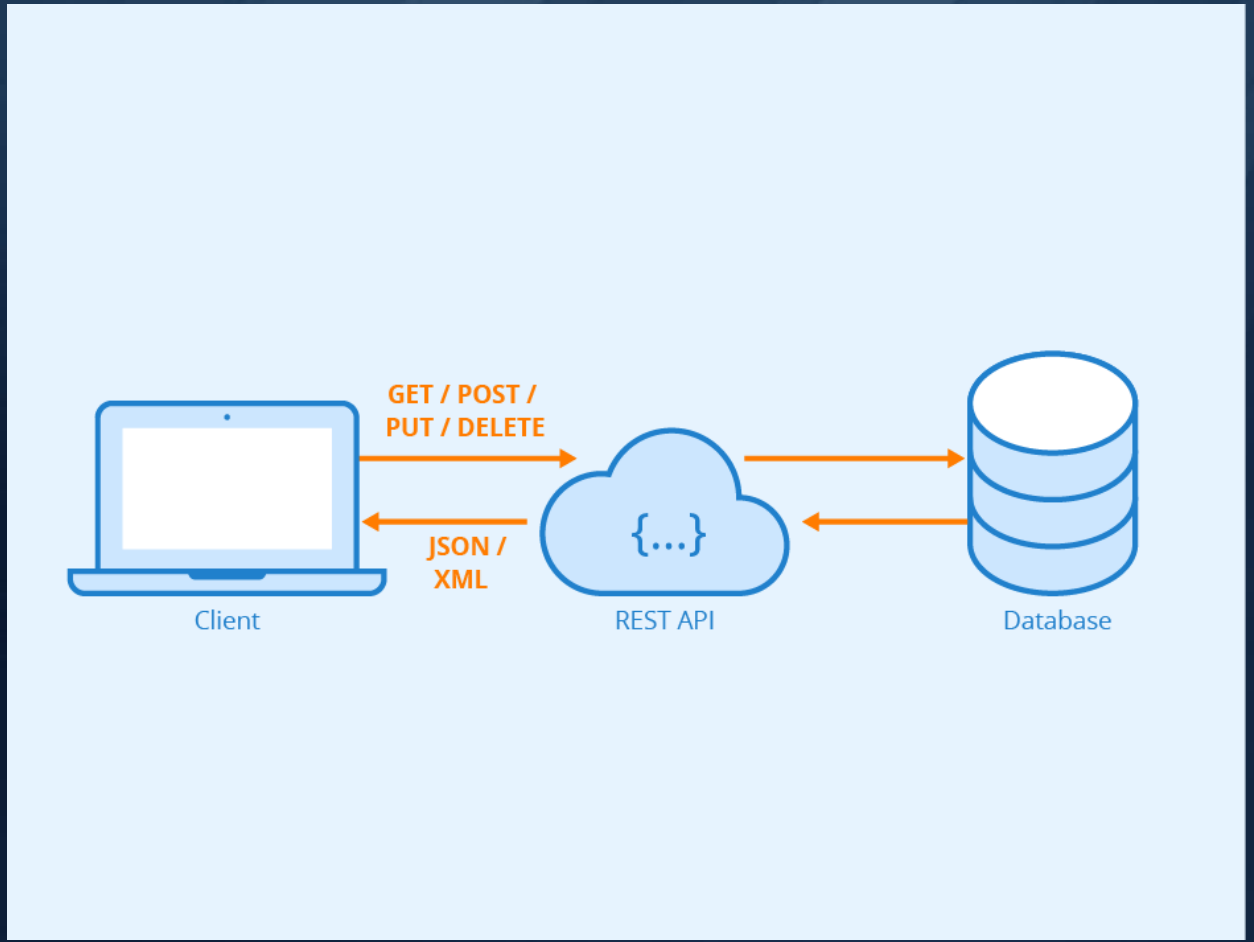
```
▼<NewDataSet>
  ▼<Table1>
     <Id>1203</Id>
     <Title>Test Discussion, Dont Reply</Title>
     <CreatedDate>4/25/2014 10:24:56 AM</CreatedDate>
     <Status>Not Sent</Status>
   </Table1>
  ▼<Table1>
     <Id>1182</Id>
     <Title>Negotiation Skills discussion</Title>
     <CreatedDate>4/25/2014 7:47:51 AM</CreatedDate>
     <Status>Not Sent</Status>
   </Table1>
 </NewDataSet>
```

JSON

```
{
    "res": true,
    "message": "Sesión iniciada",
    "usuario": {
        "id": 70,
        "nrodocumento": "7578800",
        "nombres": "JUAN",
        "primerapellido": "PEREZ",
        "segundoapellido": "AGUILAR",
        "created_at": "2022-01-18 09:57:09",
        "updated_at": "2022-02-08 14:58:20"
    }
}
```

Por: Daniel Ojalvo Canedo

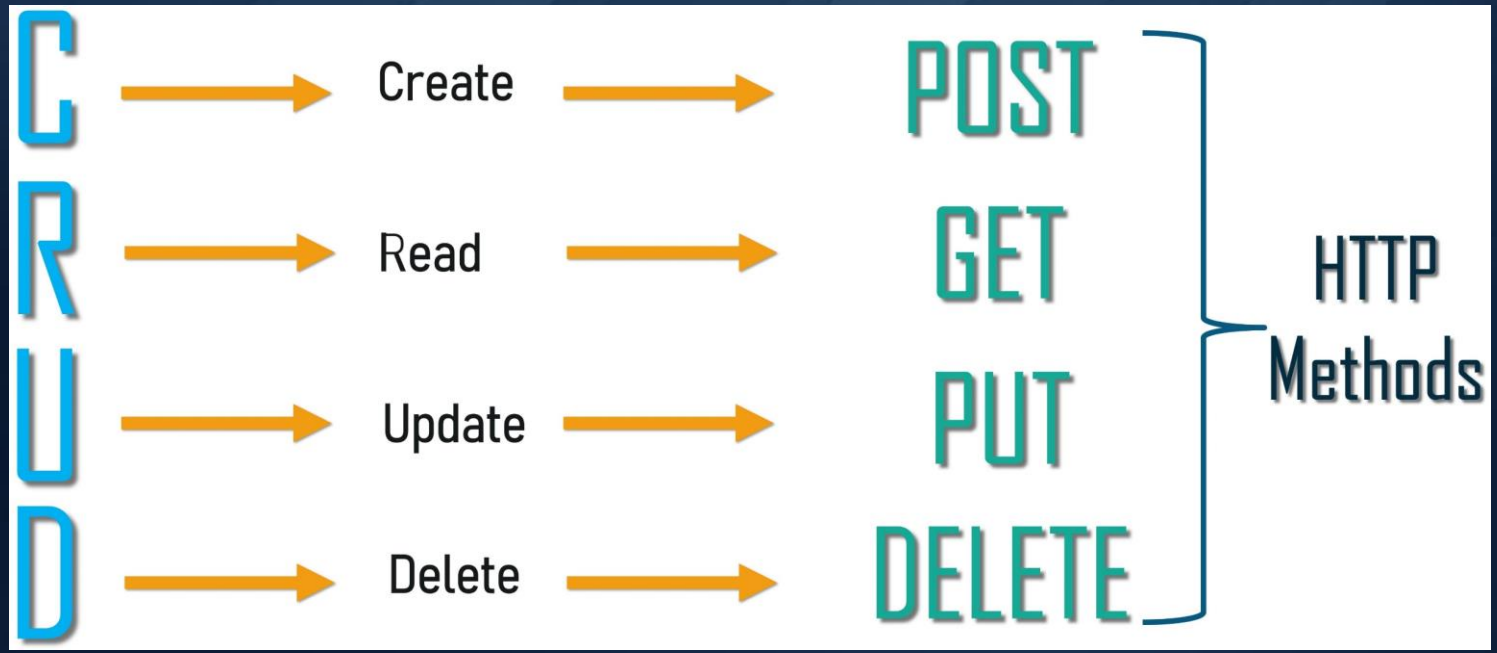
API



Por: Daniel Ojalvo Canedo

API

Funcionamiento



Por: Daniel Ojalvo Canedo

Riesgos en las API

Por: Daniel Ojalvo Canedo

API 1: Autorización de nivel de objeto vulnerable

Las API tienden a exponer puntos finales (end-points) que manejan identificadores de objetos, creando un problema de Control de Acceso y ampliando la superficie de ataque. En cada función que accede a datos utilizando una entrada del usuario, se deben considerarse agregar verificaciones adicionales de autorización a nivel de objeto.

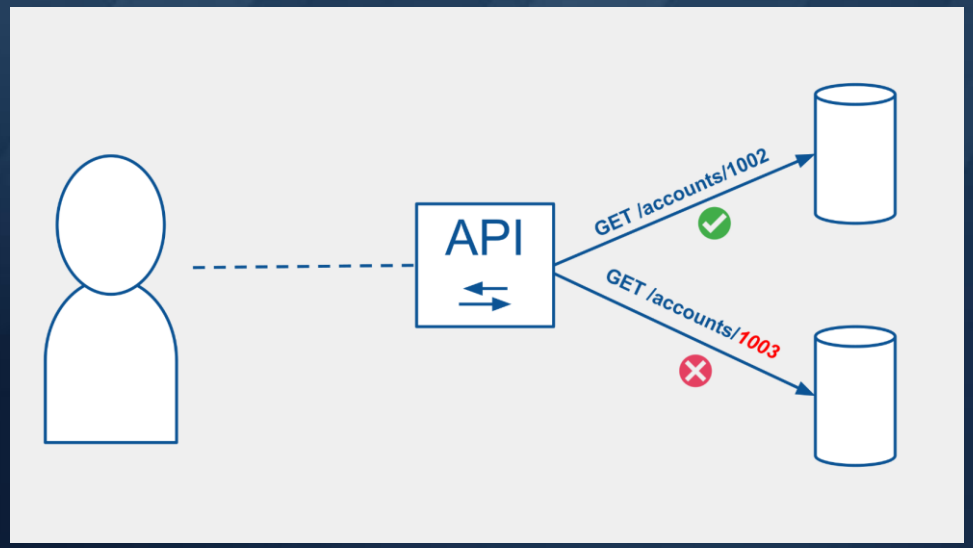
Ejemplo: /api/products/{productID}/create

Las fallas en este mecanismo generalmente conducen a la divulgación no autorizada de información, modificación o destrucción de todos los datos.

Por: Daniel Ojalvo Canedo

API 1: Autorización de nivel de objeto vulnerable

Ejemplo:




```
app.get("/accounts/:accountId", authenticateToken, (req, res) => {
  const { accountId } = req.params;

  // A mock database query
  const account = db.accounts.filter((a) => {
    return a.accountId === accountId;
  })[0];

  res.json(account);
});
```

Por: Daniel Ojalvo Canedo

API 1: Autorización de nivel de objeto vulnerable

Prevención:

- ✓ Implementar un mecanismo de autorización adecuado que se base en las políticas y la jerarquía del usuario.
- ✓ En cada función que utiliza una entrada del cliente para acceder a un registro de la base de datos, se debe usar un mecanismo de autorización para verificar si el usuario conectado tiene acceso para realizar la acción solicitada en el registro.
- ✓ Se debería usar valores aleatorios e impredecibles como GUID para las ID de los registros.
- ✓ Se deben escribir pruebas para evaluar el mecanismo de autorización.

Por: Daniel Ojalvo Canedo

API 2: Autenticación de usuario vulnerable

A menudo, los mecanismos de autenticación se implementan incorrectamente, lo que permite a los atacantes comprometer los tokens de autenticación o explotar fallas de implementación para asumir las identidades de otros usuarios de manera temporal o permanente. La incapacidad del sistema para identificar al cliente/usuario compromete la seguridad de la API en general.

Por: Daniel Ojalvo Canedo

API 2: Autenticación de usuario vulnerable

Ejemplo: /api/system/verification-codes/{smsToken}

Debido a que la API no implementa una política de limitación de velocidad, el atacante puede probar todas las combinaciones posibles utilizando un script de subprocesos múltiples, contra el punto final para descubrir el token correcto en unos minutos.

Por: Daniel Ojalvo Canedo

API 2: Autenticación de usuario vulnerable

Prevención:

- ✓ No reinventar la rueda de la autenticación, generación de tokens, almacenamiento de contraseñas. Usar los estándares.
- ✓ Los puntos finales de recuperación de credenciales y "olvido de contraseña" deben tratarse como end-points de inicio de sesión en términos de fuerza bruta, limitación de velocidad y protecciones de bloqueo.
- ✓ Donde sea posible, implementar la autenticación multifactor (MFA).
- ✓ Implementar mecanismos anti-fuerza bruta para mitigar el relleno de credenciales automático, ataques de diccionario y de fuerza bruta en los end-points de autenticación. Este mecanismo debería ser más estricto que el mecanismo de limitación de velocidad en la API.
- ✓ Implementar un mecanismo de bloqueo o recuperación de cuenta para evitar la fuerza bruta contra usuarios específicos. Implemente verificaciones de contraseña débil y un CAPTCHA.

Por: Daniel Ojalvo Canedo

API 3: Exposición excesiva de datos

La API devuelve datos confidenciales al cliente por diseño. Estos datos generalmente se filtran en el lado del cliente antes de presentarlos al usuario. Un atacante puede olfatear fácilmente el tráfico y ver los datos confidenciales.

Por: Daniel Ojalvo Canedo

API 3: Exposición excesiva de datos

Ejemplo:

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

Por: Daniel Ojalvo Canedo

API 3: Exposición excesiva de datos

Prevención:

- ✓ Nunca confiar en el cliente para filtrar datos confidenciales.
- ✓ Revisar las respuestas de la API para asegurarse de que solo contengan los datos necesarios.
- ✓ Los ingenieros de backend siempre deben preguntarse "¿quién es el consumidor de los datos?" antes de exponer un nuevo end-point API.
- ✓ Clasificar la información confidencial y de identificación personal (PII) y revisar todas las llamadas API que devuelven dicha información para ver si estas respuestas plantean un problema de seguridad o confidencialidad.

Por: Daniel Ojalvo Canedo

API 4: Fallas en la restricción y limitación de recursos

Muy a menudo, las API no imponen ninguna restricción sobre el tamaño o la cantidad de recursos que puede solicitar el cliente. Esto no solo puede afectar el rendimiento del servidor API, lo que lleva a la denegación de servicio (DoS), sino que también deja la puerta abierta a fallas de autenticación a través de fuerza bruta.

Ejemplo: /api/users?page=1&size=200

Tenemos una aplicación que contiene la lista de usuarios en una interfaz de usuario con un límite de 200 usuarios por página. La lista de usuarios se recupera del servidor. Un atacante cambia el size parámetro a 200000, lo que provoca problemas de rendimiento en la base de datos. Mientras tanto, la API deja de responder y no puede manejar más solicitudes de este o cualquier otro cliente

Por: Daniel Ojalvo Canedo

API 4: Fallas en la restricción y limitación de recursos

Ejemplo:

Legitimate – max_return and page_size request attributes are normal	**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	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": [
 	
]
}
	{
	"search_filter":
	"user_id=exampleId_100",
	"max_return": "20000",
	"page_size": "20000",
	"return_attributes": [
	
	]
	}

Por: Daniel Ojalvo Canedo

API 4: Fallas en la restricción y limitación de recursos

Prevención:

- ✓ Implementar un límite sobre la frecuencia con que un cliente puede llamar a la API dentro de un marco de tiempo definido.
- ✓ Notificar al cliente cuando se excede el límite. proporcionando el número máximo y la hora a la que se restablecerá el límite.
- ✓ Validar adecuadamente (del lado del servidor) el formato de la consulta y los parámetros del cuerpo de la solicitud, específicamente el relacionado al número de registros que se devolverán en cada respuesta.
- ✓ Definir y aplicar un tamaño máximo de datos en todos los parámetros entrantes y payloads, como la longitud máxima de las cadenas y el número máximo de elementos en las matrices y los objetos.

Por: Daniel Ojalvo Canedo

API 5: Autorización vulnerable a nivel de función

Las políticas complejas de control de acceso con diferentes jerarquías, grupos y roles, y una separación poco clara entre las funciones administrativas y regulares, tienden a conducir a fallas de autorización. Al explotar estos problemas, los atacantes obtienen acceso a los recursos y/o funciones administrativas de otros usuarios.

Por: Daniel Ojalvo Canedo

API 5: Autorización vulnerable a nivel de función

Ejemplo:

Legitimate – POST method is correctly requested	**Attack** – Request is modified to send a DELETE method
POST /example/api/v1/provision/user/search HTTP/1.1 User-Agent: AHC/1.0	**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": [
 	
]
}
	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": [
	
	]
	}

Por: Daniel Ojalvo Canedo

API 5: Autorización vulnerable a nivel de función

Prevención:

- ✓ Los mecanismos de aplicación deben denegar todos los accesos de forma predeterminada, lo que requiere concesiones explícitas a roles específicos para acceder a cada función.
- ✓ Revisar los puntos finales de la API contra fallas de autorización de nivel de función, teniendo en cuenta la lógica comercial de la jerarquía de aplicaciones y grupos.
- ✓ Asegurar de que las funciones administrativas dentro de un controlador regular implementen comprobaciones de autorización basadas en el grupo y la función del usuario.

Por: Daniel Ojalvo Canedo

API 6: Asignación masiva

La vinculación de datos proporcionados por el cliente (por ejemplo, JSON) a modelos de datos, sin un filtrado de propiedades adecuado basado en una lista blanca, generalmente conduce a una asignación masiva de datos. Adivinar las propiedades de los objetos, explorar otros end-point de la API, leer documentación o proporcionar propiedades de objetos adicionales en las cargas útiles de solicitud, permite a los atacantes modificar las propiedades de los objetos que no deberían hacerlo.

Por: Daniel Ojalvo Canedo

API 6: Asignación masiva

Ejemplo:

Legitimate - Client sends a legitimate request	**Attack** – Attackers sends the same request but adds the admin role in the request body

`PUT /api/v2/users/5deb9097 HTTP/1.1`
 	
`User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36`
`X-Forwarded-For: 19.42.129.253`
 	
`{`
`"_id": "5deb9097",`
`"address": "******, NY City, NY",`
`"company_role": "Investment Services",`
`"email": "******",`
`"first_name": "******",`
`"full_name": "******",`
`"job_title": "Broker",`
`"last_name": "******",`
`"phone_number": "******"`
`}`
	`PUT /api/v2/users/5deb9097 HTTP/1.1`
	
	`User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36`
	`X-Forwarded-For: 19.42.129.253`
	
	`{`
	`"_id": "5deb9097",`
	`"address": "******, NY City, NY",`
	`"company_role": "Investment Services",`
	`"email": "******",`
	`"first_name": "******",`
	`"full_name": "******",`
	`"is_admin": true,`
	`"is_sso": true,`
	`"job_title": "Broker",`
	`"last_name": "******",`
	`"permission_type": "admin",`
	`"phone_number": "******",`
	`"role": "admin",`
	`"sso_type": "admin",`
	`"system_user_type": "admin",`
	`"system_user_type_cd": 2,`
	`"user_type": "admin",`
	`"user_type_cd": 10`
	`}`

Por: Daniel Ojalvo Canedo

API 6: Asignación masiva

Prevención:

- ✓ Si es posible, evitar utilizar funciones que enlacen automáticamente la entrada de un cliente en variables u objetos internos.
- ✓ Incluir en una lista blanca solo las propiedades que el cliente pueda y deba actualizar.
- ✓ Utilizar las funciones integradas para incluir en la lista negra las propiedades a las que los clientes no deberían acceder.
- ✓ Si corresponde, definir explícitamente y forzar esquemas en datos de entrada.

Por: Daniel Ojalvo Canedo

API 7: Configuración incorrecta de seguridad

La configuración incorrecta de seguridad suele ser el resultado de configuraciones predeterminadas no seguras, configuraciones incompletas o ad-hoc, almacenamiento abierto en la nube, encabezados HTTP mal configurados, métodos HTTP innecesarios, uso compartido de recursos de origen cruzado permisivo (CORS) y mensajes de error detallados que contienen información confidencial.

Por: Daniel Ojalvo Canedo

API 7: Configuración incorrecta de seguridad

Ejemplo:

Legitimate – Client sends a legitimate request	**Attack** – Attackers modify the API request and specify an invalid identifier, resulting in a detailed exception error

GET /api/v2/network/connections/593065
HTTP/1.1
Accept: application/json, text/plain, */*
Accept-Encoding: gzip
 	
HTTP/1.1 200 OK
{
"status":"success",
}
	GET
	/api/v2/network/connections/5930aaaaa
	HTTP/1.1
	Accept: application/json, text/plain, */*
	Accept-Encoding: gzip
	
	HTTP/1.1 500 Server Error
	{
	"status":"failure",
	"statusMessage":"An error occurred while validating input: validation error: unexpected content \"593065d1\" ({com.tibco.xml.validation}COMPLEX_E_UNEXPECTED_CONTENT) at /{http://www.tibco.com/namespaces/tnt/plugins/json}ActivityOutputClass[1]/searchSvcReqsByRepReq[1]/search[1]/status[1]/aaaa[1]\ncom.tibco.xml.validation.exception.UnexpectedElementException: unexpected content \"aaaa\"

\n\tat com.tibco.xml.validation.state.a.a.a(CMElementValidationState.java:476)
\n\tat com.tibco.xml.validation.state.a.a.a(CMElementValidationState.java:270)
\n\tat com.tibco.xml.validation.state.driver.ValidationJazz.c(ValidationJazz.java:993)
\n\tat com.tibco.xml.validation.state.driver.ValidationJazz.b(ValidationJazz.java:898)
\n\tat …..

Por: Daniel Ojalvo Canedo

API 7: Configuración incorrecta de seguridad

Prevención:

- ✓ Un canal de comunicación seguro para todos los activos estáticos (por ejemplo, imágenes y CSS).
- ✓ Incluir un proceso automatizado para evaluar continuamente la efectividad de la configuración y las configuraciones en todos los entornos.
- ✓ Evitar que se envíen excepciones y otra información valiosa a los atacantes y, si corresponde, definir y aplicar esquemas de respuestas de error.
- ✓ Asegurar de que solo los verbos HTTP especificados puedan acceder a la API. Todos los demás verbos HTTP deben estar deshabilitados (por ejemplo, HEAD).
- ✓ Las API que esperan acceder desde clientes basados en el navegador (por ejemplo, front-end de WebApp) deben implementar una política adecuada de Cross-Origin Resource Sharing (CORS).

Por: Daniel Ojalvo Canedo

API 8: Inyección

Las inyecciones SQL, NoSQL, comandos, etc., ocurren cuando se envían datos no confiables a un intérprete como parte de un comando o consulta. Los datos maliciosos del atacante pueden engañar al intérprete para que ejecute comandos no deseados o acceda a los datos sin la autorización adecuada.

Por: Daniel Ojalvo Canedo

API 8: Inyección

Ejemplo:

Legitimate - Client sends a legitimate request	**Attack** – Attackers sends the same request but adds an injection attempt

Request:
GET /v1/customers HTTP/1.1
 	
Authorization: Bearer gwwh1Y4epjv9Y
 	
Cookie: _ga=GA1.3.630674023.1502871544; _gid=GA1.2.1579405782.1502871544;userId=207939055
Host: payments-api.dnssf.com
X-Forwarded-For: 54.183.50.90
 	
{
userId: “207939055”
}
	Request:
	POST/v1/customers HTTP/1.1
	
	Authorization: Bearer gwwh1Y4epjv9Y
	
	Cookie: _ga=GA1.3.630674023.1502871544; _gid=GA1.2.1579405782.1502871544;userId=207939055
	Host: payments-api.dnssf.com
	X-Forwarded-For: 54.183.50.90
	
	{
	userId: “207939055’ OR 1=1”
	}

Por: Daniel Ojalvo Canedo

API 8: Inyección

Prevención:

- ✓ La prevención de la inyección requiere mantener los datos separados de los comandos y consultas.
- ✓ Realizar la validación de datos utilizando una biblioteca única, confiable y mantenida activamente.
- ✓ Validar, filtrar y desinfectar todos los datos proporcionados por el cliente u otros datos provenientes de sistemas integrados.
- ✓ Los caracteres especiales se deben escapar utilizando la sintaxis específica para el intérprete de destino.
- ✓ Utilizar una API segura que proporcione una interfaz parametrizada.
- ✓ Siempre limitar el número de registros devueltos para evitar la divulgación masiva en caso de inyección.

Por: Daniel Ojalvo Canedo

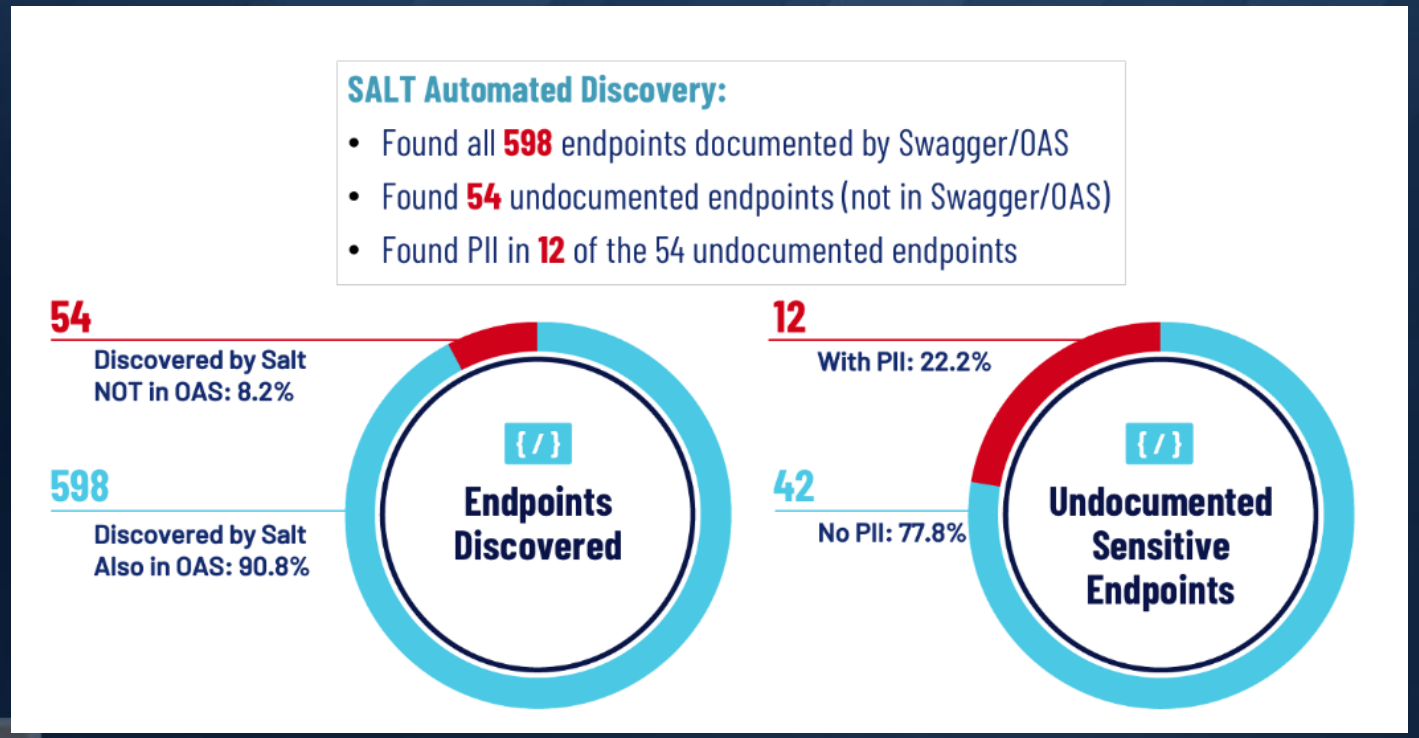
API 9: Gestión de activos inapropiada

Las API tienden a exponer más end-points que las aplicaciones web tradicionales, lo que hace que la documentación adecuada y actualizada sea muy importante. Los hosts adecuados y el inventario de versiones de API implementadas también juegan un papel importante para mitigar problemas como versiones de API obsoletas y puntos finales de depuración expuestos.

Por: Daniel Ojalvo Canedo

API 9: Gestión de activos inapropiada

Ejemplo:



Por: Daniel Ojalvo Canedo

API 9: Gestión de activos inapropiada

Prevención:

- ✓ Hacer un inventario de todos los hosts de la API y documentar los aspectos importantes de cada uno de ellos, centrándose en el entorno de la API (por ejemplo, producción, test, desarrollo), quién debe tener acceso de red al host (por ejemplo, público, interno, socios) y versiones de la API
- ✓ Hacer un inventario de los servicios integrados y documentar aspectos importantes como su papel en el sistema, qué datos se intercambian (flujo de datos) y su sensibilidad.
- ✓ Documentar todos los aspectos de la API, como autenticación, errores, redireccionamientos, limitación de velocidad, política de intercambio de recursos de origen cruzado (CORS) y puntos finales, incluidos sus parámetros, solicitudes y respuestas.

Por: Daniel Ojalvo Canedo

API 10: Registro y monitoreo insuficiente

El registro y el monitoreo insuficientes, junto con la integración faltante o ineficaz de la respuesta a incidentes, permite facilitar los ataques, mantener la persistencia, controlar y manipular más sistemas, extraer o destruir datos sin control, etc. La mayoría de los estudios de incumplimiento demuestran que el tiempo para detectar un incumplimiento es superior a 200 días, generalmente detectado por partes externas en lugar de procesos internos o monitoreo.

Por: Daniel Ojalvo Canedo

API 10: Registro y monitoreo insuficiente

Ejemplo:

Una plataforma para compartir videos fue atacada por un ataque de relleno de credenciales a "gran escala". A pesar de que se registraron los inicios de sesión fallidos, no se activaron alertas durante el tiempo del ataque. Como reacción a las quejas de los usuarios, se analizaron los registros de la API y se detectó el ataque. La empresa tuvo que hacer un anuncio público pidiendo a los usuarios que restablecieran sus contraseñas y reportar el incidente a las autoridades reguladoras.

Por: Daniel Ojalvo Canedo

API 10: Registro y monitoreo insuficiente

Prevención:

- ✓ Registre todos los intentos fallidos de autenticación, acceso denegado y errores de validación de entrada.
- ✓ Los registros deben escribirse en un formato adecuado para ser consumidos por una solución de administración de registros y deben incluir suficientes detalles para identificar al actor malicioso.
- ✓ Los registros deben manejarse como datos confidenciales y su integridad debe garantizarse en reposo y en tránsito.
- ✓ Configure un sistema de monitoreo para monitorear continuamente la infraestructura, la red y el funcionamiento de la API.

Por: Daniel Ojalvo Canedo

Por: Daniel Ojalvo Canedo