Hacking APIS

Introducción APIS - Teoria

1. REST (Representational State Transfer) es un estilo de arquitectura de software
2. REST es un conjunto de reglas o diseño de software que permite crear servicios web más simples.
3. Resumen
   - Resources and representations
   - Addressability
   - Uniform Interface:
     - **Resource identification in requests:** Resources are identified in requests and are separate from the representations returned to the client.
     - **Resource manipulation through representations:** Clients receive files that represent resources. These representations must have enough information to allow modification or deletion.

        - **Self-descriptive messages:** Each message returned to a client contains enough information to describe how the client should process the information.
        - **Hypermedia as the engine of application state:** After accessing a resource, the REST client should be able to discover through hyperlinks all other actions that are currently available.
    - Stateless
    - Hyperlink for navigation between resources
    - Client-Server
        - The client and server should be independent of each other. The changes you make on the server shouldn't affect the client and vice versa.

Conceptos

Recursos (Resources)

1. Es una abstracción, cualquier cosa puede ser un recurso. Un documento, una imagen, un servicio, etc, Un resource es una "entidad" o un "recurso que es información".
2. Cada recurso esta asociado a una URL
3. El cliente nunca ve al recurso directamente, sino una represtación del mismo. Un recurso puede tener N representaciones (json, html, una imagen...)

Verbos

Es la acción que se puede llevar a cabo en la entidad. En la petición se indica el verbo = acción que se quiere relizar.

- Get
- Post
- Put
- Patch
- Delete
- Connect
- Trace
- Options
    - Utilizado para identificar que metodos estan asociados al recurso

Media Types

Especifica al servidor y al cliente el "tipo" de dato que se solicita en el request o se envia en la respuesta. De estas manera se pueden aplicar las normas de parseado para procesar correctamente la petición.

- Application/json
- Aplication/xml

Status Code

Indica el "estado" del servidor respecto a la petición

- 200: Petición recibida y procesada - Exito
- 300: Redirección. La solicitud ha sido recibida por el servidor. Sin embargo, para asegurar un procesamiento exitoso es necesario que el *cliente* tome una acción adicional. Este tipo de

códigos aparecen principalmente cuando hay redirecciones.
- 400: Errores del cliente. El servidor ha recibido la petición pero no la puede llevar a cabo
- 500: Errores del servidor

Stateless

1. El servidor no mantiene ningú estado y no "recuerda" lo que el cliente ha hecho anteriormente, no sabe que información (como credenciales) se ha enviado antes
2. Cada petición desde un cliente a un servidor **ha de contener** toda la información necesaria para que el servidor procese la petición. Cada petición es unica y independiente del resto.

   Por ejemplo, cada petición ha de contener un "elemento" que permita autenticar al usuario que general la petición.

API endpoint

1. **Los endpoints** son las URLs de un API o un backend que responden a una petición. Los mismos entrypoints tienen que calzar con un endpoint para existir. Algo debe responder para que se renderice un sitio con sentido para el visitante. Por cada entrypoint esperando la visita de un usuario puede haber docenas de endpoints sirviendo los datos para llenar cada gráfico e infografía que se despliega en el entrypoint.
2. La diferencia entre entrypoint y endpoint es que los **endpoints no están pensados para interactuar con el usuario final**. Usualmente sólo devolverán json, o no devolverán nada. Y más que frecuentemente, un entrypoint hará varios llamados a distintos endpoints para mostrar estadísticas, galerías, últimos comentarios, etc.
3. Adicionalmente, se asume que cuando se habla de un endpoint estamos en un entorno **RESTful**, por lo cual (a diferencia del uso de un browser), el cliente *puede usar un mismo endpoint con distintos verbos*.

Ideas

1. Con una arquitectura API el servidor es usado más como un proxy para los datos
2. El procesamiento y la presentación de los datos se realiza en el cliente, no en el servidor
3. Los clientes consumen datos en raw

Pentest APIS

Primeros pasos - Approach

1. **Entender el flujo legitimo del flujo de trabajo (work-flow)**
2. Solicitar la información disponible
3. Disponer de ejemplos petición-respuesta
4. Conocer la especificicación necesaria de cada petición (que headers se han de utilizar, verbos etc)
5. Conocer alto nivel que se quiere hacer con cada api
6. Conocer el proceso de autenticación
7. Conocer la tipologia de usuarios y privilegios
8. Conocer el entorno donde se van a montar las apis (browser, app movil)
9. Conocer el objetivo principal de la API (transferencias bancarias por ejemplo)
10. Definir el scope de la revisión

Top Vulnerabilidades

Sql Injections

1. Detectar parametros controlados por el usuario
2. Comprobar si son vulnerables a SQLI
3. Usad tecnicas propias de SQLiç
4. Posible solucion: Prepared statement

Datos sensibles en GET

1. Los datos pueden registrarse en elementos intermedios de la arquitectura

Weak Auth token

1. Tokens criptograficamente inseguros
2. No son unicos
3. No caducan con el tiempo
4. Son reutilizables
5. Falta de controles de acceso

Insecure direct object references

1. Acceso a objetos de la aplicacion
2. Fallos de autorización

Leakage Data APi

1. Se comunica información al cliente que no es esencial para el correcto funcionamiento del cliente
2. Exposición API Key

Detección de endpoints no documentados

1. Revisar la documentación

Uso de versos PUT/DELETE

1. Uso de estos verbos sin la autoriazación asociada

Fallos de autorización

1. Debes evitar los recursos bajo un ID de usuario. Usa `/me/orders` en lugar de `/user/654321/orders`.
2. No uses IDs auto incrementales. Usa `UUID` en su lugar.

XXE

1. Utilizar XXE en json
2. Cambiar las cabeceras necesarias
3. External Entities

Cambiar las versiones de las API

1. Si hay un v2, probar con V1

Broken Authentication

1. Compromiso de la API
2. Compromiso de la identidad del usuario
3. Valida que todos los endpoints estén protegidos con autenticación para evitar romper el proceso de autenticación.

Limitación de tiempo a peticiones generadas

1. Existencia de ataques por fuerza bruta
2. Denegaciones de servicio

Gestión del logout

1. Vigencia de la sesion una vez el usuario se ha desautenticado

Intercambio de metodos

1. GET-> POST
2. application/son->application/xml

Fallos logica de negocio

Fallos de configuracion

1. Cors
2. Cabeceras de seguridad

Debilidades criptograficas

Herramientas

1. Postman + Burp
2. FuzzAPI (https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=94798094)
3. Astra
4. Zappa

Recursos

1. https://gitlab.com/pentest-tools/API-Security-Checklist/-/blob/master/README-es.md
2. https://itnext.io/how-to-build-a-restful-api-a-deep-dive-into-rest-apis-215188f80854
3. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
4. https://owasp.org/www-project-api-security/
5. https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces
6. https://www.udemy.com/course/hacking-rest-apis/