

Suplantaciones de peticiones de scripting multisitio

ATAQUE CSRF

A veces, incluso ING, YouTube, el New York Times y Google se equivocan. **POR KURT SEIFRIED**

Cross-Site Request Forgery (también conocido como Cross-Site Reference Forgery, CSRF y XSRF) está convirtiéndose en un serio problema de seguridad del que la mayoría de los programadores y usuarios son unos benditos ignorantes. CSRF es un ataque basado en web que ha evolucionado, aunque sigue siendo primo cercano a los más tradicionales ataques Cross-Site Scripting (XSS). En un exploit XSS, el atacante introduce contenido malicioso en una aplicación web (por ejemplo, creando una URL malformada o embebiendo código hostil en un cuadro de respuestas de un formulario) que resulta en contenido hostil, como puede ser JavaScript malicioso, que se inserta en otro contenido seguro que se sirve después a la víctima. Los ataques CSRF van más allá insertando contenido hostil, que tiene como consecuencia una acción a través del navegador web del usuario, como el cambio de un filtro configurado dentro del correo electrónico basado en web o iniciar una transferencia económica desde una cuenta bancaria online.

Ejemplo de Ataque CSRF

Imaginemos que vamos a nuestra red social favorita para chatear con nuestros amigos. Desafortunadamente, el sitio en cuestión permite a los usuarios insertar

imágenes en conversaciones basadas en web (por ejemplo, avatares para un foro). En lugar de usar una URL como

```

```

el atacante usa una URL como

```

```

Por tanto, cuando un navegador web del usuario intenta cargar la imagen, en lugar de conectarse al sitio de la red social, ejecuta un comando para cambiar la contraseña.

Este ataque también puede desarrollarse desde otros sitios. Por ejemplo, si un usuario permanece registrado en el sitio de la red social mientras está navegando la web en otra pestaña y una imagen u otro sitio apunta hacia la URL de modificación de la contraseña, esa pestaña podría ejecutar el comando, y a menos que el sitio tuviera implementado protecciones contra CSRF, la contraseña del usuario acabaría siendo cambiada.

Los ataques CSRF han terminado convirtiéndose en populares debido principalmente a tres simples razones. La primera

es la emergencia de los servicios basados en web tales como correo electrónico, comercio online, banca, etc. Estos ataques CSRF pueden acabar en el envío de dinero a un atacante a través de banca basada en web o lugares de venta de acciones. El correo basado en web permite a un atacante resetear o solicitar copias de nuestras contraseñas para distintos servicios, como registradores de DNS y sitios de comercio online. Los atacantes pueden convertir en dinero esos ataques mediante el acceso directo a las cuentas bancarias, reseteando una contraseña de usuario, etc. Algo tan simple como resetear una cuenta de usuario puede traer como consecuencia un control del agresor de la cuenta del usuario o mantendrá el servicio secuestrado. Por una módica suma, el agresor reseteará la contraseña y la devolverá al usuario (véase la Figura 1).

La segunda razón es la de la navegación con pestañas. Cuando aparecieron los primeros navegadores web, navegar la web era en gran parte una experiencia realizada en serie. Nunca se nos ocurrió que necesitaríamos más de una sesión abierta simultáneamente, porque el contenido no era tal como para mantenerlo abierto mucho tiempo (navegar la web era bastante literalmente navegar la web de un lugar a otro). Sin embargo, con la llegada del email basado en web, es normal tener al menos tres sesiones abiertas para poder enviar correos rápidamente y que nos sea notificada la entrada de un mensaje nuevo. Esto significa que un ataque CSRF es mucho más probable que ocurra porque siempre estamos registrados en nuestra cuenta de correo basado en web (personalmente utilizo un navegador separado para mi correo para evitar exactamente esto).

La tercera razón es que la mayoría de las aplicaciones web no disponen de seguridad. Son absolutamente terribles filtrando la entrada del usuario adecuadamente, permitiendo a los atacantes la inyección de contenido malicioso (como JavaScript) mediante cualquier número de vulnerabilidades de scripting multisitio. A pesar de que rara vez suelo visitar sitios web hostiles, sí que visito muchos "de

confianza” que sé de hecho que tienen una mala filtración que puede llevar a ataques XSS, permitiendo en última instancia ataques CSRF.

Además, pocas aplicaciones web implementan protecciones CSRF que evitarían estos ataques.

Defensa para Programadores

La manera de combatir a los CSRF es simple en teoría, aunque, dependiendo de la aplicación web de que se trate, el grado de dificultad de la solución práctica va desde fácil a

casi imposible de implementar adecuadamente. Para superar los ataques CSRF, una aplicación ha de verificar que cada petición se hace de forma apropiada; en otras palabras, nuestra aplicación necesita mantener el estado de manera que el contenido en la pestaña “A” del navegador que está registrado en nuestro email basado en web es el único permitido para enviar correo, y una petición hecha mediante el contenido en la pestaña “B” del navegador que tiene acceso a un sitio web hostil no tiene como consecuencia el envío o la lectura de un correo. Para ello nuestra aplicación web tiene que mantener el estado de la información. A pesar de que la web se diseñó en un principio como un sistema sin estado, cualquier adición de estado requiere algunas artimañas técnicas, ya que el navegador web no puede ayudarnos directamente.

Para enlazar el contenido en una página web a una petición realizada desde esa página web adecuadamente (por ejemplo, el usuario cumplimenta un formulario y pulsa *Submit*), necesitamos pasar un token único en el contenido que el navegador web pasa entonces hacia atrás con la petición, permitiéndonos confirmar que ésta ha venido desde el sitio adecuado.

Enviar y Recibir Tokens

Ahora este token único puede ser enviado y recibido de distintas maneras.

- Campos de Formularios Ocultos

```
<input type="hidden"
name="token"
value="randomstring" />
```

La ventaja aquí es que muchas aplicaciones soportan la adición de campos de for-

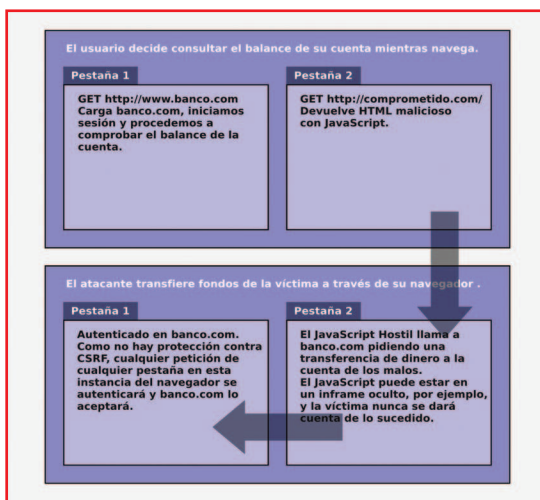


Figura 1: Ejemplo de cómo funciona un ataque CSRF.

mulario y disponen de la lógica para procesarlos. La desventaja, que las páginas web que no usan formularios aunque permitan interacción no pueden ser direccionadas tan fácilmente mediante esta técnica.

- Componentes URL (Dentro de la URL o de los Parámetros)

```
http://example.org/newpassword?
new=password&token=randomstring
```

Ésta tiene la ventaja de hacer que los datos se encuentren disponibles en el servidor, de tal forma que, en teoría, tiene un módulo Apache que valida todas las peticiones y bloquea simplemente las inválidas, evitando a la aplicación recibirlas. La desventaja (o potencialmente una ventaja, dependiendo de nuestro punto de vista) es que los usuarios ya no pueden marcar una página porque el token único ya no será válido.

- Cookies

```
PHP: setcookie("TokenCookie",
$randomstring);
```

Los cookies deben habilitarse para que esto funcione; y potencialmente pueden ser robados por un agresor inteligente (varios fallos en varios navegadores permitían el robo de cookies durante años). La ventaja de esta técnica, sin embargo, es que es bastante invisible al usuario y no requiere que la HTML sea presentada al mismo o que la URL a usar se modifique de ninguna manera.

- Requerimientos del Backend

Todos estos ejemplos necesitan algún formulario en el fondo para almacenar los datos de la sesión y crear sesiones tokens,

compararlos, y permitir o no las peticiones basadas en ellos. Además, las aplicaciones basadas en web necesitarían ser modificadas (por ejemplo, si los campos del formulario oculto se usan para pasar los datos). Lo bueno es que cada vez más las aplicaciones web implementan esta protección por defecto. Por ejemplo, el popular Framework Joomla! tiene disponible ahora la función `JRequest::checkToken()`.

Defensas para Usuarios Web

Las buenas noticias son que se encuentran disponibles defensas para ataques CSRF para navegadores web. Una común es el plugin NoScript para Firefox. Desafortunadamente, para que NoScript sea efectivo necesitamos deshabilitar JavaScript y habilitar entonces selectivamente JavaScript para sitios de confianza. Esto nos conduce a obvios problemas de usabilidad, porque muchos sitios no funcionan nada o muy poco si JavaScript no se encuentra habilitado. Además, no evitará que un agresor aproveche un error de scripting multisitio en un sitio en el que confiamos.

Sin embargo, no todos los navegadores soportan un control selectivo sobre los sitios que ejecutan JavaScript. Otra opción consiste simplemente en instalar un navegador web separado o ejecutar una instancia separada de un navegador web y usarla para actividades online de confianza tales como banca o correo basado en web.

Un navegador que posee incorporada esta estrategia es Google Chrome. Cada pestaña del navegador en Chrome es de hecho un proceso separado y no un hilo ejecutándose dentro del mismo contexto como otros hilos (pestañas). Por tanto, las pestañas no pueden interferir entre ellas, lo que inhabilita la mayoría de los ataques CSRF (para ser atacado con éxito tendríamos que registrarnos en un servicio basado en web, y luego usando esa misma pestaña, ir a un sitio hostil).

RECURSOS

- [1] Cross-Site Request Forgery (CSRF): http://www.owasp.org/index.php/Cross-Site_Request_Forgery
- [2] Zeller, W., y Felten, E.W. "Cross-Site Request Forgeries: Exploitation and Prevention," 2008, <http://www.freedom-to-tinker.com/sites/default/files/csrf.pdf>