

INSEGURIDADES

XSS y `ap_escape_html()`

El asunto de Cross Site Scripting (XSS) ha sido comentado recientemente en las noticias, así que pensé que debía comenzar esta remozada sección echando un vistazo a este clásico problema de seguridad. Como el usuario final realmente no puede hacer nada para parar el tema, XSS es una forma de ataque que los chicos malos adoran. Un ataque XSS permite a los atacantes insertar contenido hostil (como JavaScript) en las páginas web de un sitio en el que confía el usuario. Lo que hace que este problema sea particularmente grave es que ocurre en el software mismo del servidor web (servidor HTTPD de Apache). Incluso estando seguros de que nuestras aplicaciones basadas en web están seguras contra XSS, un atacante puede ejecutar el ataque a través del software del servidor de Apache.

El software HTTPD de Apache implementa la función `ap_escape_html()` para desinfectar la entrada y evadir los símbolos `<` y `>` y los caracteres de control `&`, usados a menudo en ataques XSS. Si un atacante puede colar estos caracteres en, por ejemplo, un mensaje de error enviado a la víctima, el atacante puede conseguir entrar.

En la mayoría de los casos donde la salida se pasa a una víctima potencial, `ap_escape_html()` se usa apropiadamente para desinfectar la salida y asegurarse de que nada hostil pasará. Desafortunadamente, el equipo de Apache olvidó usar esta función de desinfección en unos cuantos sitios, en bastantes sitios, de hecho.

Listado 1: `httpd-2.2.8/modules/http/http_protocol.c`

```
01 case HTTP_METHOD_NOT_ALLOWED:
02     return(apr_pstrcat(p,
03 -         "<p> The requested
method ", r - >method,
04 +         "<p> The requested
method ",
05 +         ap_escape_html(r -
>pool , r - >method) ,
06         " is not allowed for
the URL " ,
```

La reciente actualización del servidor HTTPD de Apache incluye no una corrección para una vulnerabilidad XSS, sino que también implementa `ap_escape_html()` en una docena de otros lugares donde los datos se pasan a una víctima potencial.

El código en el Listado 1 se encuentra en el formato diff universal (`-` significa que una línea fue eliminada, y `+` indica la que la sustituyó). El uso de `ap_escape_html()` es bastante sencillo y realmente no afecta a la legítima funcionalidad del software.

Es decir, el cambio mostrado en el Listado 1 refleja una atractiva situación clásica: Un problema está escondido en el código durante un tiempo, incluso aunque se encuentre disponible una relativamente simple solución.

La solución simplemente no está implementada, lo que conduce a una vulnerabilidad de seguridad.

Fail Safe

Uno de los principios fundamentales del diseño de sistemas es que éste debería controlar fallos y condiciones de error inesperados con soltura (este principio es conocido como *fail safe design* o *diseño a prueba de fallos*). El hecho es que tristemente la mayoría del software, incluyendo la mayor parte de software de código abierto, no está diseñado a prueba de fallos. De hecho, los programas fallan bastante a menudo de forma espectacular, como en los casos de los desbordamientos de búfer causantes de la ejecución de código arbitrario, corrupción de datos, etc.

¿Si no puedes confiar en la seguridad del sistema, qué puedes hacer?

La respuesta es que simplemente no confíes. Para administradores de sistemas y redes esto significa incrementar múltiples capas de seguridad; si una falla, entonces afortunadamente la capa siguiente se hará cargo del problema. En este caso, la solución más obvia es implementar algún tipo de proxy web (como el servidor web Squid) que entienda los datos que se envían y pueda filtrarlos; por ejemplo, bloquear el paso de caracteres `<` o `>` dentro de una petición URL.

Adicionalmente, un administrador podría también verificar los datos de salida y com-

probar su contenido hostil potencial; sin embargo, el hecho de que el sitio también incluya probablemente JavaScript legítimo hace que la tarea de identificación de scripts hostiles sea mucho más difícil.

Un usuario no puede esperar a que cada sitio tome medidas para prevenir ataques XSS (nótese que muchos sitios no pueden ni siquiera molestarse en hacer que las páginas se rendericen correctamente en Firefox).

Desafortunadamente, los usuarios tienen unas cuantas opciones para protegerse contra los ataques XSS, y el consejo estándar de no pulsar en enlaces de origen desconocido está lo suficientemente expandido hoy en día.

NoScript

Para usuarios Firefox, la mejor opción es usar el add-on NoScript [1]. Aunque el bloque JavaScript suministrado por NoScript hace muy poco (en algunos casos los usuarios deberían tener ya situado el sitio en la categoría *allow JavaScript*, el add-on proporciona bloqueo rudimentario de ataques de scripting XSS pasados a través de peticiones URL, avisando a los usuarios y ofreciéndoles la opción de recargar la página.

Conclusión

En este artículo muestro el porqué no soy programador. No soy lo suficientemente listo ni meticuloso para escribir código seguro. Un paso de desinfección perdido, un error lógico menor, o incluso una errata pueden conducir potencialmente a resultados desastrosos. Además, muestro por qué la seguridad es tan problemática. Incluso un administrador que hace todo debidamente, puede acabar con problemas de seguridad, y debido a la naturaleza interconectada de las redes de ordenadores moderna (o mejor, la única red a la que estamos adjuntando continuamente todo, incluyendo nuestra tostadora diaria), un problema de seguridad de alguien más puede convertirse fácilmente en nuestro propio problema de seguridad también. ■

RECURSOS

- [1] Add-on de NoScript: <http://noscript.net/>
- [2] Metasploit: <http://metasploit.com/>