

Ataque de archivos y protocolos

FUZZING

Explicamos cómo el *fuzzing* de archivos y protocolos puede mejorar directamente la calidad del código.

Aprenderemos también acerca de las herramientas de código abierto disponibles para hacer fuzzing.

POR KURT SEIFRIED

Cuando comencé a manejar ordenadores hace mucho tiempo, recuerdo que se podían abrir archivos (imágenes, documentos, o lo que fuera) sin tener que preocuparse de los virus. Todos sabían en el mundo de Windows que sólo se podía pillar un virus con archivos ejecutables como *.exe*, *.bat* y *.com*. En el mundo de Linux, aunque en realidad se podía hacer que el cliente de correo ejecutase automáticamente los archivos (ejecutables) adjuntos en los emails, el sentido común impedía hacerlo. Recuerdo cómo nos reímos cuando alguien preguntó si se podía infectar al visualizar las imágenes que recibía por email. Los tiempos han cambiado.

Los ataques basados en archivos están a la orden del día; en el momento de escribir estas líneas, se acaba de hacer público un exploit basado en PDF, y Adobe ha anunciado que lo arreglarían en un par de semanas. Mientras tanto ... puede que no sea demasiado conveniente abrir archivos PDF, al menos si el visor de PDFs instalado es Adobe Reader. Por suerte, la mayoría de los usuarios de Linux no usan

Adobe Reader para visualizar los archivos PDF, aunque muchos de ellos tendrán instalado Adobe Flash (explotable a través de archivos SWF en las versiones de Flash Player para Linux anteriores a la 10.0.12.36), o bien OpenOffice (mediante archivos EMF y WMF con las versiones de OpenOffice anteriores a la 2.4.2). La lista sigue y sigue: archivos de imagen, archivos de fuentes, páginas web, lo que sea. Muchas aplicaciones importantes fallan en algún punto al parsear los archivos para cuyo manejo están diseñadas, permitiendo a los atacantes la creación de archivos capaces de hacer que se ejecute código arbitrario al abrirse.

Cómo Funciona el Fuzzing

La premisa básica del fuzzing de archivos y protocolos es la sutil (o no tan sutil) creación de una entrada de datos malformada. Se puede hacer mediante métodos tan simples como voltear bits aleatorios de un archivo o del flujo de un protocolo para crear un programa completo o suite de pruebas que sabe cómo generar archivos válidos (o no válidos) o datos de entrada para varios protocolos. Un ejemplo exce-

lente de fuzzer de archivos es el programa CGI *mangleme*, de Michael Zalewski. Si instalamos este programa CGI en un servidor web y conectamos a él un navegador web cliente, el script *mangle.cgi* crea un archivo HTML generado aleatoriamente que incluye un elemento *META REFRESH*, provocando que el navegador web recargue el script indefinidamente y que obtenga una página HTML generada, también aleatoriamente, cada vez. De este modo, podemos apuntar el navegador web a la URL que contiene el script *mangle.cgi* y esperar a que falle (casi siempre lo hace). El uso de *META REFRESH* pone de relieve una de las claves en el fuzzing de archivos: hacer que un programa cargue una serie de archivos uno tras otro automáticamente sin que tenga que haber un humano detrás abriendo archivos.

El fuzzing provoca ciertos problemas y oportunidades interesantes. Por un lado, se puede ejecutar un programa contra una batería de pruebas de forma rápida y barata: dejando el navegador conectado a *mangleme* durante algunos días, lo sometemos a varias decenas de miles de casos de prueba. Cada petición genera una entrada de registro con el número identificador del caso de prueba, con la que podremos reproducir el problema. Aún así, puede que en algunos casos no sea tan fácil reproducir o verificar los resultados. Por ejemplo, con la versión 7.0.5730.13 de Internet Explorer, el caso de prueba 0x6dc61276 de *mangleme* no parece tener ningún efecto, excepto con una máquina en la que está instalado PowerDVD 6.0 en la que Internet Explorer falla creando en el escritorio un archivo vacío llamado *su0.mpg*, sin pedir ningún tipo de confirmación. Este método "de escopeta" para la realización de pruebas de seguridad, no es que sea demasiado preciso, pero igual que sucede cuando se dispara una escopeta en un bosque, a veces se cruza algo (que

podría o no ser lo que andamos buscando).

Fuzzing Inteligente

El fuzzing aleatorio presenta varios problemas. Por ejemplo, es poco probable que hallemos ciertas condiciones límite con el simple uso de datos sin sentido; los desbordamientos (por arriba y por debajo) de enteros son un problema frecuente, pero para provocarlos debemos manipular un archivo o el tráfico del protocolo de un modo especial. Muchos programas usan el entero de C largo con signo (que soporta valores que van desde -2147483647 hasta 2147483647) para especificar por ejemplo la longitud de los campos de datos. De este modo se consiguen valores interesantes: 0x7fffffff, que es 2147483647 (el máximo) y 0x80000000, que es -2147483648 (el valor mínimo que se puede almacenar). Si el programa no calcula adecuadamente estos valores, puede llevarnos a una situación en la que al sumar dos números positivos obtengamos un número negativo, o que al restarle un número a otro obtengamos un número aún mayor. Al usar dichos valores para ubicar la memoria en la que se guardarán los datos proporcionados por el usuario, tendremos como resultado el típico desbordamiento de búfer o de pila, normalmente explotables y que permiten la ejecución de código arbitrario.

Escribe una Vez, Ejecuta un Millón de Veces

Uno de los aspectos más fascinantes del software es que casi todo el gasto se produce en la creación de la primera copia. Una vez implementada ésta, es posible generar un millón de copias a un coste casi nulo. La implementación de una herramienta para fuzzing de archivos o protocolos no tendría por qué ser diferente. Pero comprender un formato de archivo no es barato. Los fundamentos de PDF, por ejemplo, están definidos en un documento de 756 páginas [1]. Y aún así no están cubiertos todos los contenidos que se pueden encontrar en un PDF, ya que actualmente pueden integrarse varios formatos de imagen e incluso código JavaScript en el interior del archivo PDF.

Podríamos decir que la documentación completa para PDF, con todos los archivos de imagen soportados, JavaScript, etc. constaría de varios miles de páginas (motivo por el cual probablemente sea buena idea el uso de un visor de archivos PDF con menos funcionalidades cuando se está preocupado por la seguridad). De todos modos, una vez llegados a esta documentación e implementado un fuzzer funcional, hay muchas probabilidades de encontrar bugs explotables rápidamente. Un ejemplo es la suite de pruebas PROTOS [2], de la Universidad de Oulo, Finlandia.

Con un framework de Codenomicon Ltd., un grupo relativamente pequeño y de presupuesto limitado, pudimos escribir suites para el testado de protocolos como WAP, http, LDAP, SNMP, SIP, H.323, ISAKMP y DNS. La suite de pruebas tuvo tanto éxito hallando bugs, que el proyecto CVE no pudo asignar el número de CVE por bug habitual y tuvo que agruparlos en unos pocos de números debido a la cantidad de fallos encontrados.

Herramientas de Código Abierto para Fuzzing

Si se quiere jugar con herramientas de fuzzing, o simplemente se planea poner a prueba algún sistema o software, las opciones disponibles son varias (Tabla 1).

Algunas de las herramientas, como *mangleme* o *QueFuzz*, se pueden tener operativas en unos pocos minutos. Otras, como *SPIKE*, presentan una curva de aprendizaje pronunciada y están orientadas más bien a personas con intención de implementar sus propias herramientas de fuzzing con fines prospectivos (tienen una curva de aprendizaje muy parecida a la de *Matterhorn*).

Conclusiones

La buena noticia es que las herramientas de fuzzing repercuten directamente en la calidad del código. Un desarrollador tiene pocas excusas cuando en un caso de prueba (en forma de archivos o flujo de datos de red) se produce un fallo en su aplicación. En el mejor de los casos, el desarrollador creará un código más robusto y tendrá más cuidado con las entradas de datos inválidos o malformados, aunque la historia demuestra que no es probable que ocurra a corto plazo. La mala noticia es que los chicos malos, que cada vez son más listos, comenzarán a usar herramientas de fuzzing para encontrar fallos que explotar (como prueban los actuales ataques 0-day contra Adobe Acrobat y Microsoft Excel perpetrados por el gusano *conficker*, que ya había infectado al menos a 15 millones de sistemas Windows a fecha de 26 de Enero de 2009). ■

Tabla 1: Herramientas de Fuzzing

| Herramienta | Descripción | URL |
|------------------------|--|---|
| mangleme | Generador de HTML aleatorio | http://lcamtuf.coredump.cx/soft/mangleme.tgz |
| Browser Fuzzer 2 | Fuzzer para navegadores que genera CSS, COM, HTML y JavaScript aleatorios | http://www.krakowlabs.com/dev/fuz/bf2/bf2.tar.gz |
| fzem | Fuzzer para clientes de email (genera respuestas de servidor inválidas, etc) | http://www.krakowlabs.com/dev/fuz/fzem/fzem.tar.gz |
| fsfuzzer | Fuzzer para sistemas de archivos (ntfs, ext3, ext2, vfat, iso9660, etc) | http://projects.info-pull.com/mokb/fsfuzzer-0.6.tgz |
| FileP y FileH | Fuzzers basados en Python y Haskell que mutan una lista de archivos y se la pasan a las aplicaciones | http://www.isecpartners.com/file_fuzzers.html |
| ProxyFuzz | Un Fuzzer MITM capaz de mutar aleatoriamente el tráfico de red | http://theartoffuzzing.com/downloads/proxyfuzz/proxyfuzz.py |
| Peach Fuzzing Platform | Una plataforma de fuzzing completa con modelado de datos y de estados | http://peachfuzzer.com/ |
| GPF | Toma capturas de la red, modifica los datos y los envía a un servidor para ver qué pasa. | http://www.vdalabs.com/tools/efs_gpf.html |
| SPIKE | Juego de herramientas escrito en C para la creación de fuzzers | http://www.immunitysec.com/resources-freesoftware.shtml |
| QueFuzz | Usa iptables para interceptar y mutar los paquetes de red | http://code.google.com/p/quefuzz/ |

RECURSOS

[1] Especificación de PDF: http://www.adobe.com/devnet/acrobat/pdfs/PDF32000_2008.pdf

[2] PROTOS: <http://www.ee.oulu.fi/research/ouspg/>