Fuzzing y seguridad

José Miguel Esparza Muñoz
Security Researcher S21sec labs

10 de agosto de 2007

Resumen

Con este artículo se pretende dar las nociones básicas para el acercamiento del lector a una de las técnicas más utilizadas actualmente a la hora de descubrir fallos en aplicaciones, mostrando como ejemplo práctico una nueva vulnerabilidad ya publicada.

1. Introducción

Vivimos en plena era tecnológica, donde cada vez más información y datos de carácter personal son informatizados. Con este escenario en mente, la seguridad de los sistemas toma progresivamente mayor importancia, sobretodo ahora que la mayoría de los atacantes ya no se contentan con hacer acto de presencia, sino que son contratados por mafias y empresas de la competencia para saltarse las medidas de protección y robar o destruir datos importantes.

Existen diversas formas de intrusión en un sistema, desde el acceso físico al remoto, aprovechándose este último de los servicios ofrecidos por las máquinas. Normalmente se piensa que con un buen antivirus, un cortafuegos con buenas reglas de filtrado y una buena política de parcheo es suficiente. Pero nadie se para a pensar en las llamadas vulnerabilidades 0-day, de las que nadie tiene conocimiento y que se descubren a diario en cantidad de aplicaciones ya sean servidores web, ftp, smtp, pop3, y un largo etcétera. En concreto, se les llama así a los bugs en aplicaciones de los que el fabricante no tiene conocimiento, o si lo tiene, no hay solución publicada en ese momento. Si estos descubrimientos los realiza un investigador de seguridad

normalmente no habrá problemas, pero si es un criminal, según el tipo de aplicación afectada, las consecuencias pueden ser devastadoras.

Para lograr el descubrimiento de estos fallos se suelen emplear, entre otras, unas herramientas que se dedican a lanzar peticiones mal formadas de forma automática. A éstas se les llama fuzzers, y pueden estar enfocadas tanto a protocolos de red, como a formatos de archivos, sistemas de ficheros, etc. Son de gran utilidad para auditores informáticos y desarrolladores de software, entre otros, para evitar intrusiones indeseadas unos y para obtener un producto seguro los otros.

2. ¿Qué es el Fuzzing?

Se llama fuzzing a las diferentes técnicas de testeo de software capaces de generar y enviar datos secuenciales o aleatorios a una o varias áreas o puntos de una aplicación, con el objeto de detectar defectos o vulnerabilidades existentes en el software auditado. Es utilizado como complemento a las prácticas habituales de chequeo de software, ya que proporcionan cobertura a fallos de datos y regiones de código no testados, gracias a la combinación del poder de la aleatoriedad y ataques heurísticos entre otros.

El fuzzing es usado por compañías de software y proyectos *open source* para mejorar la calidad del software, por investigadores de seguridad para descubrir y publicar vulnerabilidades, por auditores informáticos para analizar sistemas, y, en última instancia, por delincuentes para encontrar agujeros en sistemas y explotarlos de forma secreta.

Ésta técnica no es nueva, ya que fue desarrollada en la Universidad de Wisconsin Madison por el catedrático Barton Miller y sus estudiantes en el año 1989[2]. En la actualidad sigue investigando sobre el tema, demostrando que hasta los sistemas operativos más modernos pueden volverse inestables por la acción de un sencillo fuzzing[4].

Como se ha comentado, las herramientas semiautomáticas que utilizan esta técnica se llaman fuzzers[3]. Son semiautomáticas porque pese a ser un proceso automatizado de envío de datos, se necesita de una persona que analice los resultados y verifique las posibles vulnerabilidades encontradas.

En general, la mayoría de los fuzzers intentan encontrar vulnerabilidades del tipo *buffer overflow*, *integer overflow*, *format string* o *condiciones de carrera*, aunque también pueden abarcar otros tipos de errores, como *inyecciones sql*, por ejemplo.

El funcionamiento de los fuzzers suele componerse de las siguientes etapas:

- **Obtención de datos:** dependiendo del tipo de fuzzing deseado y según la implementación de la herramienta, se obtendrán los datos a enviar de

una lista estática almacenada en archivos o en el propio código fuente, o se generará en el mismo momento según las configuraciones efectuadas. Este proceso se puede realizar únicamente al inicio de la sesión o justo antes de cada envío.

- **Envío de datos:** una vez se dispone de la información que se desea enviar a la aplicación objetivo, se realizará el proceso, dependiendo, por ejemplo, de si se hace a través de una red informática o de si se quiere realizar únicamente un chequeo local.

- **Análisis:** después de realizado el envío, sólo quedará esperar los resultados del fuzzing. Si no se espera ninguna respuesta por parte del objetivo, se deberá estar alerta por si se produce un comportamiento inesperado. Si, en cambio, se recibe una respuesta, entonces en este momento se comprobará si ésta indica un comportamiento normal o si, por el contrario, el ataque ha tenido éxito y la aplicación ha quedado inestable.

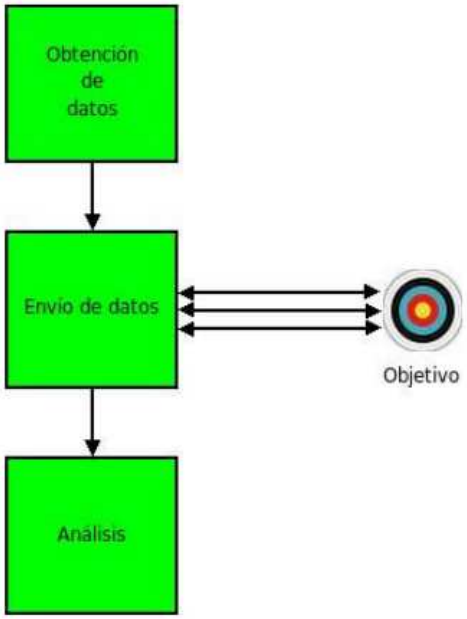

Figura 1: Arquitectura genérica de un fuzzer

En este proceso los puntos importantes son la obtención de los datos y el análisis posterior al envío. El primero es fundamental porque los datos deben ser útiles a la hora de crear un punto de inestabilidad en el objetivo, pero

el segundo también lo es, ya que sin esa fase nunca se sabría si el ataque ha sido efectivo o no.

Gracias a esta técnica, y a los diversos fuzzers existentes, se han descubiertos miles de bugs, dejando claro que su efectividad está más que demostrada.

3. Técnicas de Fuzzing

Dependiendo de los datos que se envíen al objetivo se puede hablar de diferentes técnicas de fuzzing. Se suelen englobar en dos grupos, la mutación y la generación de datos. Dentro de este último, y dependiendo de la vulnerabilidad que se busque, habrá otras tantas formas de realizarlo.

3.1. Mutación

Es un método bastante rápido y efectivo, y consiste en partir de una entrada válida y realizar ciertas mutaciones de esos datos, con la intención de que sigan siendo válidas para la aplicación pero lo suficientemente inesperadas para lograr el objetivo.

Figura 2: Mutación de datos

3.2. Generación

Se trata de un proceso más lento que el anterior, porque se deben crear los datos antes de enviarlos, pero puede descubrir fallos que el otro método obviaba.

Normalmente, los datos obtenidos también se sustituirán en una entrada válida del objetivo, ya que si se envía simplemente un dato aleatorio, se corre el riesgo de que el programa lo deseche sin ni si quiera procesarlo.

Dependiendo de la forma de generar los datos se encuentran otros dos subgrupos. Por una parte, los recursivos, que se obtienen de la iteración sobre un alfabeto dado o de la repetición de un mismo carácter. Algunos ejemplos de este tipo son los siguientes:

- **Permutación:** teniendo en cuenta, por ejemplo, el alfabeto hexadecimal "0,1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F" y buscando datos de seis caracteres se podrían obtener 16^6 combinaciones posibles para sustituir de la forma siguiente:

Figura 3: Permutación de datos

- **Repetición:** se usa generalmente para la búsqueda de fallos del tipo *buffer overflow*. Se suele usar un incremento en cada repetición para no alargar el proceso inútilmente, ya que los puntos críticos se encuentran cerca de los límites de potencias de 2, como 256, 512, 1024, etc. La Figura 4 muestra un ejemplo de este tipo.

El otro subgrupo es el que hace uso de la sustitución, que se centrará en reemplazar cierta parte de una entrada válida por los elementos de un *vector de fuzzing*, que es una lista de posibles test a realizar, dependientes de la vulnerabilidad a chequear. Así pues, dos de los casos más importantes son los siguientes:

Figura 4: Incremento del número de caracteres

- ***Integer overflow*:** ocurre cuando un programa recibe en una entrada un número que está fuera del rango que es capaz de manejar: un cero, un número negativo, un número mayor o menor que el rango actual, o uno con diferente tipo numérico (decimal en vez de entero). Este agujero puede derivar en un ataque de *buffer overflow*. Un posible ejemplo de esto puede ser la modificación del campo *Content-Length* en una petición HTTP.
- **Errores de cadenas de formato (*format string*):** este tipo de ataques se producen cuando se introducen ciertos elementos para proporcionar formato a las cadenas de texto y no existe un mínimo control de seguridad. De esta forma, se puede conseguir una denegación de servicio o incluso la ejecución de código arbitrario en el sistema.

4. Fuzzing práctico

A modo de ejemplo se va a detallar a continuación una vulnerabilidad descubierta gracias al uso de un fuzzer de reciente creación llamado *Malybuzz*[5]. En este caso, la aplicación objetivo era el servidor FTP *Wzdftpd* y concretamente se utilizó la técnica de generación por repetición con la intención de encontrar un fallo de *buffer overflow*.

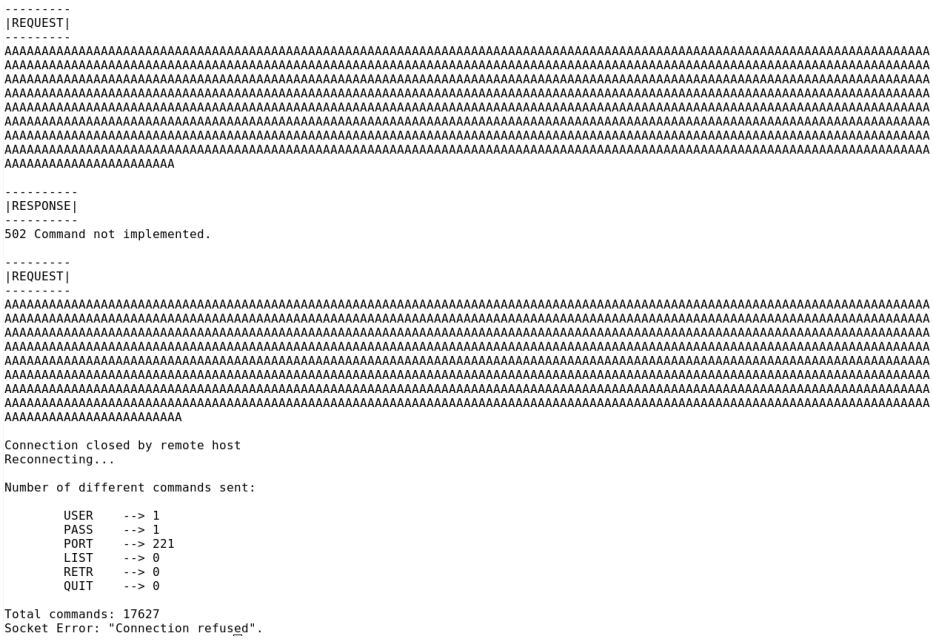

Figura 5: Fuzzing del servidor FTP con Malybuzz

Como se puede ver en la figura anterior, después del envío de una serie de caracteres repetidos, el servidor se cierra inesperadamente, provocando un fallo de conexión al intentar enviar otro comando. La imagen siguiente muestra el fallo provocado por la acción del fuzzer:

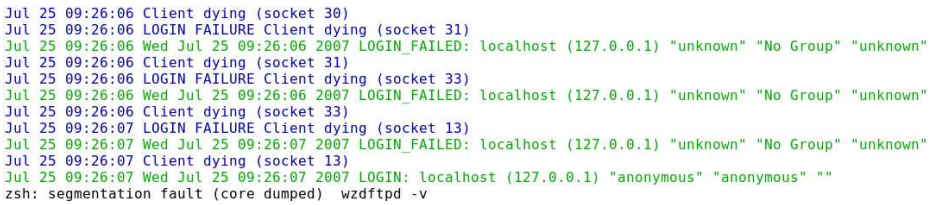

Figura 6: Instante del cierre brusco del servidor FTP

Después de la correspondiente investigación se determinó que la vulnerabilidad no desembocaba en la corrupción de la pila del sistema, como podía esperarse, sino que se trataba de un mal manejo de los datos de entrada, obteniéndose de ello un puntero nulo que creaba el error, dando lugar a una denegación de servicio. Más concretamente, el fallo se debía a que, una vez hecha la autenticación, el servidor truncaba los datos de entrada a bloques de 1024 caracteres, para más tarde eliminar los caracteres de control `\t, \r y \n`.

De esta forma, al enviar 1024 caracteres más los dos de retorno de carro y salto de línea, y después del proceso ya comentado, se obtenía un caracter nulo que se iba pasando de función en función hasta dar con una que no manejaba correctamente los punteros nulos, provocando así el error fatal.

```
int chtbl_lookup(const CHTBL *htab, const void *key, void **data)
{
  ListElmt *element;
  CHTBL_Elmnt *entry;
  unsigned int index;

  index = htab->h(key) % htab->containers;

  for (element=list_head(&htab->table[index]); element != NULL; element =
list_next(element))
  {
    entry = list_data(element);
    if (!entry) return -1;
    if (htab->match(key, entry->key)==0) {
      if (data) *data = entry->data;
      return 0;
    }
  }

  return 1;
}
```

Figura 7: Función vulnerable

Esta vulnerabilidad se puede reproducir también de otras formas, como por ejemplo, enviándo simplemente una tabulación al servidor. Este fallo resultó en un 0-day, informando al autor en cuestión, para posteriormente publicarse el correspondiente *advisory* en las listas habituales[1].

Este ejemplo viene a corroborar lo ya dicho de la efectividad del uso del fuzzing en el descubrimiento de nuevas vulenrabilidades en las aplicaciones, en este caso, de red.

5. Conclusiones

El fuzzing, como se ha comentado, es una alternativa muy eficaz a la hora de encontrar vulnerabilidades en aplicaciones, sobre todo nuevas vulnerabilidades, pero es preciso recordar que no existe ninguna herramienta que pueda garantizar la seguridad de un software al 100 %, por lo que el uso de este tipo de soluciones debe ser una parte más de la auditoría de un sistema, sin olvidar los clásicos scanners de vulnerabilidades o scanners de código fuente, por ejemplo.

El éxito de los fuzzers, y la gran cantidad de vulnerabilidades descubiertas gracias a su ayuda, dejan en tela de juicio a las compañías de software, más preocupadas por lanzar un nuevo producto o una nueva versión cuanto antes que de la seguridad de su programación, dejando totalmente indefensos a sus clientes. Por ello, es necesaria una inmediata concienciación sobre este tema, con el objetivo de que el software sea cada vez más seguro y, por ende, los sistemas sean más robustos frente a la gran cantidad de ataques actuales.

Referencias

[1] *Denegación de servicio en wzdftpd*, `http://www.s21sec.com/avisos/s21sec-033-en.txt`.

[2] *Fuzz testing*, `http://en.wikipedia.org/wiki/Fuzz_testing`.

[3] Matthew Franz, *Fuzzing tools*, 2007, `http://www.threatmind.net/secwiki/FuzzingTools`.

[4] Barton P. Miller, *Fuzz testing of application reliability*, 2006, `http://pages.cs.wisc.edu/~bart/fuzz/`.

[5] José Miguel Esparza Muñoz, *Malybuzz*, `http://malybuzz.sourceforge.net`.