Exploits remotos y locales

José María Alonso Cebrián
Jordi Gay Sensat
Antonio Guzmán Sacristán
Pedro Laguna Durán
Alejandro Martín Bailón
Jordi Serra Ruiz

PID_00208395

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Hemos podido ver cómo un programa funcionalmente correcto podría ser inseguro debido a un *bug*. Estos *bugs* pueden ser aprovechados por programas para sacarles algún partido que incida en la seguridad del sistema. Estos programas que sacan partido de los *bugs* se conocen como *exploits*. En el presente módulo vamos a familiarizarnos con los *exploits* antes de empezar a ver cómo pueden ser creados a partir del módulo siguiente.

Clasificar todos los tipos de *exploits* que existen es una ardua tarea, sobre todo teniendo en cuenta la cantidad de características que se pueden utilizar para establecer las clases. Se podría escoger la plataforma hardware a la que afectan o el sistema operativo para el que están diseñados. Es innegable que cuando se analiza la seguridad de un sistema, estos dos órdenes de clasificación son muy importantes. Desde el punto de vista de la gestión de la seguridad quizá el impacto y la peligrosidad son clasificaciones a tener muy en cuenta, ya que los planes de remediación deben estar guiados por un orden claro de criticidad.

Existe una página web, http://www.milw0rm.com/, donde la categorización se realiza dependiendo de si el *exploit* afecta a una vulnerabilidad remota o local, aunque además se hacen dos distinciones más: *exploits* destacados y *exploits* para aplicaciones web.

Captura del sitio milw0rm.com

En esta página se publican *exploits* de todo tipo para casi cualquier *bug* que se encuentre. La publicación de estos *exploits* es un tanto caótica, dado que los administradores publican el texto del *exploit* tal como le es reportado a la web. Por consiguiente, aunque todos los *exploits* suelen conservar una estructura similar, no todos siguen un mismo patrón o estructura como se hace en otras páginas como Secunia o Security Focus.

Queremos recomendar milworm como fuente de conocimiento constructivo y nunca destructivo para comprender mejor el funcionamiento de los *exploits* y trucos comúnmente usados en la explotación de los *bugs*.

Desde el punto de vista académico, lo más importante para nosotros sería establecer un orden basado en la arquitectura del *exploit*, es decir, en entender cómo está creado, cuál es el patrón de creación del *exploit* y cuál su arquitectura interna.

En los próximos apartados se van a estudiar las características de algunos *exploits* locales y remotos y un *framework* creado para la creación y uso de *exploits*, *Metasploit*. Las arquitecturas internas de los *exploits* se irán viendo en los próximos módulos.

1. *Exploits* Remotos

Estos *exploits*, capaces de sacar partido de un *bug*, son sin duda los más populares, peligrosos, efectivos y deseados por los creadores de *exploits*. Además, son los más cotizados.

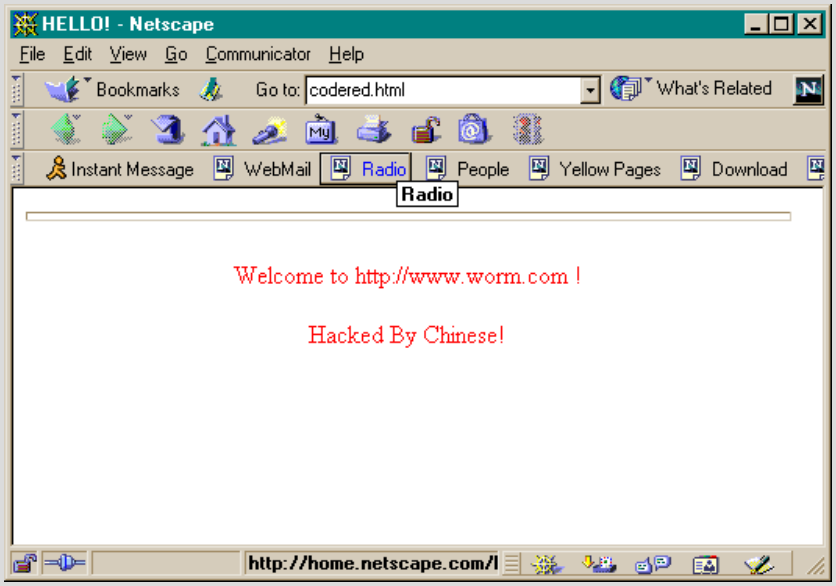
Un *exploit* que permita lograr privilegios de administrador en una máquina remota de la que sólo conocemos nombre o su IP y un puerto con un servicio vulnerable a la escucha es lo mismo que la posibilidad de utilizar una o mil máquinas como si fueran en propiedad.

Esto hace que las compañías de desarrollo de software deban estar especialmente preocupadas en las medidas de protección de los servicios que se exponen a la red en un servidor.

Un claro ejemplo de estos exploits remotos son los gusanos que se comentaron en el primer módulo. Un gusano que se difunde por una red como *Sasser*, *Blaster* o *Code Red* tiene que tener una forma de copiarse de una máquina a otra. Estos gusanos, por ejemplo, tenían en común que aprovechaban servicios de red vulnerables explotables en remoto.

Code Red

Code Red se puso en Internet el día 13 de julio del año 2001. Utilizaba como patrón de replicación servidores *Microsoft Windows NT 4.0 o 2000* que ejecutaban el software de servidor de *Internet Information Server 4.0* con el servicio vulnerable *Indexing Service 2.0*. La vulnerabilidad era un *Buffer Overflow* que permitía ejecutar código inyectado en memoria para hacer un *defacement*, es decir, una suplantación de la página principal del servidor web, por una que decía *hacked by Chinese*.



Servidor infectado por *Code Red*

Para infectar una máquina desde otra, *Code Red* lanzaba un *exploit* contra el programa del servicio de indexación que corría en el servidor web. El servicio utilizaba el programa `default.ida` que tenía un desbordamiento de buffer en el parámetro de entrada y con una petición como la siguiente conseguía infectarse de un servidor a otro:

```
GET
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%
u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u909
0%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0
```

Como se puede observar, la cabecera contenía gran número de caracteres `N`, usados para desbordar el *buffer* y colocar el código deseado en memoria. Sólo se necesitaba localizar un servidor vulnerable y lanzar una petición correctamente malformada para lograr infectar al sistema.

En este caso, el *exploit* hacía uso del protocolo http, pero se podría aplicar a cualquier servicio vulnerable que reciba datos desde la red. Un servidor FTP que no controle el tamaño del campo del nombre de usuario o un servidor DNS que acepte IPs de un rango no válido podrían ser ejemplos de servicios vulnerables utilizables para explotar un servidor en remoto.

MSN Messenger 7.X (MS07-054

Baste como otro ejemplo de *exploits* remotos el que se publicó el 11 de septiembre de 2007 para el cliente de mensajería instantánea *MSN Messenger 7.X (MS07-054*) que permitía la ejecución remota de código a través del protocolo de video de la *webcam*.

El fallo se producía al enviar un paquete de *webcam* malformado. Se descubrió que en el protocolo de comunicación existía una falta de validación en lo que al tamaño asignable a las variables correspondía, dando lugar a un fallo de *heap overflow*. Los fallos de *heap overflow* serán vistos durante este curso, pero la idea consiste en desbordar las variables situadas en la parte del *heap*, destinada a valores dinámicos. Este *bug* permitió crear un *exploit* que generaba paquetes de video malformados y que lograba, en los entornos vulnerables, una *shell* remota en el equipo atacado.

Algunos *exploits* remotos requieren la interacción con la víctima o un entorno autenticado para que puedan ser explotados. El gran riesgo es la aparición de *exploits* remotos que permitan ejecutar código arbitrario en la máquina atacada en servicios ampliamente usados, como ya pasó en el pasado con los gusanos mencionados.

1.1. *Frameworks* para creación de *exploits*

A la hora de crear un *exploit* para un sistema remoto es importante conocer todos los detalles posibles sobre el mismo. La arquitectura y las medidas de protección de esos sistemas no son iguales y por supuesto es posible que el código que se ejecute tampoco lo sea. No va a ser lo mismo crear un *exploit* para un sistema operativo Windows XP con el *Service Pack 2* en idioma inglés que para un sistema operativo Windows XP con el *Service Pack 3* en alemán. Por ejemplo, el fallo anterior relativo a la *webcam* en el programa *Microsoft MSN Messenger* sólo devolvía una *shell* remota si se trataba de un sistema operativo Microsoft Windows 2000 con el *Service Pack 4*.

Para poder crear de una forma automatizada *exploits* que funcionen en plataformas similares, se ha trabajado en la creación de un *framework* que permita al desarrollador de exploits, con pequeños cambios, crear un *exploit* que funcione en todas las plataformas vulnerables.

1.2. *Metasploit*

Metasploit es un proyecto *Open Source* basado en este concepto. Esta herramienta no está pensada únicamente para la creación de *exploits*, sino también para la ejecución y prueba de los mismos en entornos de tests de penetración.

Página Web

Puede ser descargado desde la URL del proyecto y tienen versiones para sistemas Microsoft Windows y sistemas UNIX: http://www.metasploit.org/

Metasploit Framework 3.2

Uno de los conceptos que hay que tener en cuenta cuando se utiliza *Metasploit* es la diferenciación entre un *exploit* y un *payload*.

El *exploit* es el la parte necesaria para el aprovechamiento de un *bug*. Es decir, si el *bug* fuera del tipo en el que se puede producir un desbordamiento de buffer en un parámetro, entonces en *Metasploit* se llamaría *exploit* al programa que consigue desbordar ese parámetro y obtener el control de programa para ejecutar cualquier código.

Por el contrario, el *payload* será la parte independiente del tipo de *bug* que queremos que el ataque ejecute. Es decir, supongamos que deseamos que tras lanzar el *exploit* se obtenga una *shell* en la máquina por un determinado puerto. En este entorno el *payload* será el código necesario para crear la *shell* corriendo en ese puerto.

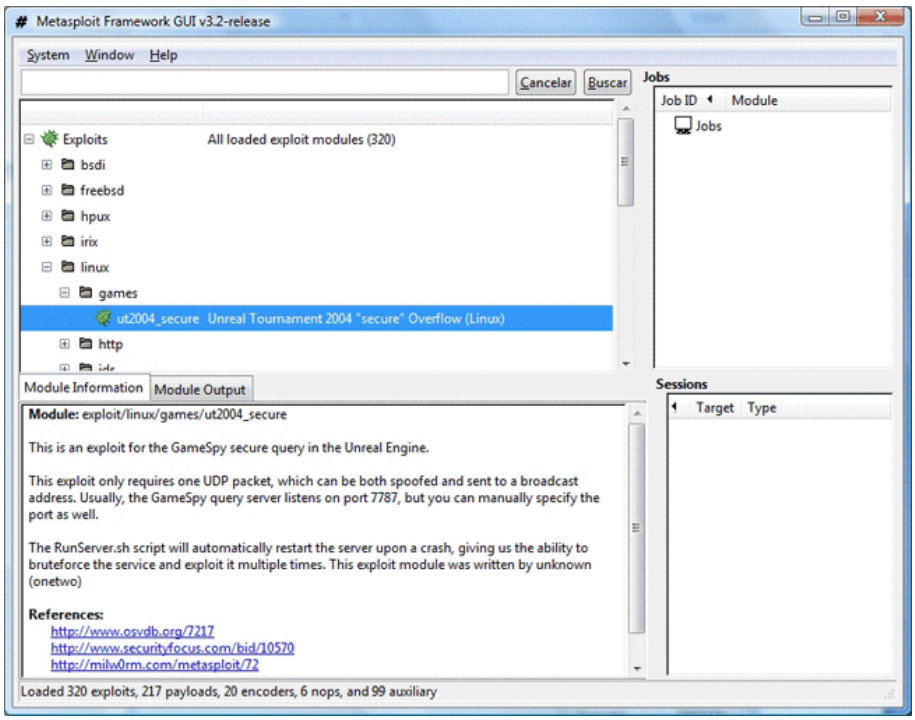
Metasploit proporciona una lista de *payloads* programados que pueden ir desde ejecutar un comando ADD USER en un sistema para lograr que se cree una cuenta de usuario hasta lograr una conexión de control remoto por VNC en el sistema vulnerado.

1.3. Un ejemplo con *Metasploit*

Metasploit, una vez unido el *exploit* en concreto y el *payload* adecuado, generará un *exploit* completo que permitirá explotar el *bug* en un servidor vulnerable con la ejecución del *payload* seleccionado.

Para ver el funcionamiento de este entorno de escritura de *exploits* vamos a realizar una simulación de aprovechamiento de una vulnerabilidad del *Unreal Tournament* 2004 en una plataforma Linux. Este software es un servicio de red que permite jugar a múltiples usuarios al juego *Unreal* vulnerable a un *bug* de buffer *overflow*.

Para crear este *exploit* basta con desplegar el nodo de *exploits*, seleccionar la plataforma vulnerable y elegir el tipo de software vulnerable. Es decir, Linux, *Games*, *Unreal Tournament* 2004.



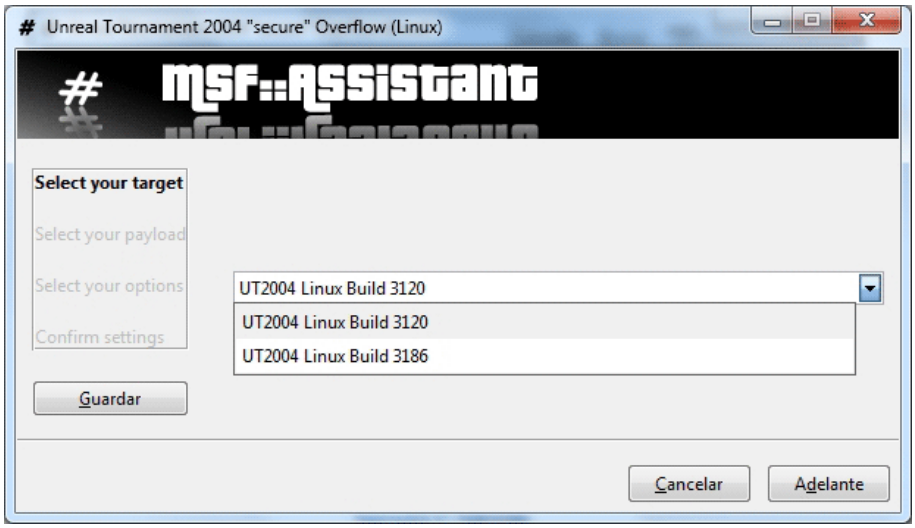
Creación de *Exploit* para *Unreal Tournament* 2004

Este *bug* es explotable mediante el envío de un paquete UDP malformado y permite la ejecución de código arbitrario en la máquina vulnerable, con lo que sería posible desde ejecutar cualquier comando hasta crear una *shell* utilizable en remoto por el atacante.

Como se puede ver en la parte de información del módulo, no sólo tenemos la descripción del fallo y el funcionamiento del mismo, sino que además viene acompañado con los links de consulta a las bases de datos de expedientes de seguridad e incluso, en este caso, a la publicación de un *exploit* completo en *Milw0rm.com*

Para la creación de este *exploit* de ejemplo bastaría con hacer doble clic sobre el elemento en cuestión, es decir, sobre el *exploit* según la terminología utilizada por *Metasploit*.

Automáticamente aparecerá un asistente que nos guiará por la creación del *exploit*. En este caso, en primera instancia se nos permite seleccionar la versión vulnerable del software para el que se quiere construir un *exploit*. Esto puede ser debido a que, en ocasiones, el parámetro vulnerable cambia de tamaño, ubicación, nombre, forma de ser llamado o forma de ser explotado en diferentes versiones vulnerables. En este ejemplo, como se puede ver en la siguiente figura, se hace distinción entre la versión 3120 y la 3186.



Selección de la versión

Una vez seleccionada la versión del software objetivo, deberemos pasar a configurar el *payload*, es decir, el código que se debe ejecutar después de haber explotado correctamente la vulnerabilidad. Cada *payload* tendrá una serie de opciones distintas a configurar que serán seleccionadas en el tercer paso del asistente. Una vez terminado de configurar el *payload*, el *exploit* ya estará creado.

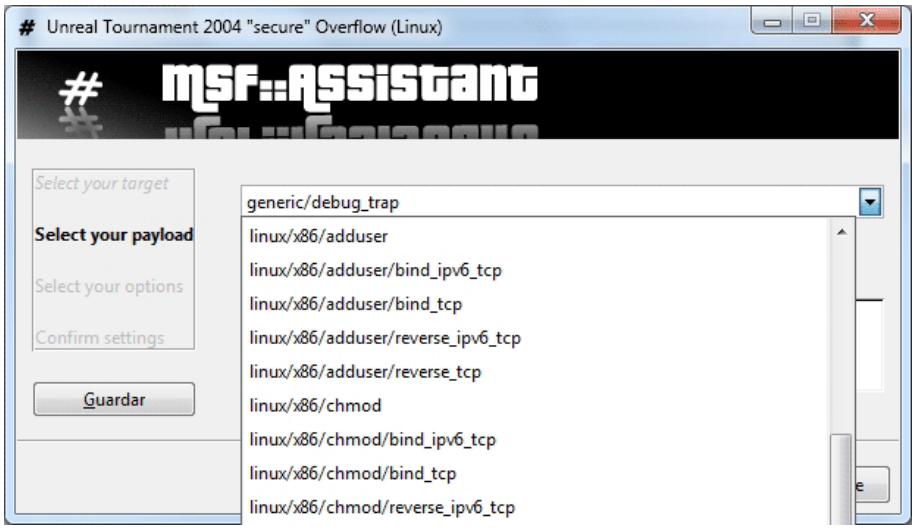
La lista de *payloads* que aparecerá es propia de cada sistema operativo y será automáticamente proporcionada por *Metasploit*. Sobre las técnicas de post explotación, es decir, de cuáles son las acciones a seguir tras la explotación exitosa de un *bug*, existen muchas corrientes y técnicas distintas.

Para sistemas Windows existen opciones como la consecución de una conexión remota por medio de VNC o incluso la instalación de un módulo completo de *scripts* automatizados para la ejecución de acciones en la máquina vulnerada.

Meterpreter es una librería de vínculos dinámicas (*.dll*) especialmente creada para la automatización de *scripts* que se inyecta en la memoria del sistema vulnerado. Permite realizar mediante *scripts* una multitud de acciones como crear una consola, subir o bajar ficheros, listar procesos, matarlos o crear otros nuevos utilizando simplemente llamadas a *scripts*.

En sistemas Linux las opciones son menos espectaculares, pero no menos efectivas. Existe la opción de ejecutar el comando `chmod`, que permite cambiar los permisos de un fichero concreto, listar directorios, crear usuarios o ejecutar cualquier otro comando.

Métodos comunes, con implementaciones distintas en cada sistema operativo, son los de añadir usuario al sistema o devolver una *shell* remota del sistema vulnerado con los privilegios de la cuenta que utiliza el servicio vulnerado.



Selección del *payload*

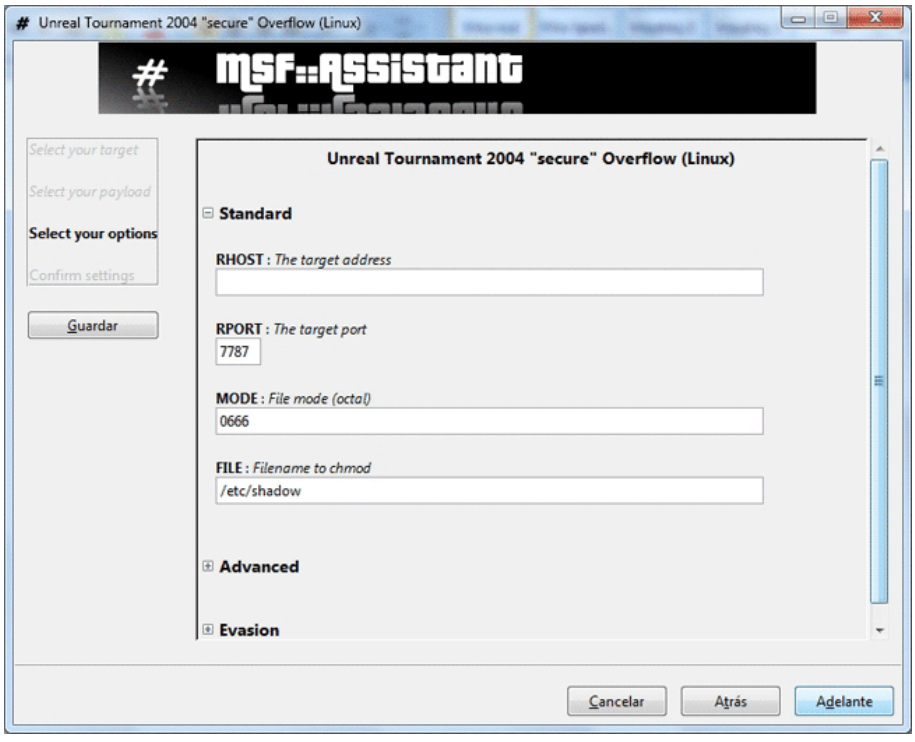
Como se puede apreciar en la figura anterior, la lista de *payloads* disponibles es larga. Por lo tanto, es una opción perfecta para los creadores de *exploits*, ya que con construir la cabecera del *exploit* para un nuevo *bug* descubierto tienen a su disposición un volumen alto de comportamientos ya construidos para automatizarlo.

Para este ejemplo se ha seleccionado el *payload* `linux/x86/chmod`. Con ese *payload* se va a poder cambiar, mediante el uso del comando chmod, la lista de permisos asociados a un fichero.

Para terminar la construcción del *exploit* sería necesario configurar la información necesaria final del *exploit*. En este ejemplo habría que configurar los parámetros relativos al software vulnerable, es decir la dirección IP del *host* remoto que se configuraría en el campo RHOST y el puerto por el que está escuchando el servicio vulnerable del *Unreal Tournament 2004* que se configuraría en el campo `RPORT`.

Además, por supuesto, es necesaria la información referente al *payload*. En este caso, el fichero al que se le quiere cambiar los permisos, que se configuraría en el campo `FILE` y la nueva lista de permisos que se quiere establecer, que se configuraría en el campo `MODE`.

Además, como se puede apreciar en la siguiente figura, existen características avanzadas que pueden ser configuradas para ajustar correctamente la construcción del *exploit*. Si el servidor tuviera alguna configuración especial cambiada con el tipo de codificación o con el envío de parámetros, podría ser ajustado en esta parte.



Configuración del *payload*

Además de las opciones avanzadas, algunos *exploits* vienen acompañados de opciones de evasión. Estas opciones de evasión están pensadas para cambiar las firmas de los ataques y conseguir que no se pueda detectar el envío de uno de estos *exploits* por Sistemas de Detección de Intrusiones o *Firewalls* que puedan estar defendiendo la red.

Como se puede ver en la imagen, este *exploit*, a falta de ser configurada la dirección IP del servidor a atacar, cambia los permisos del archivo `/etc/shadow` para que pueda ser leído por cualquier usuario. Este archivo almacena los *hashes* de las contraseñas de los usuarios y por defecto sólo puede ser leído por el root.

Una vez terminado el asistente, el *exploit* ya está listo para ser lanzado y esperar los resultados. Como se puede ver, es una herramienta muy cómoda para el ajuste y creación de *exploits* a la carta, pero que requiere la creación de las cabeceras de los *exploits* para estar actualizada y al día con los nuevos fallos descubiertos.

1.4. Definición del *exploit*

Sin embargo, nuestro interés es conocer cómo se ha escrito y cómo se ha generado el *exploit* con esta herramienta. Si echamos un vistazo al código fuente del módulo del *exploit* podremos acceder a cómo está generado cada uno de ellos. Basta con hacer clic con el botón derecho y seleccionar la opción contextual de mostrar el código fuente del módulo.

Si miramos el código del *exploit*, podremos ver que utiliza un lenguaje propio para trabajar con *Metasploit*. No es un lenguaje complejo y su alta estructuración permite que sea fácilmente leído por cualquier usuario con ciertos conocimientos técnicos.

A continuación se muestra un extracto del *exploit* lanzado anteriormente:

```
def exploit

connect_udp

buf = make_nops(1024)
buf[24, 4] = [target['Rets'][1]].pack('V')
buf[44, 4] = [target['Rets'][0]].pack('V')
buf[56, 4] = [target['Rets'][1]].pack('V')
buf[48, 6] = "\x8d\x64\x24\x0c\xff\xe4" #LEA/JMP

buf[0, 8] = "\\secure\\"
buf[buf.length - payload.encoded.length,
payload.encoded.length] = payload.encoded

udp_sock.put(buf)
handler

disconnect_udp

end
```

Extracto de *exploit* para *Unreal Tournament 2004*

Como se puede observar, existe una función *exploit* que genera un paquete UDP para enviar al servidor. Leyendo el código se puede ver cómo se genera un *array* de *NOPs* de 1024 elementos. Estos *NOPs* saturarán el buffer del servidor con instrucciones que no hacen nada, permitiendo al atacante colocar el *payload* en la zona de memoria que desee.

Cómo se puede ver, el *payload*, independientemente del *payload* elegido por el usuario, se trata como una secuencia de texto que se introduce en un *array* que el programador del *exploit* coloca donde corresponde. En este caso, lo coloca al final del paquete UDP que va a ser lanzado al servidor web.

En este ejemplo, al inicio del *array buf* se coloca el *exploit* que hará que el programa busque las instrucciones del *payload* en la zona de memoria que será cargada con el final de la variable *buf*, donde se ha colocado el *payload*.

El uso de las instrucciones *NOP* que se introducen entre la cabecera del *exploit* y el *payload* evita tener que acertar con la dirección exacta de memoria donde comienza el *payload*. Será suficiente con tener una idea aproximada de la dirección, con un margen de error de 1024 instrucciones, ya que si el contador de programa encuentra una instrucción *NOP*, ejecutará la instrucción siguiente. Este proceso irá pasando de instrucción *NOP* a instrucción *NOP* hasta que llegue a la dirección de memoria donde se encuentra el *payload*. Este es un buen truco cuando no se sabe exactamente la dirección de memoria.

Una buena metáfora para explicar esto sería imaginar que tenemos una pistola de agua y queremos, desde una distancia de unos cinco metros, llenar una botella. Si apuntamos a la boca de la botella, nos resultará muy complicado llenarla, y más aún si sólo disponemos de un disparo. En cambio, si colocamos un embudo en la boca de la botella, podremos acertar con mucha más facilidad. Cuanto mayor sea el embudo que coloquemos, más tardará el agua en llegar a la botella, pero más fácil será acertar y al final el agua llegará, que es nuestro objetivo. Exactamente igual pasa con los *NOPs* y el *payload* en este ejemplo.

2. *Exploits* Locales

No todos los *exploits* son remotos y el impacto de los *exploits* locales puede ser de alta importancia. Vimos cómo un fallo explotable en local permitía saltarse las protecciones de la consola WII para acceder a todo el sistema y poder ejecutar todo el código que se desee.

Además, el número de programas a los que se puede acceder en local es mucho mayor que los ofrecidos por red, y con la distribución adecuada del *exploit* pueden convertirse en remotos igualmente.

> Ejemplos actuales son los *exploits* que se distribuyen localmente a través de los dispositivos de almacenamiento USB. Al viejo estilo de los virus distribuidos por disquetes, muchos gusanos aprovechan *exploits* locales y la transmisión humana para distribuirse masivamente. Un ejemplo de esto ha sido el gusano *Conficker* para los sistemas operativos Windows Vista.

Otros *exploits* se generan a raíz de *bugs* que se encuentran en el manejo de los ficheros de entrada que maneja un programa.

> Ejemplos típicos de esto son los mp3 maliciosos que explotan una vulnerabilidad en un reproductor de audio y una vez que toman el control se conectan a un servidor y se descargan un *malware*.

Esto se ha repetido hasta la saciedad con ficheros *zip* o *rar*, con ficheros de video o audio *quicktime* o *mpeg*. Usualmente los programas almacenan los ficheros en un formato propietario más o menos conocido que puede analizarse para descubrir la manera en la que se almacenan los datos.

Estos *exploits* se pueden explotar de manera local enviando y engañando a un usuario un fichero malformado para que lo abra, esperando que use una versión de software vulnerable al *exploit* desarrollado. También se puede llegar a explotar estos fallos creando una página web que cargue automáticamente el *plugin* de un software vulnerable y un fichero malformado o utilizando la publicación de estos *exploits* con nombres sugerentes en las redes de intercambios de archivos *P2P*.

A continuación se muestra un extracto del código que genera un *exploit* para las versiones inferiores a la 0.9.6 del reproductor *VLC Media Player*. El código genera un fichero con extensión *.rt* que provoca, en el caso del *exploit* aquí expuesto que se ejecute la calculadora de Windows. Ejecutar la calculadora de Windows es un ejemplo recurrente en las pruebas de concepto de los sistemas Microsoft Windows, pues además de ser inocuo, demuestra que es posible ejecutar cualquier programa.

```
[...]
open(my $rt, "> s.rt");
print $rt "\x3C\x77\x69\x6E\x64\x6F\x77\x20\x68\x65".
          "\x69\x67\x68\x74\x3D\x22\x32\x35\x30\x22".
          "\x20\x77\x69\x64\x74\x68\x3D\x22\x33\x30".
          "\x30\x22\x20\x64\x75\x72\x61\x74\x69\x6F".
          "\x6E\x3D\x22\x31\x35\x22\x20\x62\x67\x63".
          "\x6F\x6C\x6F\x72\x3D\x22\x79\x65\x6C\x6C".
          "\x6F\x77\x22\x3E\x0D\x0A\x4D\x61\x72\x79".
          "\x20\x68\x61\x64\x20\x61\x20\x6C\x69\x74".
          "\x74\x6C\x65\x20\x6C\x61\x6D\x62\x2C\x0D".
          "\x0A\x3C\x62\x72\x2F\x3E\x3C\x74\x69\x6D".
          "\x65\x20\x62\x65\x67\x69\x6E\x3D\x22".
          $char x 72 . $eip . $jmp . $addr . $nop x 12 .
          $shellcode . $char x 1024 .
          "\x22\x2F\x3E\x0D\x0A\x3C\x62\x72\x2F\x3E".
          "\x3C\x74\x69\x6D\x65\x20\x62\x65\x67\x69".
 [...]
```

Extracto de *exploit VLC Media Player*

En este caso podemos observar cómo se escribe un fichero al que se le añaden algunas variables como son `$eip`, `$addr`, `$shellcode` o `$nop` con valores configurables que permiten que este *exploit* sea adaptado cambiando el *shellcode*, las direcciones de memoria o el colchón de *NOPs*.

2.1. *Fuzzing*

Descubrir este tipo de *bugs* en software local suele ser una tarea fácilmente automatizable. Lejos han quedado las técnicas de *testing* de software basadas en casos, pues el número de posibilidades de entradas que se puede dar hoy en día a un software utilizado en millones de equipos es tan grande que el *testing* basado en casos se queda muy corto.

Para ello se utilizan las técnicas de *fuzzing*. Consisten en enviar datos aleatorios a las entradas de los programas y lanzar tantos datos aleatorios con la mayor dispersión y factor de entropía posibles buscando detectar errores no controlados en el software.

Así, la forma más habitual de localizar y probar *exploits* locales como el que se ha mostrado para *VLC Media Player* suele ser utilizar un código en Perl, Python o incluso C que genere un fichero con la extensión adecuada para que se abra por defecto con la aplicación deseada.

Estos ficheros contendrán cabeceras válidas e inválidas, datos parcialmente correctos o totalmente incorrectos que irán variando aleatoriamente hasta que se produzca un fallo en la aplicación. Es en ese momento en el que se comprueba que datos de entrada han generado la caída o el comportamiento anómalo del programa para descubrir un *bug* explotable.

Aunque a priori el proceso para explotar una vulnerabilidad de tipo local pueda parecer más complicado, puede llegar a tener un gran impacto si se descubre en un software que se ejecute en la amplia mayoría de los ordenadores,

como puede ser *Adobe Flash Player* o *Acrobat Reader*. De hecho, el periodo de vida de un *exploit* para este tipo de utilería suele ser mucho más grande ya que la mayoría de este tipo de software normalmente se encuentra sin parchear en los ordenadores.

Al contrario de lo que ocurre con Metasploi,t no existen *frameworks* de generación de *exploits* multiplataforma para este tipo de fallos, por lo que cada uno se mostrará de la manera que el descubridor considere apropiada.