Explotación de Overflow en memoria heap en Windows

Explotación de memoria heap personalizada en Windows 7

Autor: Souhail Hammou
Traducción al Español (Latinoamericano,México): Christian Guzmán
Blog: http://rce4fun.blogspot.com/

Hola, en este artículo estaré hablando de explotar una memoria heap personalizada: la cual es un gran pedazo de memoria reservada por la aplicación en modo usuario usando VirtualAlloc, por ejemplo. La aplicación funcionará administrando bloques en "heap" y liberando (en el bloque reservado) de una manera específica con completa ignorancia de administrador de memoria heap de Windows. Este método le da al software mucho más control sobre su heap personalizado, pero puede resultar en fallas de seguridad si el administrador no hace su trabajo adecuadamente, lo veremos a más detalle posteriormente.

La vulnerabilidad que explotaremos juntos es una "heap" overflow (desbordamiento de memoria heap) que está ocurriendo en una memoria heap personalizada construida por la aplicación. El software vulnerable es:
ZipItFast 3.0

Y estaremos explotándola y ganando ejecución de código en Windows 7.
ASLR, DEP, SafeSEH no están habilitados por defecto en la aplicación lo cual lo hace más confiable para nosotros. Aun así, existen algunas sorpresas dolorosas esperándonos...

Por favor refiéranse a los cursos de C/C++ acerca de las listas simples y doblemente enlazadas para mejor entendimiento de este material.

Para ver una implementación de un administrador de memoria heap en C/C++ por favor refiéranse a mi blog previo:

http://rce4fun.blogspot.com/2014/01/creating-and-using-your-own-heapmanager.html

Código fuente : http://pastebin.com/2LgcByyC

Comencemos:

El Exploit:

Tengo de hecho la PoC (prueba de concepto) la cual puedes ver aquí:

http://www.exploit-db.com/exploits/17512/

y el exploit completo puedes verlo aquí:

http://www.exploit-db.com/exploits/19776/

Desafortunadamente, no aprenderás mucho de la explotación completa ya que únicamente funciona para Windows XP SP1,¿ Por qué ? simplemente porque está usando una técnica que consiste en sobrescribir el nodo de manejo de excepciones vectorizado que existe en una dirección estática bajo Windows XP SP1. Brevemente, todo lo que tienes que hacer es encontrar el apuntador a tu shellcode en el stack. Entonces toma la dirección del stack que apunta a tu apuntador y después de restar 0x8 de esa dirección, hacer la sobrescritura.

Cuando una excepción es generada, los manejadores de excepción serán despachados antes que cualquier manejador de cadena SEH y tu shellcode será llamado usando una llamada CALL D WORD PTR DS:[ESI + 0x8](ESI =apuntador de stack al apuntador de tu buffer-0x8).

Puedes buscar _VECTORED_EXCEPTION_NODE en google y revisar estos elementos.

¿Y por qué no funcionaría este trabajo en otras versiones de Windows?
Simplemente porque Microsoft se dio cuenta del uso de esta técnica EncodePointer y ahora es usada para codificar el apuntador al manejador, cuando un nuevo manejador es creador por la aplicación, y entonces el DecodePointer es llamado para decodificar el apuntador antes de que el manejador sea invocado.

Ok, empecemos a armar nuestro exploit desde cero. La PoC crea un archivo zip con el nombre más grande posible, intentémoslo:

Nota: Si quieres hacer algunas pruebas, ejecuta el software desde la línea de comandos como sigue:

Cmd :> C:\blabla\ZipItFast\ZipItFast.exe C:\blabla\exploit.zip

Intentemos ejecutarlo:

```
00401C6F  . 8950 04       MOV DWORD PTR DS:[EAX+4],EDX
00401C72  . 5B            POP EBX
00401C73  . C3            RETN                              EDX = entry->Blink ; [EDX] = entry->Blink->Flink
00401C74  > 8B00          MOV EAX,DWORD PTR DS:[EAX]        EAX = ptr to the FreeList entry for our block ; [EAX] = entry->Flink
00401C76  . 8902          MOV DWORD PTR DS:[EDX],EAX        List->Blink->Flink = List->Flink (EAX)
00401C78  . 8950 04       MOV DWORD PTR DS:[EAX+4],EDX      List->Flink->Blink = List->Blink (EDX)
00401C7B  > 5B            POP EBX
00401C7C  L. C3           RETN
```

Una violación de acceso ocurre en 0x00401C76 tratando de accesar a un apuntador inválido (0x41414141) en nuestro caso. Veamos los registros:

Básicamente la Lista libre FreeList usada en este software es una lista doblemente enlazada similar a la de Windows.

La cabeza de esta lista doble enlazada está en la sección .bss en la dirección 0x00560478 y sus apuntadores flink y blink están apuntando a la cabeza cuando el administrador de heap personalizado es inicializado por el software.
No revisé la implementación completa de FreeList y las operaciones de liberación y reservación para ver si eran similares a las de Windows (mapa de bits, coalescencia…etc.). Es fundamental además saber que en nuestro caso, el bloque está siendo desenlazado de la lista FreeList por que el administrador tuvo una solicitud de reservar un nuevo bloque, y este fue escogido como el mejor para la reservación.

Regresemos a analizar el crash:

Primero voy a mencionar que estaremos llamando el apuntador a la lista estructura de entrada FreeList:"entry"
Registros en 0x00401C76 :
EAX = entry->Flink
EDX = entry->Blink
[EAX] = entry->Flink->Flink
[EAX+4] = entry->Flink->Blink (Siguiente bloque Previo Bloque)
[EDX] = entry->Blink->Flink (Previo bloque Siguiente bloque)
[EDX+4] = entry->Blink->Blink

Hablando con lógica: Siguiente bloque Previo Bloque y Previo bloque Siguiente bloque no son más que el bloque actual.
Así que las 2 instrucciones que hacen el desenlace del bloque del FreeList solamente:
-Establece la entrada previa de FreeList flink a la entrada de bloque flink
-Establece la siguiente entrada de FreeList blink a la entrada de bloque blink.

Al hacer esto, el bloque ya no pertenece más a la lista FreeList y la función simplemente regresa después de eso. Así que será fácil adivinar qué está

pasando aquí, el software reserva un bloque estático en memoria heap para almacenar el nombre del archivo y sería mejor reservar el bloque basado en la longitud del archivo del encabezado de ZIP (este podría ser un fix para ese bug, pero desbordamientos de heap podrían encontrarse dondequiera, Mejor propondré un método más eficiente para reparar pero no totalmente este bug en este artículo).

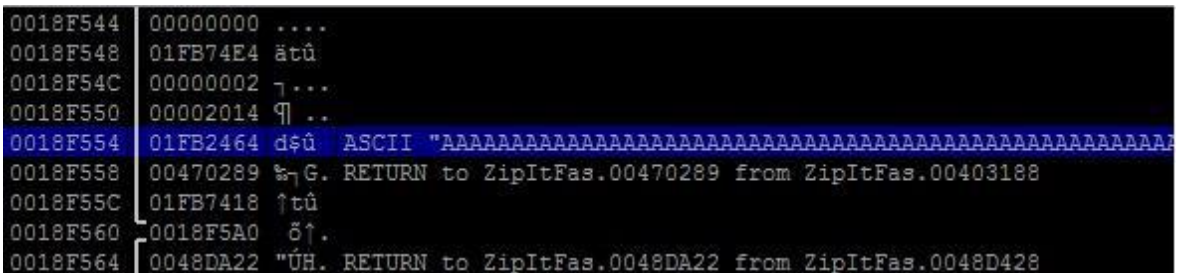
Ahora, sabemos que estamos describiendo nuestro bloque de heap y de este modo sobrescribimos los metadatos personalizados del siguiente bloque heap (apuntadores flink y blink). Así que, necesitaremos encontrar una forma confiable de explotar este bug, ya que como las dos instrucciones desenlazantes son las únicas disponibles para nosotros, y que nosotros controlamos ambos EAX y EDX,

Si no es posible, en otro caso, puedes ver si hay otras instrucciones de cierre que pueda ayudar), puedes pensar en sobrescribir la dirección de retorno o el apuntador al manejador de excepciones estructurado ya que tenemos un stack que no será rebasado después de reinicio.

Esta puede ser una solución funcional en otro caso donde tu buffer está almacenado en una localidad de memoria estática.
En Windows 7, no es el caso, VirtualAlloc reserva un bloque de memoria con una base diferente en cada programa ejecutado. En adición, aun si la dirección era estática, la localización del bloque liberado que nosotros sobrescribimos varia, así que en ambos casos necesitaremos encontrar un apuntador a nuestro buffer.

El mejor lugar para buscar es en el stack, recuerda que el software está intentando desenlazar (reservar) el bloque que sigue al bloque donde hemos escrito el nombre, de esta forma todos los apuntadores cercanos en el stack (el stack frame actual y el previo) están apuntando al nuevo bloque reservado (apuntador a metadatos). Esto es lo que nosotros no queremos ya que flink y blink que podemos establecer, pueden no ser opcodes válidos y pueden causar excepciones, así que todo lo que necesitamos hacer es intentar encontrar un apuntador al primer carácter del nombre, y entonces imaginar cómo usar el apuntador para ganar la ejecución del código, este apuntador puede estar en stack frames previos.

Y aquí está un apuntador apuntando al inicio de nuestro buffer: a 3 stack frames de distancia



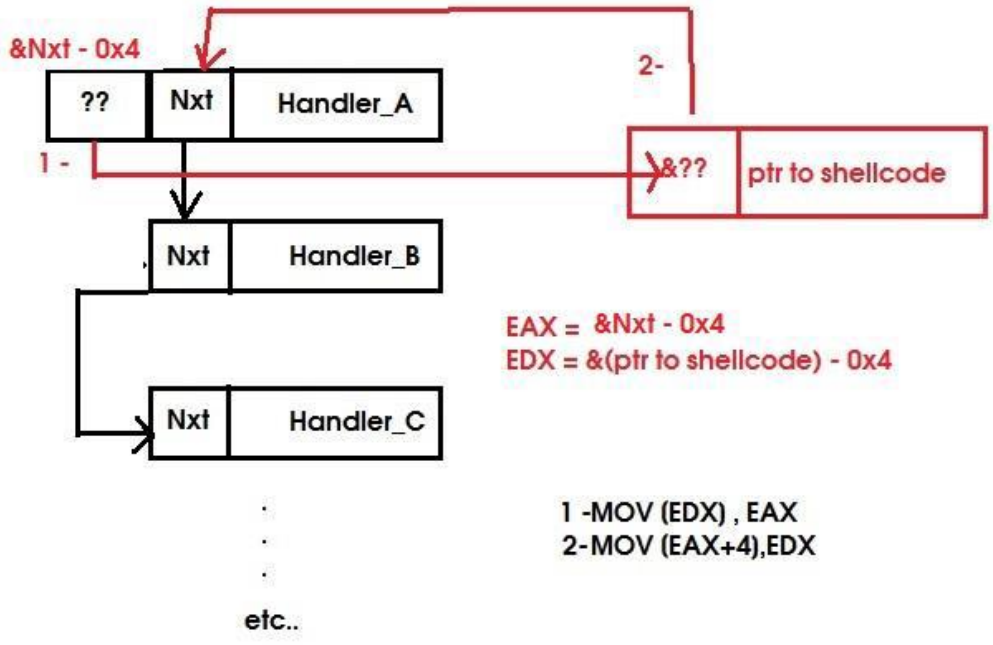
Recuerda que 0x01FB2464 ciertamente va a ser algo más cuando reinicies el programa, pero el apuntador 0x0018F554 será siempre estático, aun cuando se reinicie la máquina.
Un uso para este apuntador sería sobrescribir el nodo manejador vectorizado con 0x0018f554 - 0x8 en Windows XP SP1, pero no es el caso aquí **en absoluto**
Así que cuando yo estaba en esta fase, empecé a pensar acerca de la forma que me ayudaría a redirigir la ejecución de mi shellcode la cual esta seguramente en la dirección apuntada por 0x0018F554, y para usar únicamente lo que tengo disponible:
-Registros controlados: EAX y EDX
-Apuntador a stack a un apuntador de buffer dinámico
-2 instrucciones desenlazantes
-No rebasar el stack

Explotando la vulnerabilidad y ganando la ejecución del código:

Y luego pensé, ¿porque yo querría corromper la cadena SEH y crear un frame falso? Porque cuando intento crear una cadena SEH falsa hay 3 cosas que debes saber:
-SafeSEH y SEHOP están ausentes
-Tienen un apuntador a un Frame SEH existente
-Tienen un apuntador a un apuntador al shellcode

El apuntador al shellcode debe ser tratado como el manejador, y el valor en (ptr a ptr a shellcode) -0x4 debe ser tratado como el apuntador al siguiente Frame SEH.

Ahora ilustremos el acto de corromper la cadena:(con una tonta ilustración, lo siento)



Déjenme explicar:

Necesitamos lograr nuestra meta usando 2 instrucciones:

MOV [EDX],EAX

MOV [EAX+4], EDX
Necesitaremos 2 apuntadores y controlaremos 2 registros, ¿pero cuál apuntador te da cada registro? Esto debe ser una elección aleatoria ya que debes sobrescribir el apuntador al shellcode si elegiste EAX como apuntador a tu Frame SEH falso.

En adición nosotros no necesitamos saber acerca del valor del "siguiente Frame SHE" de nuestro Frame Falso.
Así que nuestra meta principal es sobrescribir el apuntador al "siguiente frame SEH" de un Frame existente, para llegar a esto necesitamos tener el apuntador a nuestro frame falso en uno de los 2 registros. Como [EAX+4] sobrescribirá el apuntador al buffer si es usado como apuntador al frame SHE falso, usaremos EDX en su lugar.

Tampoco debemos sobrescribir el apuntador al manejador original debido a que será primeramente ejecutado para tratar de manejar la excepción, si falla, nuestro manejador falso (el shellcode) será invocado entonces.
Así:

EDX = & (apuntador a shellcode) - 0x4 = Apuntador a elemento "Next SEH frame" del frame falso.

EDX debe residir en el campo siguiente frame del frame original el cual es: [EAX+4]. y **EAX = SEH Frame - 0x4.**

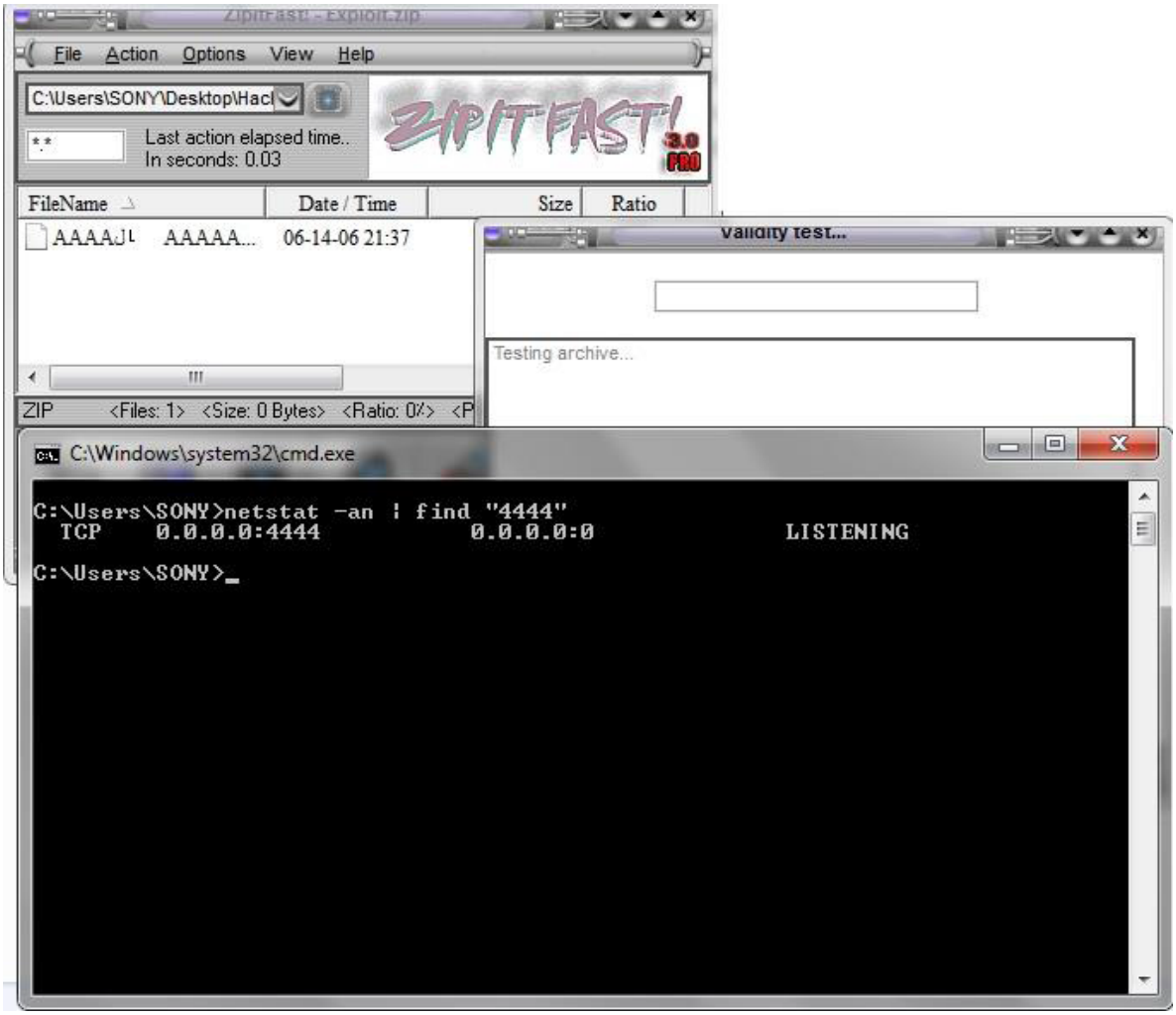
Frame original después de sobre escritura:

Apuntador al siguiente SEH: Frame Falso
Manejador de excepciones: Manejador Válido

Frame Falso:
Apuntador al siguiente SEH: (Frame Original) - 0x4 (este no nos importa mucho)
Manejador de excepciones: apuntador al shellcode
El frame SEH que escogí esta en: 0x0018F4B4
Así que: EAX = 0x0018F4B4 - 0x4 = **0x0018F4B0** y EDX =0x0018F554 - 0x4 = **0x0018F550**

Ahora todo lo que tenemos que hacer es calcular la longitud entre el primer carácter del nombre y los apuntadores flink y blink, y entonces insertar nuestros apuntadores en la POC.

Insertando el shellcode:
El espacio entre la dirección de inicio del buffer y los metadatos sobrescritos del heap no es tan grande, así que es mejor poner un **salto incondicional en el comienzo de nuestro buffer pasando el flink y blink sobrescritos**
Y entonces poner el shellcode justo después de los apuntadores. Mientras podamos calcular el tamaño, no tendremos problema.
El exploit final está aquí: http://pastebin.com/pKyQJicy
Escogí un shellcode bind, el cual abre una conexión a (0.0.0.0:4444). Intentemos abrir el archivo ZIP usando ZipItFast y entonces verifiquemos "netstat -an | find "4444" :



¿Una solución a esta vulnerabilidad?

El método que les mencione anteriormente consiste en reservar el bloque basado en la longitud del nombre de archive de los encabezados de ZIP puede ser válido solamente para reparar la vulnerabilidad de este caso, pero que pasa si el atacante fuera capaz también de causar un overflow en otra parte cualquiera del software?

La mejor forma de arreglar el bug es que: cuando un bloque está a punto de ser reservado y está por ser desenlazado de la lista FreeList, la primera cosa que debe hacerse es revisar la validez de la lista doble enlazada
, para hacerlo: debes ejecutar el **desenlazado seguro** el cual fue introducido en versiones posteriores de Windows

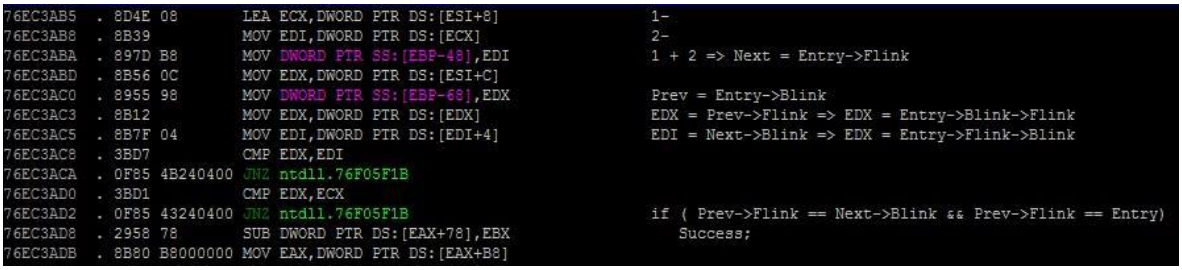
El desenlazado seguro se realiza de la siguiente forma:

```
if ( entry->flink->blink != entry->blink->flink || entry->blink->flink != entry){
//Fallo , Freelist esta corrompida , salir del proceso
}
else {
```

//Desenlace, entonces regrese el bloque al llamante (caller)
}

Veamos qué tan seguro es ejecutado el Desenlace en Windows 7:

La función que buscamos es: **RtlAllocateHeap** exportado por ntdll



Aún si el método parece seguro, hay algunas investigaciones publicadas en línea que muestran debilidades en esta técnica y como puede ser evadida. También me aseguré de implementar esta técnica en mi manejador de heap personalizado (**Línea 86**), el enlace está arriba.
Espero que hayan disfrutado este artículo… y se aseguren de revisar mi blog por más cosas interesantes.

Mis mejores deseos
Souhail Hammou.