UNDERCODE

Taller. Bug Hunting

Taller dedicado al Analisis de Software en busca de Fallas

Tutores

MaztoR - DarkSpark - SnaKingMax

Temas tratados

- Errores de programacion
- Pruebas de Software
- lo interno del Software

Analizando el interior!

En Busca de Fallas en un Software

MAESTRO

En este taller vamos a ver cómo se detectan, a qué se deben, y cómo podemos encontrar bugs en el software.

Lo primero que debemos hacer cuando programamos en un nuevo lenguaje es **leer acerca de las funciones peligrosas** (funciones que suelen derivar en problemas de seguridad) y los consejos de **cómo programar código seguro para dicho lenguaje.**

Hay varias pruebas que podemos aplicar a nuestros programas para comprobar si son seguros. Aunque, sin duda, lo mas importante es programar correctamente.
Entre ellas están las siguientes:

Pruebas de caja blanca: Estas se centran en detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente.
Ya que.. **Disponemos del código fuente.**
El testeador escoge distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados.
Estas pruebas suelen abarcar: El flujo de control, el de flujo de datos, bifurcación (*branch testing*), caminos básicos.

Para ello se utilizan herramientas como son los IDE debuggers just in time.

Pruebas de caja negra: El software se analiza desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno.
Esto se debe a que **no disponemos del código fuente** del software.
Cuando se analiza un malware, por ejemplo, no tienes el código de fuente y eso se convierte en lo que se denomina una putada increíble.
Para estas pruebas se utilizan herramientas como: Sniffers, desambladores, editores hexadecimales, debuggers, software que recopile información sobre el programa, scanners de vulnerabilidades, fuzzers, y sobre todo testear el software.

Pruebas de caja gris: Como podéis intuir viene siendo una mezcla entre las dos anteriores.

Pruebas de caja blanca

1 – El programador:
Una de las cosas que influyen mucho a la hora de que un software contenga o no fallas, es la calidad de los programadores.
Existen funciones peligrosas en los lenguajes de programación con las que hay que tener un especial cuidado.
En ocasiones se recomienda desactivarlas, por ejemplo, si programamos en PHP. Conviene utilizar la directiva disable_functions en el php.ini
Este sería un ejemplo:

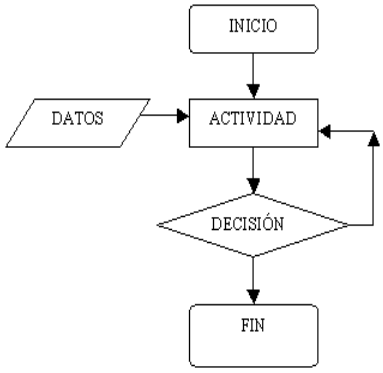
```
disable_functions
="system,passthru,escapeshellarg,escapeshellcmd,proc_close,proc_open,ini_alter,popen,sho
w_source,pcntl_exec"
```

En C, por ejemplo, la mala utilización de getenv, setenv, strcpy... entre otras funciones, puede desencadenar en un desbordamiento del buffer, vulnerabilidad que, en muchas ocasiones, puede ser aprovechada por un atacante para obtener los mismos privilegios que el programa vulnerable.
Hay que tener cuidado, filtrar las entradas de datos, y sobre todo, pensar en los casos anormales porque para el uso normal el software suele funcionar. Pero siempre hay algún hincha-pelotas.... xD

El correcto uso de la visibilidad de las variables, atributos, clases... etc, etc, etc. Programar bien.

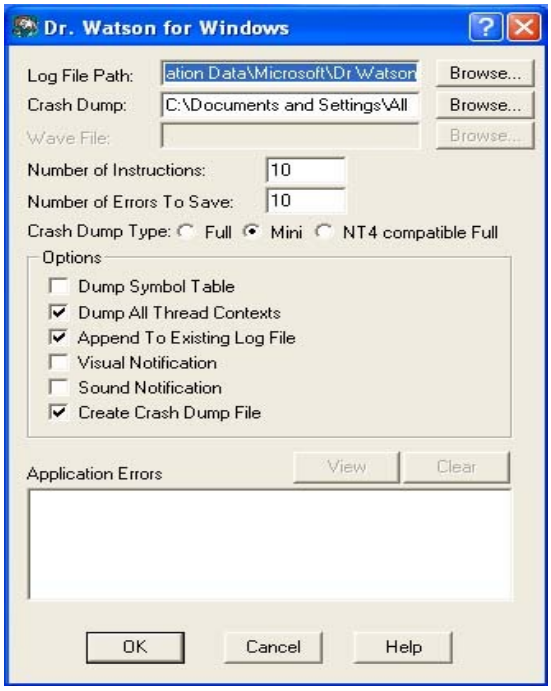
2 -El flujo de control:

Radica en comprobar el orden en que se ejecutan las instrucciones, comprobar que, al dar valores poco comunes en las bifurcaciones y todo eso el programa vaya por donde tiene que ir, sin saltar a sitios que no debe.



3 -Bifurcación (*branch testing*):

Comprueba que se hayan cubierto todas las opciones en las bifurcaciones.

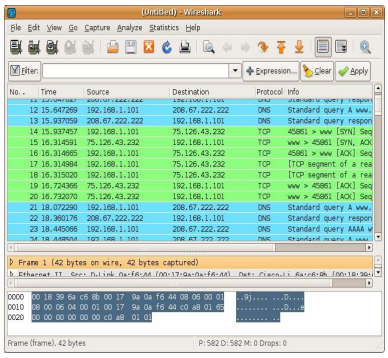


3 – Debuggers just in time:

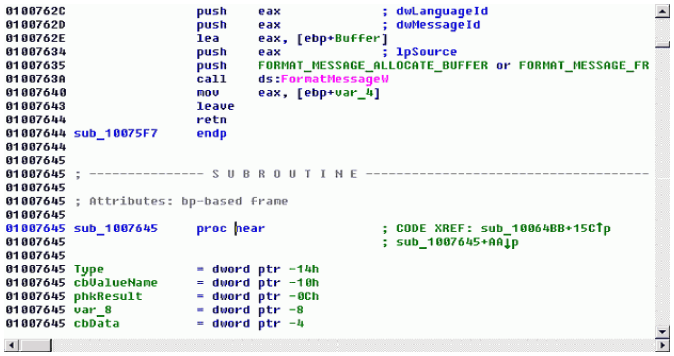
Cuando un programa falla, salta tu debugger just in time y te permite ir a la línea exacta donde se produjo el error.

Pruebas de caja negra

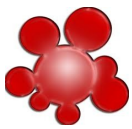
1 – Sniffers:
Analizamos el tráfico entrante y saliente de nuestra aplicación en busca de fallas de seguridad como: MITM, datos que deberían ir encriptados pero no lo hacen, robos de sesiones y demás catástrofes imaginables.



2 – Desensambladores:
Traducen nuestro ejecutable a ensamblador. No tenemos el código fuente pero podemos traducirlo a el casi código máquina. En ocasiones está empacado, comprimido... y no entendemos una mierda de lo que hace.. pero es lo que hay. Nadie dijo que fuese sencillo.

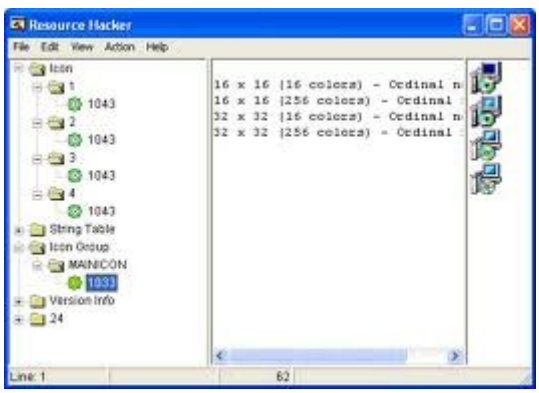


```
01007620                push    eax             ; dwLanguageId
01007621                push    eax             ; dwMessageId
01007622                lea     eax, [ebp+Buffer]
01007634                push    eax             ; lpSource
01007635                push    FORMAT_MESSAGE_ALLOCATE_BUFFER or FORMAT_MESSAGE_FR
0100763A                call    ds:FormatMessageW
01007640                mov     eax, [ebp+var_4]
01007643                leave
01007644                retn
01007644 sub_10075F7    endp
01007644
01007645
01007645 ; ---------------  S U B R O U T I N E  ---------------------------------------
01007645
01007645 ; Attributes: bp-based frame
01007645
01007645 sub_1007645    proc near               ; CODE XREF: sub_10064BB+15C↑p
01007645                                        ; sub_1007645+AA↓p
01007645
01007645 Type           = dword ptr -14h
01007645 cbValueName    = dword ptr -10h
01007645 phkResult      = dword ptr -0Ch
01007645 var_8          = dword ptr -8
01007645 cbData         = dword ptr -4
```

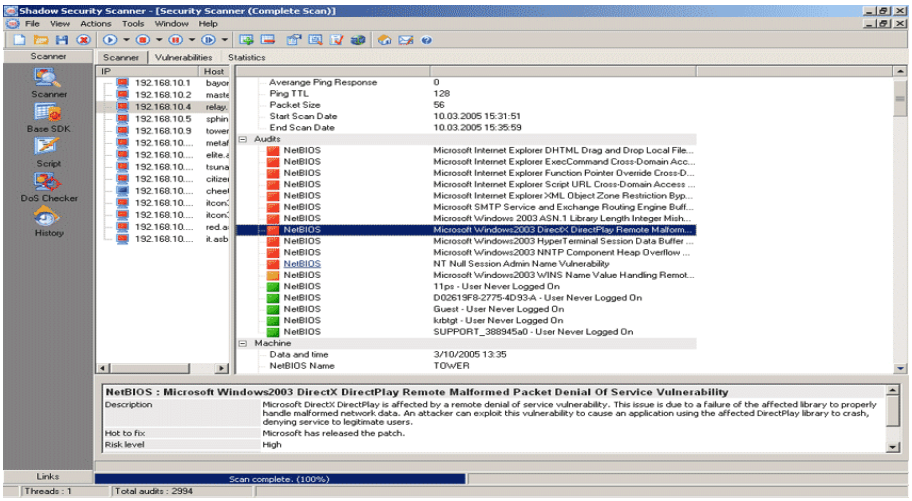


3 – Debuggers:
Nos permiten ejecutar el programa y hacerle cosas. Meterle breakpoints (pausan la ejecución del programa en algun punto), ir viendo cómo responde a las entradas que le metemos.... Eso sí, desde nuestro lenguaje favorito, ASM. XD
A veces el ejecutable está protegido contra debuggers peeeero hay formas de saltarse esas protecciones.

4 – Software que recopile información sobre el programa:
Bien, con esto me refiero a programas como el resource hacker y otros similares, que simplemente extraen información del programa e incluso alguno te permite parchear el ejecutable.



5 – Analizan en busca de vulnerabilidades conocidas:
Entre ellos los que buscan fallas de XSS, Inyecciones varias, versiones de software con fallas...



6 – Fuzzers:
Los fuzzers son programitas que ponen a prueba la entrada de datos en nuestros programas (o en los de otros xD), basicamente sirven para enviar cadenas pseudoaleatorias como entrada y esperar a que estas produzcan algun error que no se ha previsto.

Software que Utilizaremos

Software que será necesario para el taller:

- Sistema Operativo Ubuntu linux x86.
  http://www.ubuntu.com/download/ubuntu/download

- Recomendable utilizar VirtualBox para virtualizar ubuntu... pero no es necesario.

- Software necesario:
gdb
nasm
ndisasm
objdump
hexdump
execstack
ld

De momento simplemente quiero que os vayáis familiarizando con las herramientas y los conceptos que vamos a utilizar en el taller.

Se que es un paper muy muy muy báaaasico y que algunos esperarían algo mas, pero es el primer paper.

En el próximo comenzaremos ya con las prácticas, es decir, supongo que veremos algo de ensamblador fuzzearemos alguna aplicación y a jugar con la sorpresita que nos encontremos.

Hasta entonces... Un saludo,
SnakingMax