Muerte

Resumen: En este proyecto, codificarás tu último virus "metamórfico".

Versión: 4

Contenido

Capítulo I

Preámbulo

Capítulo II

Introducción

Si aceptas este proyecto, en nombre de todo el personal queremos felicitarte. Has logrado "grandes cosas", pero aún tienes que asumir el reto máximo de tu formación como virólogo.

Tras la oligomorfía con Pestilence y la polimorfía con War, emprenderás un viaje por el camino metamórfico. La característica principal de un virus metamórfico reside en su capacidad de modificar su estructura interna así como las instrucciones que lo componen.

El metamorfismo es un mecanismo de defensa utilizado por diferentes programas para pasar desapercibidos ante programas de control y protección como los antivirus.

Un virus metamórfico no contiene ni un decodificador ni un cuerpo viral constante, pero puede crear nuevas generaciones de sí mismo para cada replicación, evitando al mismo tiempo que una generación se parezca a la anterior.

Capítulo III

Objetivos

¡Ahora, que empiece la parte divertida! Gracias al virus que ya has desarrollado, has descubierto métodos de ofuscación con programación polimórfica... Pero aún te quedan algunas cosas divertidas por descubrir. Para este proyecto, vas a tener que pensar de forma innovadora. Los métodos que has utilizado hasta ahora para encontrar soluciones a una problemática determinada van a ser seriamente cuestionados.

De hecho, además de las funciones que has implementado en los proyectos anteriores, este proyecto requerirá que crees un código cuya estructura interna se desarrollará parcialmente. Para ello, tendrás que utilizar el lenguaje de máquina. Solo tendrás que desarrollar un último programa con varias propiedades que ya has aprendido. Pero vamos a hacer que nuestro programa sea mucho más interesante.

En cuanto a su dificultad, este proyecto no está hecho para cualquiera. Si estás dispuesto a emprenderlo, necesitarás motivación y nunca debes rendirte. El resultado es muy gratificante :)

Capítulo IV

Parte obligatoria

La muerte es un binario de tu propio diseño que tendrá que:

- como Famine, infecta binarios presentes en 2 carpetas específicas diferentes y aplica su sign- naturaleza sin alterar el modo en que funciona dicho binario.
- al igual que Pestilence, no desencadena el proceso de infección si se está ejecutando un proceso específico, si el programa se inicia desde cualquier depurador y presenta una parte de la rutina de infección de manera ofuscada.
- como la guerra, lleva una HUELLA DIGITAL que se desarrolla con cada nueva ejecución de rutina de infección. Cución.

La firma tendrá que parecerse remotamente a esto:

```
Versión 1.0 de D34TH codificada por <first-login> - <second-login> - [HUELLA DIGITAL]
```

Ahora, pasemos a la parte loca de este tema. Tómate todo el tiempo que necesites para leer esta parte una y otra vez: debes asegurarte de que tu programa nunca será estructuralmente el mismo, una vez que se haya ejecutado.

Para ello, solo tendrás que aprender a volver a desarrollar tu virus para que la parte infecciosa pueda desarrollarse. Deberías poder infectar dos archivos idénticos que muestren una diferencia real en su código de máquina. Por supuesto, los archivos infectados deberían poder ejecutarse sin problemas.

Instrucciones generales:

- El archivo ejecutable se llamará Muerte.
- Este ejecutable se codificará como un ensamblador, en C o C++. Nada más. Se tolerará Java para este proyecto y eso es todo.
- Su programa no mostrará nada en la salida estándar o el puerto de error.
- TENDRÁS QUE trabajar en una máquina virtual.

- Como es habitual, puedes elegir el sistema operativo de destino. No obstante, tendrás que configurar una máquina virtual apropiada durante la evaluación.
- Su programa tendrá que actuar en las carpetas /tmp/test y /tmp/test2 (y solo en estas carpetas) o su equivalente, dependiendo del sistema operativo de destino. Usted es responsable de la propagación de su programa.
- ¡ADVERTENCIA! Solo será posible una infección en el binario seleccionado.
- Las infecciones comenzarán en archivos binarios de su tipo de sistema operativo creados en una arquitectura de 64 bits.

Capítulo V

Ejemplos de uso

He aquí un ejemplo de uso:

Sentemos las bases:

```
# ls -al ~/Total de muertes
736
drwxr-xr-x 3 root root 4096 24 de mayo 08:03 . drwxr-xr-x 5 root root 4096
24 de mayo 07:32 .. -rwxr-xr-x 1 root root 744284 24 de mayo 08:03 Muerte
```

Creamos una muestra .c para nuestras pruebas:

```
# nl muestra.c 1
 #include <stdio.h>
2 int 3
main(void) { printf("Esto
        es difícil...\n"); 4 5 return 0; 6 } # gcc -m64 ~/Virus/
sample/sample.c


#
```

Copiamos nuestros binarios (tests + ls) para nuestras pruebas:

```
# cp ~/Virus/sample/sample /tmp/test2/. # ls -al /tmp/test total 16 drwxr-
xr-x 2 root root 4096 24 de mayo
08:07 .
drwxrwxrwt 13 root root 4096 24 de mayo 08:08 .. -rwxr-xr-x 1 root root 6712
24 de mayo 08:11 sample # /tmp/test/sample ¡Hola, mundo! # archivo /tmp/test/
sample /tmp/test/sample: ejecutable LSB ELF de 64 bits, x86-64, versión 1 (SYSV),
enlazado dinámicamente,
intérprete /lib64/ld-linux-
x86-64.so.2, para GNU/Linux 2.6.32,
BuildID[sha1]=938[...]10b, no despojado

# cadenas /tmp/prueba/muestra | grep "wandre" # cp /bin/ls /tmp/prueba2/ #
ls -al /tmp/prueba2 total 132


drwxr-xr-x 2 root root 4096 24 de mayo 08:11 . drwxrwxrwt 14 root root 4096
24 de mayo 08:11 .. -rwxr-xr-x 1 root root 126480 24 de mayo 08:12 ls # archivo /
tmp/test2/ls / tmp/test2/ls: ejecutable LSB ELF de 64 bits, x86-64, versión 1
(SYSV), enlazado dinámicamente,
intérprete / lib64/ld-linux-x86-64.so.2, para GNU/Linux 2.6.32, BuildID[sha1]=67e[...]281, despojado

#
```

Lanzamos Death sin el proceso de "test" y observamos el resultado:

```
# pgrep "prueba"
# ./Muerte #
cadenas /tmp/test/sample | grep "wandre" virus.custom versión 1.3 (c)oded
may-2017 por wandre - 42424242 # /tmp/test/sample

Esto es difícil..
# cadenas /tmp/test2/ls | grep "wandre" virus.custom versión 1.3
(c)odada mayo-2017 por wandr - 43434343 # /tmp/test2/ls -la /tmp/test2/ total 132


drwxr-xr-x 2 root root 4096 4 de mayo 12:13 . drwxrwxrwt 14 root root
4096 4 de mayo 12:11 .. -rwxr-xr-x 1 root root xxxxxx 4 de mayo 12:19 ls #
gcc -m64 ~/Virus/sample/sample.c -o /tmp/test/sample # ls -al /tmp/test total 16
drwxr-xr-x 2 root root 4096 4 de mayo 12:13 . drwxrwxrwt 13 root root 4096 4 de mayo 12:11 ..
-rwxr-xr-x 1 root root xxxx 4 de
mayo 12:19
sample # /tmp/test/sample



Esto es difícil..
# archivo /tmp/test/sample /tmp/test/
sample: ejecutable LSB ELF de 64 bits, x86-64, versión 1 (SYSV), vinculado dinámicamente, intérprete
        /lib64/ld-linux-x86-64.so.2, para GNU/Linux 2.6.32, BuildID[sha1]= xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx,
    no se ha eliminado # cadenas /tmp/test/sample | grep "wandre" # /tmp/test2/ls -la /tmp/test2/ total
132 drwxr-xr-x 2 root root 4096 3 de mayo 12:03 . drwxrwxrwt 14 root root
4096 3 de mayo 12:01 .. -rwxr-xr-x 1 root root xxxxxx 3
de mayo 12:20
ls # cadenas /tmp/test/sample | grep "wandre" virus.custom versión 1.3
(c)oded may-2017 por wandre - 444444



#
```

Vemos muy claramente el desarrollo de la "firma". Por supuesto, tendrás que asegurarte de que la modificación sea claramente visible. La parte de la firma que se supone que se va a desarrollar debe estar controlada. Si es aleatoria, también lo será tu calificación. Recuerda: no debe producirse una doble infección. Para esta parte, te sugiero que desarrolles tu lógica como ensamblador. Puede que obtengas algunos resultados con algunos métodos de C/C++, pero creo que perjudicará seriamente el logro de este proyecto.

Lanzamos Muerte con el proceso de "prueba" junto con el entorno inicial y observamos El resultado:

```
# pgrep "test" 57452 # ./
Muerte
# cadenas /tmp/
test/sample | grep "wandre" # /tmp/test/sample Esto es difícil... # cadenas /
tmp/test2/ls | grep "wandre" # /
tmp/test2/ls -la /tmp/test2/
total 132



drwxr-xr-x 2 root root 4096 4 de mayo 12:03 . drwxrwxrwt 14 root root 4096
4 de mayo 12:01 .. -rwxr-xr-x 1 root root xxxxxx 4 de mayo 12:21 ls # gcc -m64
~/Virus/sample/sample.c -o /tmp/test/sample # ls -al /tmp/test total 16 drwxr-xr-x
2 root root 4096 3 de mayo 12:03 . drwxrwxrwt 13 root root 4096 3 de mayo 12:01 .. -rwxr-xr-x 1 root
root xxxx 3 de mayo 12:21
sample # /tmp/
test/sample



Esto es difícil... # archivo /
tmp/test/sample /tmp/test/sample:
ejecutable ELF LSB de 64 bits, x86-64, versión 1 (SYSV), vinculado dinámicamente, intérprete
        /lib64/ld-linux-x86-64.so.2, para GNU/Linux 2.6.32, BuildID[sha1]= xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx, no
    se ha eliminado # cadenas /tmp/test/sample | grep "wandre" # /tmp/test2/ls -la /tmp/test2/ total 132
drwxr-xr-x 2 root root 4096 3 de mayo 12:03 . drwxrwxrwt 14 root root 4096 3
de mayo 12:01 .. -rwxr-xr-x 1 root root xxxxxx 3 de mayo
12:23 ls #
cadenas /tmp/test/sample | grep "wandre" #
```

Vamos a intentar ejecutar Death en el depurador. Voy a usar gdb. Escribí una pequeña nota para que quede más claro:

```
# gdb -q ./Muerte (gdb) run
 Programa de
inicio: /root/Muerte
DEPURACIÓN..
 [Inferior 1 (proceso 2683) salió con el código 01] # cadenas /tmp/test/sample | grep
"wandre" # /tmp/test/sample

Esto es difícil..
# cadenas /tmp/test2/ls | grep "wandre"
#
```

Y ahora, pasemos a la parte metamórfica. Vamos a tomar dos muestras idénticas que, naturalmente, llamaremos sample1 y sample2 y haremos una diferencia para contrarrestar la cantidad de diferencias con una simple diferencia. Para mayor claridad, voy a hacer que sea lo más fácil de leer posible.

```
# nl muestra.c 1
 #include <stdio.h>
2 int 3
main(void) { 4 printf("Esto
es difícil..\n"); return 0;
5
6 } #
gcc -m64 ~/Virus/sample/sample.c -o /tmp/test/sample1 # cp /tmp/test/sample1 /tmp/test/sample2 # /
tmp/test/sample1 Esto es difícil... # /tmp/test/sample2 Esto es difícil... #
objdump -D /tmp/test/sample1 >
samp ; objdump -D /tmp/
test/sample2 > samp2 ; diff -y --
suppress-

        líneas comunes samp samp2 | grep                '^' | wc-l
1
```

No notarás ninguna diferencia aquí, excepto por el nombre del programa. La diferencia solo te mostrará la cantidad diferente de líneas entre cada binario en el nivel de código de máquina. Vamos a ejecutar el virus y observar los resultados.

```
# ./Muerte # /
tmp/test/sample1 Esto es difícil...
# /tmp/test/sample2 Esto
es difícil... # cadenas /tmp/test/
sample1 | grep "wandre"
virus.custom versión 1.3 (c)oded may-2017 por wandre - 41414141 # cadenas /
tmp/test/sample2 | grep "wandre" virus.custom versión 1.3 (c)oded may-2017 por wandre - 42424242 # objdump
-D /tmp/test/sample1 > samp ; objdump -D /tmp/test/sample2 > samp2 ; diff -y
--suppress-

        líneas comunes samp samp2 | grep                '^' | wc-l
165
#
```

No hace falta profundizar más. Ya sabes lo que hay que hacer. Tienes mucho trabajo por delante.

Capítulo VI

Parte extra

Ideas extra:

- Poder infectar binarios de 32 bits.
- Poder infectar todos los archivos a partir de la raíz de tu sistema operativo de forma recursiva.

Optimizarás esta parte ejecutando binarios infectados.

- Permitir la infección en archivos no binarios.
- Utilizar métodos de empaquetado directamente sobre el virus para hacer el binario lo más ligero posible.
- Añadir un metamorfismo total a su programa (a juzgar por la dificultad del tarea, esto validará toda la sección de bonificación).

La parte adicional solo se evaluará si la parte obligatoria es PERFECTA. Perfecto significa que la parte obligatoria se ha realizado íntegramente y funciona sin fallas. Si no ha aprobado TODOS los requisitos obligatorios, su parte adicional no se evaluará en absoluto.

Capítulo VII

Presentación y evaluación por pares

Entregue su tarea en su repositorio de Git como de costumbre. Solo el trabajo dentro de su repositorio será evaluado durante la defensa. No dude en volver a verificar los nombres de sus carpetas y archivos para asegurarse de que sean correctos.

- Se controlará la rutina que hace que su programa sea "polimórfico". Esto es muy importante.
- Tendrás que explicar tu parte metamórfica.
- Debes gestionar los errores de forma razonable. Tu programa nunca debe cerrarse sin previo aviso. Previsiblemente (fallo de segmentación, etc.).
- Durante la evaluación, deberás estar en una máquina virtual. Para tu información, la escala de calificación Fue construido con un Debian 7.0 estable de 64 bits.
- Puedes usar cualquier cosa que necesites, excepto las bibliotecas, que harán el trabajo sucio por ti. Esto se consideraría hacer trampa.