MÁSTER EN ANÁLISIS DE MALWARE Y REVERSING

MÓDULO 3. ANÁLISIS DE CÓDIGO FUENTE

TAREA 1

MÁSTER EN *ANÁLISIS DE MALWARE Y REVERSING*

Campus Internacional CIBERSEGURIDAD

ENIIT INNOVA IT BUSINESS SCHOOL

UCAM UNIVERSIDAD CATÓLICA DE MURCIA

1. EJERCICIO

Cuando nos dan un código para que lo estudiemos, lo primero de todo es averiguar en que lenguaje está escrito. Ya sea ensamblador o un lenguaje de alto nivel, nuestro primer movimiento es observar los detalles del código para responder las siguientes preguntas: ¿Es un ensamblador o es código de alto nivel? ¿El sistema de tipos es dinámico o estático? ¿Es un lenguaje compilado o se trata de un lenguaje interpretado?

Para ello, vamos a entrenar la vista con trozos (la expresión angosajona sería "snippet") de código sobre los que debéis averiguar en que lenguaje esta escrito e indicar como lo habéis averiguado. Es decir, no basta con decir: "Es Perl."

Debereis comentar sobre qué os habéis basado para responder.

1.1. TROZO 1

```
func InMemLoads() (map[string]Image, error) {
        ret := make(map[string]Image)
        s, si, p := GetModuleLoadedOrder(0)
        start := p
        i := 1
        ret[p] = Image{uint64(s), uint64(si)}
        for {
                s, si, p = GetModuleLoadedOrder(i)
                if p != "" {
                        ret[p] = Image{uint64(s), uint64(si)}
                }
                if p == start {
                        break
                }
                i++
        }

        return ret, nil
}
```

1.2. TROZO 2

```
export default class Encoder {
    data: number[] = []

    pushByte(value: number) {
        this.data.push(value & 0xff)
    }
    pushInt(value: number, n: number, littleEndian=false) {
        for (let i = 0; i < n; i++) {
            const curShift = littleEndian ? i : n - 1 - i
            this.data.push((value >> (curShift * 8)) & 0xff)
        }
    }
    pushInt20(value: number) {
        this.pushBytes([(value >> 16) & 0x0f, (value >> 8) & 0xff, value & 0xff])
    }
    pushBytes(bytes: Uint8Array | Buffer | number[]) {
        bytes.forEach (b => this.data.push(b))
    }
```

1.3. TROZO 3

```
Option Explicit
Sub PDF2Workbook()

      Application.Run "PDFTables2Workbook", , True

End Sub

Sub ShowPagesLeft()

      MsgBox PDFTablesPages, vbOKOnly + vbInformation, " PDFTables"

End Sub
Sub PDFTables2Workbook(Optional ByVal InitialFolderFile As String = vbNullString, Optional ByVal AllowMultiSelect As Boolean = False)
```

1.4. TROZO 4

```
long uv__idna_toascii(const char* s, const char* se, char* d, char* de) {
  const char* si;
  const char* st;
  unsigned c;
  char* ds;
  int rc;

  ds = d;

  si = s;
  while (si < se) {
    st = si;
    c = uv__utf8_decode1(&si, se);

    if (c == -1u)
      return UV_EINVAL;
```

1.5. TROZO 5

```
Out-File -FilePath $PayloadPath -InputObject $Payload -Encoding ascii
$OSVersion = (Get-WmiObject -Class win32_OperatingSystem).BuildNumber
switch($method)
{

    "Sysprep"
    {
        Write-Output "Using Sysprep method"
        if ($OSVersion -match "76")
        {
            Write-Verbose "Windows 7 found!"
            $dllname = "CRYPTBASE.dll"
            $PathToDll = "$env:temp\$dllname"
            Write-Verbose "Writing to $PathToDll"
            [Byte[]] $temp = $DllBytes -split ' '
            [System.IO.File]::WriteAllBytes($PathToDll, $temp)
        }
```

2. EJERCICIO

Ahora vamos a centrarnos en las funciones, que son el elemento básico en casi todos los lenguajes de programación.

Cuando estamos analizando código, una de las primeras tareas es ir siguiendo la pista de las llamadas a funciones. Es clave fijarse en la signatura de una función, ver que parámetros necesita, cuales poseen valores por defecto, que tipo de dato devuelve, que otras funciones llama, ¿modifica algún parámetro o solo lee los valores recibidos?, ¿devuelve un objeto creado dentro de ella?, ¿modifica objetos globales?, etc.

Si entrenamos nuestra capacidad para clasificar funciones, podremos distinguir de nuestro análisis aquellas que son importantes o clave de otras más auxiliares o cuya importancia es segundaria e incluso funciones que no son alcanzables salvo condiciones poco probables (tratamiento de excepciones raras, etc)

Por cada función presentada, se pide que desarrolléis un comentario sobre esta y sus atributos, entendiendo por estos los siguientes:

- Si es función o procedimiento
- Signatura de la función (debéis desarrollar que tipo de parámetros posee)
- Tipo de la función
- Si es o no predicado
- Si es o no función hoja
- Si es o no función de sistema
- Si muta o lee valores fuera de su ámbito. Es decir, valores no creados dentro de la función o que no forman parte de los parámetros de la función
- Si posee o no parámetros constantes
- Si posee o no parámetros por defecto y que valores poseen
- Si es o no una función con parámetros variables

Vamos a estudiar un ejemplo:

```
void dump_disassembly(int indent, struct bytecode* bc) {
  if (bc->nclosures > 0) {
    printf("%*s[params: ", indent, "");
    jv params = jv_object_get(jv_copy(bc->debuginfo), jv_string("params"));
    for (int i=0; i<bc->nclosures; i++) {
      if (i) printf(", ");
      jv name = jv_array_get(jv_copy(params), i);
      printf("%s", jv_string_value(name));
      jv_free(name);
    }
    jv_free(params);
    printf("]\n");
  }
  dump_code(indent, bc);
  for (int i=0; i<bc->nsubfunctions; i++) {
    struct bytecode* subfn = bc->subfunctions[i];
    jv name = jv_object_get(jv_copy(subfn->debuginfo), jv_string("name"));
    printf("%*s%s:%d:\n", indent, "", jv_string_value(name), i);
    jv_free(name);
    dump_disassembly(indent+2, subfn);
  }
}
```

- La función no posee un valor de retorno ni instrucciones de retorno, por lo que se trata de un procedimiento.
- La signatura de la función (descartando valor de retorno) es:
  - Toma un entero sin signo llamado ‘indent’
  - Toma un puntero a una estructura de tipo ‘bytecode’
- Se trata de una función con n-aridad par; luego es binaria.
- Ambos parámetros son no constantes.
- No posee parametrización variable.
- No es predicado, ya que no devuelve valor alguno. Además, sabemos que es un procedimiento, luego implica que no es predicado.
- No es función de sistema, ya que como vemos, se trata de una implementación propia. Además, no está documentada en las APIs de Windows o Linux si buscásemos información.

- Los parámetros de la función no toman valores por defecto, luego por cada llamada que observemos de esta, los dos parámetros deberán contener valores apropiados.
- El parámetro 'indent' no es modificado, su valor solo es leído (es más, podría ser constante, es decir "const int indent")
- El parámetro 'bc' es un puntero a una estructura. Esto es común, dado que no se suele pasar una estructura como argumento para que esta sea copiada en cada llamada. Además, no se modifica 'bd' ni sus miembros de forma directa.

¿Es necesario rellenar TODO?

No, pero cuanta más información aportéis de su análisis mayor será la nota que consigáis

2.1. FUNCIÓN 1

```
int aeResizeSetSize(aeEventLoop *eventLoop, int setsize) {
    int i;

    if (setsize == eventLoop->setsize) return AE_OK;
    if (eventLoop->maxfd >= setsize) return AE_ERR;
    if (aeApiResize(eventLoop,setsize) == -1) return AE_ERR;

    eventLoop->events = zrealloc(eventLoop->events,sizeof(aeFileEvent)*setsize);
    eventLoop->fired = zrealloc(eventLoop->fired,sizeof(aeFiredEvent)*setsize);
    eventLoop->setsize = setsize;

    /* Make sure that if we created new slots, they are initialized with
     * an AE_NONE mask. */
    for (i = eventLoop->maxfd+1; i < setsize; i++)
        eventLoop->events[i].mask = AE_NONE;
    return AE_OK;
}
```

2.2. FUNCIÓN 2

```
static int
args_cmp(struct args_entry *a1, struct args_entry *a2)
{
        return (a1->flag - a2->flag);
}
```

2.3. FUNCIÓN 3

```
uint16_t checksum_generic(uint16_t *addr, uint32_t count)
{
    register unsigned long sum = 0;

    for (sum = 0; count > 1; count -= 2)
        sum += *addr++;
    if (count == 1)
        sum += (char)*addr;

    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += (sum >> 16);

    return ~sum;
}
```

2.4. FUNCIÓN 4

```
void rand_str(char *str, int len) // Generate random buffer (not alphanumeric!) of length len
{
    while (len > 0)
    {
        if (len >= 4)
        {
            *((uint32_t *)str) = rand_next();
            str += sizeof (uint32_t);
            len -= sizeof (uint32_t);
        }
        else if (len >= 2)
        {
            *((uint16_t *)str) = rand_next() & 0xFFFF;
            str += sizeof (uint16_t);
            len -= sizeof (uint16_t);
        }
        else
        {
            *str++ = rand_next() & 0xFF;
            len--;
        }
    }
}
```

2.5. FUNCIÓN 5

```
static const char *TextFormat(const char *text, ...)
{
    #define MAX_FORMATTEXT_LENGTH    64

    static char buffer[MAX_FORMATTEXT_LENGTH];

    va_list args;
    va_start(args, text);
    vsprintf(buffer, text, args);
    va_end(args);

    return buffer;
}
```

3. EJERCICIO

Una función representa un punto de entrada, posiblemente con parámetros, que puede necesitar acceder a variables globales, que puede manipular ciertas variables y que, finalmente, retorna o no un resultado. En más, puede retonar no solo en un punto sino en varios, dado que también puede tomar decisiones mediante las estructuras de control que hemos estudiado.

Tenemos el código de una función y vamos a analizarla. Necesitamos señalar:

- Donde y que variables crea y necesita; cual es su ámbito.
- Donde modifica (escribe) los parámetros.
- Donde lee o modifica valores globales (fuera de la función o sus parámetros)
- Que estructuras de control posee. Debemos identificar los bucles (while, for, …), estructuras selectivas (if, switch, …).
- Que puntos de salida posee y con que valor lo hacen.

Un ejemplo sencillo:

```
1   listNode *sentinelGetScriptListNodeByPid(pid_t pid) {
2       listNode *ln;
3       listIter li;
4
5       listRewind(sentinel.scripts_queue,&li);
6       while ((ln = listNext(&li)) != NULL) {
7           sentinelScriptJob *sj = ln->value;
8
9           if ((sj->flags & SENTINEL_SCRIPT_RUNNING) && sj->pid == pid)
10              return ln;
11      }
12      return NULL;
13  }
```

- Toma un solo parámetro del tipo “pid_t” llamado “pid”.

- Crea dos variables propias:
    - ln, que es un puntero a una estructura del tipo “listNode”
    - li, que es una estructura de tipo “listIter”

- Llama a la función “listRewind”:
    - Hace uso de un objeto externo o global: “sentinel”
    - Hace uso de una variable propia, “li”, la cual es pasada como puntero (su dirección) posiblemente para que sea modificada, dado que no devuelve ningún valor.

- Efectúa una iteración “while” desde la línea 6 a la línea 11. La condición de salida del bucle es que la variable “ln” sea NULL. Es decir, en cuanto “ln” valga NULL nos posicionaremos en la línea 12.

- Debemos anotar cuando se está comparando (línea 6), se llama a la función “listNext” con la dirección de la variable propia “ln”, lo cual indica que esta será leída, puesto que hacemos uso del valor devuelto.

- Dentro del bucle “while”, se crea una variable del tipo puntero a “sentinelScriptJob” llamada “sj” y se le asigna el valor de la variable “ln->value”.

- En la línea 9, dentro del bucle “while”, tenemos una condición “if” que decidirá si la función retorna en este punto con el valor que actualmente posea la variable local a la función “ln”. Dicha condición lógica es que la variable local al “if” posea los valores de la constante SENTINEL_SCRIPT_RUNNING y el del argumento “pid” en los miembros correspondientes de la estructura.

En resumen:

- (1) Argumento del tipo “pid_t”.
- (2, 3) Valores locales a la función: “puntero a listNode”, “listIter”
- (5) Llamada a función externa “listRewind”
- (6) Estructura de repetición o bucle “while”
- (7) Variable local a (6)
- (9) Condicional “if”:
    - Si es positiva, se retorna con el valor actual de “ln”
    - Si es negativa no se hace nada
- (12) Retorna null.

Ahora se pide hacer un estudio similar de la siguiente función:

```
1  void sentinelPropagateDownAfterPeriod(sentinelRedisInstance *master) {
2      dictIterator *di;
3      dictEntry *de;
4      int j;
5      dict *d[] = {master->slaves, master->sentinels, NULL};
6
7      for (j = 0; d[j]; j++) {
8          di = dictGetIterator(d[j]);
9          while((de = dictNext(di)) != NULL) {
10             sentinelRedisInstance *ri = dictGetVal(de);
11             ri->down_after_period = master->down_after_period;
12         }
13         dictReleaseIterator(di);
14     }
15 }
```

Código disponible en: https://pastebin.com/W0RXzeUd

4. EJERCICIO

Ahora vamos a aprender a construir una base de datos simbólica del código fuente utilizando algunas de las herramientas mostradas en el temario. Esto nos ayudará enormemente a comprender las relaciones entre funciones y objetos.

Este tipo de datos se emplea sobre todo en Bug Hunting de proyectos Open Source y análisis de vulnerabilidades, aunque en la práctica nos ayudará a navegar por todo el código fuente de cualquier proyecto.

La idea es tener claro que abordar un programa puede ser una tarea tediosa y deberemos apoyarnos en herramientas, pero siempre comprendiendo que estamos haciendo y por qué.

Este tipo de herramientas no son "mágicas" en el sentido de apretar un botón y que realicen el análisis y sus resultados por nosotros. Son meros apoyos con los que contamos para emprender la tarea de análisis.

En primer lugar, vamos a coger un proyecto Open Source cualquier, por ejemplo: REDIS. Una base de datos distribuida llave-valor bastante popular. Está escrita en C y su código disponible en Github.

* Clonamos el proyecto:

$ git clone https://github.com/redis/redis --depth=1

* Utilizad la o las herramientas del capítulo 6 que creáis oportunas para extraer los símbolos del código.

OJO, algunas instrucciones pueden no funcionar copiándolas directamente del PDF.

Por ejemplo (pag. 120), si lo copias literalmente contiene caracteres incompatibles con la terminal.

$ find . -name "*.[h,c]" > cscope.files

* Consultad los manuales de las herramientas o páginas man. No os baseis solo en la presentación que se hace de ellas en el temario, investigad sus funciones.

En este ejercicio se pide:

1.- Muestra las instrucciones (capturas de pantalla si quieres) de los pasos dados para crear la base de datos simbólica en la/las herramienta/s que hayas elegido.

2.- **Busca todas las funciones "main" que existen** en el proyecto. Como sabrás, son las funciones principales de entrada a los distintos ejecutables que existen. Haz una captura o copia la lista resultante.

3.- Enumera o pon una captura de pantalla de **todas las funciones que llaman a** la función "stringmatch".

4.- **Buscad la declaración de "struct stream" en el código**, tanto el archivo donde está, como la línea donde comienza. Debéis hacerlo usando las herramientas, no vale con mostrar su localización, debéis mostrar **como** se localiza.

5.- Por último, vamos a utilizar una funcionalidad extraordinaria cuando nos encontramos con una función extremadamente larga y difícil de analizar (suelen ser una pesadilla). Enumerad, mediante una lista o capturas de pantalla, **qué funciones llama la función 'genRedisInfoString'**.

5. EJERCICIO

Vamos a analizar las variables de un programa y donde se van a ubicar en memoria.

Respecto del siguiente programa, indicad **de que tipo son cada una de las variables, en que zona o segmento de la memoria se ubicará y que valor por defecto poseerá al inicio del programa**.

```c
#include <stdio.h>
#include <stdlib.h>

int gni;
int gi = 99;

void f() {
    static int eni;
    static int ei = 99;

    printf("Dir: %p Val: %i\n", &eni, eni);
    printf("Dir: %p Val: %i\n", &ei, ei);
}

int main() {
    f();
    int *pae = malloc(sizeof(int));
    printf("Dir: %p Val: %i\n", pae, *pae);

    *pae = 100;
    printf("Dir: %p Val: %i\n", pae, *pae);

    gni = 9;
    printf("Dir: %p Val: %i\n", &gni, gni);
    printf("Dir: %p Val: %i\n", &gi, gi);

    free(pae);
    return 0;
}
```

Por si queréis compilar el código o estudiarlo en un editor de código:

https://pastebin.com/6d4cqSZf

6. EJERCICIO

IMPORTANTE IMPORTANTE IMPORTANTE

NO USEIS VUESTRO SISTEMA

USAD LA MÁQUINA VIRTUAL

DESCONECTAD LA RED DE LA VIRTUAL

DESECHAD EL ESTADO DE LA VIRTUAL UNA VEZ FINALIZADO EL EJERCICIO

IMPORTANTE IMPORTANTE IMPORTANTE

Ahora viene un ejercicio que no solo emula, sino que te pone en la piel de un analista en cualquier SOC. Es su día a día, recibir muestras de malware o imágenes de disco en los que existen indicios de actividad maliciosa.

Han encontrado un script malicioso alojado en un servidor web de la compañía de un cliente. Es evidente la ofuscación e incluso codificación. Dado que se trata de un incidente grave, se necesita analizar este fragmento y extraer la funcionalidad que posee y los IOC (Indicadores de compromiso[1]) asociados (dominios, IP, etc) para crear reglas eficaces en los sistemas de defensa de la empresa.

- Deberéis de desofuscar el ejemplar.
- Analizad de forma estática (no lo ejecutéis bajo nigún concepto) su comportamiento y describid que hace o parece hacer.
- Extraer los IOCs si es que existe alguno.
- EXPLICAD PASO A PASO (si es con imágenes mejor) como lo hacéis.

No es necesario que finaliceis el ejercicio, hasta donde llegueis se os puntuará.

Se valora que demostréis que teneis claro lo que estáis haciendo con explicaciones detalladas.

Disponible aquí para que podáis trabajar con él en un editor de código:

[1] https://es.wikipedia.org/wiki/Indicador_de_compromiso

https://pastebin.com/jvCuaK2W

Ese fragmento está codificado y comprimido, luego tenéis que realizar las siguientes operaciones para obtener el original (vamos a suponer que lo habéis guardado en un archivo: ejercicio6.txt):

$ base64 -D -i ejercicio6.txt -o ejercicio.zip

$ unzip ejercicio.zip

La contraseña es: **master2021**

6.1.1. PISTAS

- Nada es lo que parece a simple vista
- Javascript es un lenguaje con un comportamiento…curioso
- A veces es mejor intentar cosas pequeñas, de forma aislada…
- Node es tu amigo
- Sustitución es mejor que fuerza bruta

IMPORTANTE IMPORTANTE IMPORTANTE

NO USEIS VUESTRO SISTEMA

USAD LA MÁQUINA VIRTUAL

DESCONECTAD LA RED DE LA VIRTUAL

DESECHAD EL ESTADO DE LA VIRTUAL UNA VEZ FINALIZADO EL EJERCICIO

IMPORTANTE IMPORTANTE IMPORTANTE