

# Malware Analysis

(ransomware “policía federal”)

6/6/2013

Autor: Alejandro Torres Ramírez (TorresCrack)

T: @torrescrack248

<http://Torrescrack.blogspot.com>

## Resumen

La muestra analizada es sobre un ransomware que usurpa la identidad de la policía federal. Ransomware es un tipo de malware (software malintencionado) que los criminales instalan en su PC sin su consentimiento y les da a los criminales la capacidad de bloquear su equipo, luego presentará una ventana emergente con un aviso que dice que su ordenador que se encuentra bloqueado y afirma que no podrá acceder al mismo a no ser que pague.

El análisis dinámico con el sandbox Anubis nos devuelve información incompleta ya que el malware hace uso de la API denominada "Sleep", la cual duerme durante algunos minutos lo cual altera el funcionamiento de Anubis, pues únicamente deja en ejecución el proceso durante poco tiempo y lo cierra por lo tanto no da oportunidad al malware de ejecutarse por completo y de capturar por completo sus funciones.

El malware analizado se copia en diferentes carpetas dentro del sistema, modifica un path para que se ejecute al iniciar Windows, copia y borra algunas llaves del registro encargadas del reinicio "modo seguro" impidiendo el mismo, guarda la copia del contenido real de las mismas con un nombre diferentes.

Utiliza únicamente la encriptación XOR para desencriptar todos o gran parte de los strings y el mismo método lo utiliza para desencriptar una gran parte del código que utiliza en diferentes capas.

Lanza el proceso "svchost.exe" en modo suspendido, el cual es un proceso del sistema que aloja varios servicios de Windows, crea una sección nueva dentro del proceso y coloca dentro de ella otra parte del código del malware, después sobrescribe el "entry point" o punto de entrada que es donde inicia la ejecución el proceso y coloca un salto directo a la sección creada y escrita por el malware para seguir con la ejecución del malware en el proceso inyectado.

Las conexiones cliente – servidor las realiza en dos servidores diferentes en el primer servidor de ucrania únicamente descarga la imagen .jpg que muestra y en el segundo servidor manda la información que se colocó en el espacio donde solicita la clave del pago que se debe realizar.

Se analizaron rápidamente los servidores, encontrando en uno de ellos información de la base de datos SQL en la cual se registraban las IP's de las víctimas, la información del pago y la fecha en que mando la Información.

## Tabla de Contenidos

|   |          |
|---|----------|
| <b>1. Introduccion .....</b>              | <b>4</b> |
| 1.1 Contexto .....                        | 4        |
| 1.2 Objetivo .....                        | 4        |
| <b>2. Información del binario.....</b>    | <b>5</b> |
| 2.1 Ejecutable encriptado .....           | 5        |
| 2.1.1 Hash del archivo.....               | 5        |
| 2.1.2 Tipo de Archivo .....               | 5        |
| 2.1.3 Informacion PE .....                | 5        |
| 2.2 Informacion Virus Total.....          | 6        |
| 2.3 Informacion Anubis .....              | 7        |
| <b>3. Herramientas.....</b>               | <b>8</b> |
| 3.1 Preparando el escenario.....          | 8        |
| 3.2 Escaneando el binario .....           | 8        |
| <b>4. Desempacando.....</b>               | <b>9</b> |
| 4.1 Primer capa.....                      | 9        |
| 4.1.1 Guardando Datos Encriptados.....    | 10       |
| 4.1.2 Rutina Desencriptacion .....        | 13       |
| 4.1.3 Guardando Datos Encriptados.....    | 14       |
| 4.1.4 Rutina Desencriptacion .....        | 15       |
| 4.1.5 Copiando Datos Desencriptados ..... | 16       |
| 4.2 Segunda capa .....                    | 17       |
| 4.2.1 Obteniendo direcciones API's.....   | 18       |
| 4.2.2 Guardando Datos Encriptados.....    | 19       |
| 4.2.3 Rutina Desencriptacion .....        | 22       |
| 4.2.4 Copiando datos desencriptados ..... | 23       |
| 4.3 Tercer capa .....                     | 25       |

|  |           |
|--|-----------|
| 4.3.1 Modificación en el registro de Windows .....             | 26        |
| 4.3.2 Rutina Descriptacion .....                               | 27        |
| 4.3.3 Creacion del proceso suspendido .....                    | 29        |
| 4.3.4 Deteccion del Sistema Operativo .....                    | 30        |
| 4.3.5 Inyectando en el proceso suspendido .....                | 32        |
| 4.3.6 Modificando el entrypoint .....                          | 33        |
| <b>5 Proceso inyectado .....</b>                               | <b>34</b> |
| 5.1 Modificacion en el registro de windows .....               | 35        |
| 5.2 Copiando y eliminando Ficheros .....                       | 36        |
| 5.3 Copiando y eliminando llaves del registro de windows ..... | 36        |
| 5.4 Descriptando datos de conexión la red .....                | 38        |
| 5.5 Estableciendo controles de la ventana de dialogo.....      | 38        |
| 5.6 Monitor y cierre de procesos en el sistema.....            | 40        |
| <b>6 Palabras finales.....</b>                                 | <b>41</b> |
| <b>7 Agradecimientos .....</b>                                 | <b>41</b> |
| <b>8 Contacto .....</b>  | <b>41</b> |

## 1. Introduccion

**POLICÍA FEDERAL**

Todas las operaciones que se realizan en este ordenador se están grabando. Si Usted usa una cámara-web, entonces los materiales de video y foto se guardan para realizar la identificación.

Video-Grabación: SI Audio-Grabación: SI

**Su ordenador se ha bloqueado!**

El funcionamiento de su ordenador fue interrumpido a causa de índices de ciber actividad desautorizada.

Los delitos posibles cometidos por ud se indican abajo:

Cláusula 274 - Derechos de autor  
Multa o privación de libertad de hasta 4 años, (utilización o distribución de los ficheros protegidos con derechos de autor - películas, software)

Cláusula 183 - Productos pornográficos  
Multa o o privación de libertad de hasta 2 años, (utilización o distribución de los ficheros pornográficos)

Cláusula 184 - Productos pornográficos con participación de los menores (menos de 18 años)  
Multa o privación de libertad de hasta 15 años, (utilización o distribución de los ficheros pornográficos)

Cláusula 104 - Popularización del Terrorismo  
Privación de libertad de hasta 25 años, (Ud ha entrado las páginas web de organizaciones terroristas)

Cláusula 297 - Uso despectivo del ordenador que causó las consecuencias graves  
Multa o privación de libertad de hasta 2 años, (Su ordenador fue infectado por un virus que a su vez infectó otros ordenadores)

Cláusula 108 - juegos de azar  
Multa o privación de libertad de hasta 2 años, (Ud ha jugado los juegos de azar, aunque según la ley de su país el negocio de juegos de envite y azar está prohibido)

A la vista de Resolución del Gobierno de 22 de agosto se puede considerar todos estos delitos como condicionales con la multa pagada.

La suma de multa es **1000 MXN**. El pago debe ser realizado durante 48 horas a partir de revelación de delito.

Si la multa no es pagada, será iniciada la causa penal contra ud.

Una vez esté pagada la multa su ordenador será desbloqueado

Para desbloquear su ordenador y evitar la persecución judicial Usted tiene que hacer el pago por el valor de **1000 MXN**.

1. 2. 3. 4.

**Ukash**

Usted puede obtener Ukash en cientos de miles de lugares del mundo, en línea, de carteras, en quioscos y cajeros automáticos.

¿Dónde puedo comprar Ukash

**farmacias**  
**ya**  
**PayWin**

Cambie el dinero efectivo al cupón Ukash e introduzca el código del cupón al formulario indicado a continuación.

**paysafecard**

En México puedes obtener paysafecard en las tiendas de 7Eleven, extra®, Ley, Comercial mexicana, Blockbuster, Sanborns, Superette, Fasti, Del Rio, Soriana y en selectos puntos de OXXO.

¿Dónde puedo comprar Paysafecard

**fasti**  
**oasis**  
**extra**  
**OXXO**  
**OfficeMax**

Cambie el dinero efectivo al cupón Paysafecard e introduzca el código del cupón al formulario indicado a continuación.

Por favor, preste atención al hecho de que la multa debe ser pagada durante 48 horas. Si Usted no tendrá la posibilidad de hacer el pago durante el plazo indicado, no será posible desbloquear su ordenador.

En este caso se iniciará la causa penal contra Usted automáticamente.

100% Pagos Seguros

### 1.1 Contexto

Este documento fue creado para dar un ejemplo de un análisis exhaustivo de malware. El documento puede ser compartido a todos los interesados en el mismo.

### 1.2 Objetivo

El objetivo es hacer un análisis completo de un “ransomware” que usurpa la identidad de la policía federal y entender cómo funciona el malware, controlarlo y si es posible recuperar los daños causados por el mismo.

## 2. Información del binario

### 2.1 Ejecutable encriptado

#### 2.1.1 Hash del archivo

SHA256: 0975e128554064c484237aba089153e0fb0200e01a346ea8fe28026c1354f41a  
SHA1: 49627b699a37864375ff44098a879e6b71f690e5  
MD5: bf26ea9dc4af592c658b5af66c14c11d

#### 2.1.2 Tipo de Archivo

Win32 EXE executable PE for MS Windows (GUI) Intel 80386 32-bit

#### 2.1.3 Informacion PE

Fecha de compilación: 2013-03-12 14:24:35

EntryPoint: 0x00009590

Secciones:

| Name    | Virtual address | Virtual size | Raw size | Entropy | MD5                              |
|---------|-----------------|--------------|----------|---------|----------------------------------|
| .textRR | 4096            | 591          | 1024     | 3.38    | 57b2ada72a312cd3d1820c15965af89f |
| .text   | 8192            | 35166        | 35328    | 7.11    | 86a7122622ffd7d15502ef964d19a7b6 |
| .data   | 45056           | 100          | 512      | 0.87    | 5501f3f560b3455b778fcbdda0df04c6 |
| .rdata3 | 49152           | 100          | 512      | 0.00    | bf619eac0cdf3f68d496ea9344137e8b |
| .rdata2 | 53248           | 100          | 512      | 0.00    | bf619eac0cdf3f68d496ea9344137e8b |

## 2.2 Informacion Virus Total

| Antivirus            | Resultado                        |
|----------------------|----------------------------------|
| AhnLab-V3            | Trojan/Win32.Blocker             |
| AVG                  | Generic32.FYN 20130320           |
| BitDefender          | Trojan.Generic.KDZ.10799         |
| DrWeb                | Trojan.MulDrop4.26873            |
| ESET-NOD32           | Win32/LockScreen.AQT             |
| F-Secure             | Trojan.Generic.KDZ.10799         |
| GData                | Trojan.Generic.KDZ.10799         |
| Kaspersky            | Trojan-Ransom.Win32.Blocker.avzn |
| Kingsoft             | Win32.Troj.Undef.(kcloud)        |
| Malwarebytes         | Trojan.Ransom                    |
| McAfee               | PWS-Zbot-FAKU!BF26EA9DC4AF       |
| McAfee-GW-Edition    | PWS-Zbot-FAKU!BF26EA9DC4AF       |
| MicroWorld-eScan     | Trojan.Generic.KDZ.10799         |
| nProtect             | Trojan/W32.Blocker.47896         |
| Panda                | Trj/Dtcontx.C                    |
| PCTools              | HeurEngine.MaliciousPacker       |
| Symantec             | Packed.Generic.401               |
| TrendMicro           | Mal_Ransom-1                     |
| TrendMicro-HouseCall | Mal_Ransom-1                     |
| VBA32                | Hoax.Blocker.auxi                |
| VIPRE                | Trojan.Win32.Reveton.a (v)       |
| ViRobot              | Trojan.Win32.Ransom.47896        |

## 2.3 Informacion Anubis

### - Registry Values Modified:

| Key   | Name | New Value         |
|---|------|-------------------|
| HKU\S-1-5-21-842925246-1425521274-308236825-00\SOFTWARE\Microsoft\Windows\Current version | DNS  | C:\MigAutoPla.exe |

MigAutoPlay.exe - File Activities

### - File System Control Communication:

| File                           | Control Code | Times |
|--------------------------------|--------------|-------|
| C:\Program Files\Common Files\ | 0x00090028   | 1     |

### - Processes Created:

| Executable                      | Command Line |
|---------------------------------|--------------|
| C:\WINDOWS\system32\svchost.exe |              |

svchost.exe - Registry Activities

### - Registry Keys Created:

|   |
|---|
| HKLM\System\CurrentControlSet\Control\SafeBoot\mini |
|---|

### Registry Keys Deleted:

|  |
|--|
| HKLM\System\CurrentControlSet\Control\SafeBoot\Minimal |
|--|

### - Files Deleted:

|                    |
|--------------------|
| C:\MigAutoPlay.exe |
|--------------------|

### - Files Created:

|  |
|--|
| C:\Documents and Settings\All Users\Application Data\MigAutoPlay.exe |
|--|

### - Files Read:

|  |
|--|
| C:\Documents and Settings\All Users\Application Data\MigAutoPlay.exe |
|--|



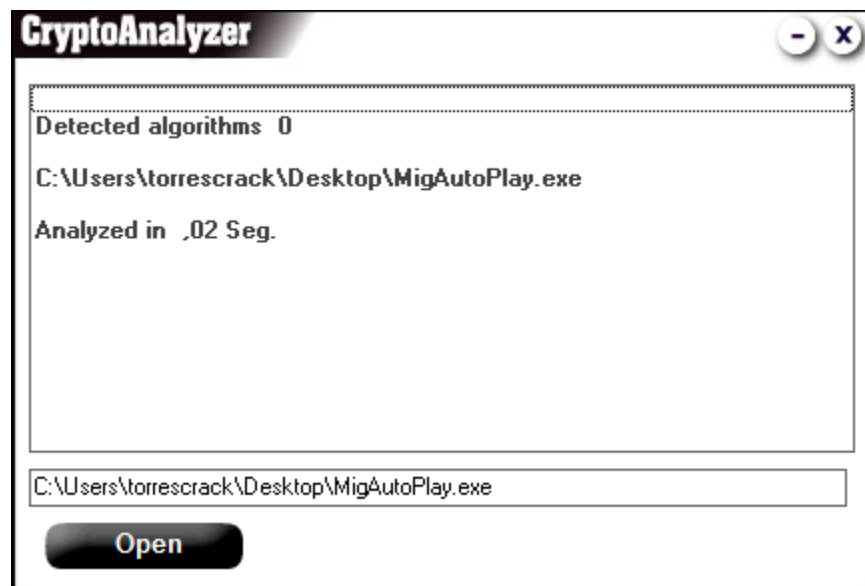
## 3. Herramientas

### 3.1 Preparando el escenario

las herramientas utilizadas para este análisis:

- Olly debugger
- VMware – Windows XP (x86)
- RDG Packer Detector

### 3.2 Escaneando el binario



No detecto ningún algoritmo de encriptación pero no hay que descartar que pudiera ocupar algún método que no fue detectado, pero este usa el tipo de encriptación/descriptación XOR para strings y algo de código.

## 4. Desempacando

### 4.1 Primer capa

Al cargarlo con el debugger podemos ir viendo un poco más detallado todo y como va cargando las cosas, bien seguimos.

|      |                 |  |   |
|------|-----------------|--|---|
| 9590 | SS              | PUSH EBP                                       |   |
| 9591 | . 8BEC          | MOV EBP,ESP                                    |   |
| 9593 | . 81EC 94090000 | SUB ESP,0x994                                  |   |
| 9599 | . C785 0CF7FFF  | MOV DWORD PTR SS:[EBP-0x8F4],MigAutoP.00400000 |   |
| 95A3 | . C785 A8F6FFF  | MOV DWORD PTR SS:[EBP-0x958],0x4               |   |
| 95AD | . C785 ACF6FFF  | MOV DWORD PTR SS:[EBP-0x954],0x20019           |   |
| 95B7 | . C785 28F7FFF  | MOV DWORD PTR SS:[EBP-0x8D8],MigAutoP.0040B000 | ASCII "22ftware\classes"                    |
| 95C1 | . C785 2CF7FFF  | MOV DWORD PTR SS:[EBP-0x8D4],MigAutoP.0040B014 | ASCII "htmlDlghelper.htmlDlghelper.1\clsid" |
| 95CB | . C685 48F7FFF  | MOV BYTE PTR SS:[EBP-0x8B8],0x39               |   |
| 95D2 | . C785 30F7FFF  | MOV DWORD PTR SS:[EBP-0x8D0],0x0               |   |
| 95DC | . C685 48F7FFF  | MOV BYTE PTR SS:[EBP-0x8B8],0x39               |   |
| 95E3 | . C785 30F7FFF  | MOV DWORD PTR SS:[EBP-0x8D0],0x0               |   |
| 95ED | . C685 48F7FFF  | MOV BYTE PTR SS:[EBP-0x8B8],0x39               |   |
| 95F4 | . C785 30F7FFF  | MOV DWORD PTR SS:[EBP-0x8D0],0x0               |   |
| 95FE | . C685 48F7FFF  | MOV BYTE PTR SS:[EBP-0x8B8],0x39               |   |
| 9605 | . C785 30F7FFF  | MOV DWORD PTR SS:[EBP-0x8D0],0x0               |   |
| 960F | . C685 48F7FFF  | MOV BYTE PTR SS:[EBP-0x8B8],0x39               |   |
| 9616 | . C785 30F7FFF  | MOV DWORD PTR SS:[EBP-0x8D0],0x0               |   |

Veamos las strings a ver si hay algo que nos pueda ayudar, nombres de archivos, carpetas o algo que ocupe para hacer destrozos en el sistema veamos:

```

040246C ASCII "6gA)S"
0402478 ASCII "1)A)Sa",0
040247F ASCII "%QI",0
0402493 ASCII "QI",0
0402497 ASCII "QI",0
0402498 ASCII "PI",0
040249F ASCII "PI",0
04024A2 ASCII "JUI",0
04024A7 ASCII "UI",0
04024AB ASCII "UI",0
04024AF ASCII "UI",0
04024B3 ASCII "UI",0
04024B7 ASCII "UI",0
04024BB ASCII "UI",0
04024BF ASCII "UI",0
04024C7 ASCII "_)",0
04024CA ASCII "#Sa",0
04024CE ASCII "f0)P;:jAE/(e:&a",0
0402505 ASCII "0/el)Sa",0
0402516 ASCII "/el)Sa",0
040251D ASCII ")Sa",0
0402521 ASCII ")Sa",0
0402525 ASCII ")Sa",0
0402529 ASCII ")S]",0
040253D ASCII ")d",0
0402545 ASCII ")!)",0
0402618 ASCII "%S",0
04026D1 ASCII "*Sa",0
04026D9 ASCII "DMa",0
04026DD ASCII "PSa",0
04026E1 ASCII "PSa",0
04026E5 ASCII "@Sa",0
04026E9 ASCII "0Sa",0
04026EE ASCII "Sa",0

```

No hay nada, únicamente strings encriptadas, que seguramente deben desencriptarse al momento de utilizarlas, también desde aquí puedo deducir que cargara otra sección de memoria y parte de todo esto es código lo colocara ahí dentro.

## 4.1.1 Guardando Datos Encriptados

Seguimos viendo directo a las zonas importantes para no alargar demasiado intentado explicar las vueltas que da para llegar a cargar una API, seguimos:

|         |      |   |                                       |
|---------|------|---|---------------------------------------|
| 24F7FFF | MOV  | DWORD PTR SS:[EBP-0x8DC],EAX                |                                       |
| 9820400 | MOV  | ECX,DWORD PTR DS:[<&KERNEL32.VirtualAlloc>] | kernel32.VirtualAlloc                 |
| 50B0400 | MOV  | DWORD PTR DS:[0x40B050],ECX                 | kernel32.VirtualAlloc                 |
| 24F7FFF | MOV  | EDX,DWORD PTR SS:[EBP-0x8DC]                |                                       |
|         |      | PUSH EDX                                    |                                       |
| 040000  | CALL | MigAutoP.00409C70                           | [Arg1 = 00000002<br>MigAutoP.00409C70 |
| 040000  |      | ADD ESP,4                                   |                                       |

# WHITE PAPER

Quiero mostrarles como guarda el offset de *VirtualAlloc* en el buffer y después entra a un call en la dirección 0x409C70 :

```
PUSH ECX
MOV EDX, DWORD PTR SS:[EBP-0x10]
PUSH EDX
MOV EAX, DWORD PTR SS:[EBP-0xC]
PUSH EAX
MOV ECX, DWORD PTR SS:[EBP-0x8]
PUSH ECX
CALL DWORD PTR DS:[0x40B050]
MOV ESP, EBP
POP EBP
RET
```

|          |          |   |
|----------|----------|---|
| 0012F5C8 | 00000000 | Address = NULL                          |
| 0012F5CC | 00006300 | Size = 6300 (25344.)                    |
| 0012F5D0 | 00003000 | AllocationType = MEM_COMMIT MEM_RESERVE |
| 0012F5D4 | 00000040 | Protect = PAGE_EXECUTE_READWRITE        |
| 0012F5D8 | 00003000 |   |
| 0012F5DC | 00000000 |   |

```
EAX 00300000
ECX 0012F580
EDX 775164F4
EBX 7FFDF000
```

Llama a la función *VirtualAlloc* que reserva o asigna una región de páginas del espacio de direcciones virtual del proceso que la invoca.

Su sintaxis es:

## Syntax

*LPVOID VirtualAlloc(*

*LPVOID lpAddress, // dirección de la región a reservar o asignar*

*DWORD dwSize, // tamaño de la región*

*DWORD flAllocationType, // tipo de petición*

*DWORD flProtect // tipo de protección de acceso*

*);*

Crea una sección de memoria vacía (en la dirección 0x300000) en la cual más adelante meterá algunos datos encriptados, seguimos y al avanzar vemos esta rutina:

# WHITE PAPER

|          |                 |                                   |                         |
|----------|-----------------|-----------------------------------|-------------------------|
| 00409869 | > 8B4D F8       | MOV ECX, DWORD PTR SS:[EBP-0x8]   |                         |
| 0040986C | . 3B8D 24F7FF   | CMP ECX, DWORD PTR SS:[EBP-0x8DC] |                         |
| 00409872 | . 73 7D         | JNB SHORT MigAutoP.004098F1       |                         |
| 00409874 | . 8B95 C8F6FF   | MOV EDX, DWORD PTR SS:[EBP-0x938] |                         |
| 0040987A | . 52            | PUSH EDX                          | MigAutoP.0040241F       |
| 0040987B | . 8B85 B8F6FF   | MOV EAX, DWORD PTR SS:[EBP-0x948] |                         |
| 00409881 | . 50            | PUSH EAX                          |                         |
| 00409882 | . E8 99030000   | CALL MigAutoP.00409C20            |                         |
| 00409887 | . 83C4 08       | ADD ESP, 0x8                      |                         |
| 0040988A | . 8985 C0F6FF   | MOV DWORD PTR SS:[EBP-0x940], EAX |                         |
| 00409890 | . 8B8D C0F6FF   | MOV ECX, DWORD PTR SS:[EBP-0x940] |                         |
| 00409896 | . 51            | PUSH ECX                          | n = 64 (100.)           |
| 00409897 | . 8B15 58B04000 | MOV EDX, DWORD PTR DS:[0x40B058]  | MigAutoP.004023AC       |
| 0040989D | . 0395 14F7FF   | ADD EDX, DWORD PTR SS:[EBP-0x8EC] | src = MigAutoP.0040241F |
| 004098A3 | . 52            | PUSH EDX                          |                         |
| 004098A4 | . 8B85 B4F6FF   | MOV EAX, DWORD PTR SS:[EBP-0x94C] |                         |
| 004098AA | . 0345 F8       | ADD EAX, DWORD PTR SS:[EBP-0x8]   | dest = 00300064         |
| 004098AD | . 50            | PUSH EAX                          | memcpy                  |
| 004098AE | . E8 AF060000   | CALL <JMP.&msvcrt.memcpy>         |                         |
| 004098B3 | . 83C4 0C       | ADD ESP, 0xC                      |                         |
| 004098B6 | . 8B8D 1CF7FF   | MOV ECX, DWORD PTR SS:[EBP-0x8E4] |                         |
| 004098BC | . 038D B8F6FF   | ADD ECX, DWORD PTR SS:[EBP-0x948] |                         |
| 004098C2 | . 038D 14F7FF   | ADD ECX, DWORD PTR SS:[EBP-0x8EC] |                         |
| 004098C8 | . 898D 14F7FF   | MOV DWORD PTR SS:[EBP-0x8EC], ECX |                         |
| 004098CE | . 8B55 F8       | MOV EDX, DWORD PTR SS:[EBP-0x8]   |                         |
| 004098D1 | . 0395 B8F6FF   | ADD EDX, DWORD PTR SS:[EBP-0x948] |                         |
| 004098D7 | . 8955 F8       | MOV DWORD PTR SS:[EBP-0x8], EDX   | MigAutoP.0040241F       |
| 004098DA | . 8B85 C8F6FF   | MOV EAX, DWORD PTR SS:[EBP-0x938] |                         |
| 004098E0 | . 2B85 C0F6FF   | SUB EAX, DWORD PTR SS:[EBP-0x940] |                         |
| 004098E6 | . 8985 C8F6FF   | MOV DWORD PTR SS:[EBP-0x938], EAX |                         |
| 004098EC | . ^ E9 78FFFFFF | JMP MigAutoP.00409869             |                         |
| 004098F1 | > 8B8D 24F7FF   | MOV ECX, DWORD PTR SS:[EBP-0x8DC] |                         |

|          |          |                         |
|----------|----------|-------------------------|
| 0012F5E8 | 00300064 | dest = 00300064         |
| 0012F5EC | 0040241F | src = MigAutoP.0040241F |
| 0012F5F0 | 00000064 | n = 64 (100.)           |
| 0012F5F4 | 0006F7FF |                         |
| 0012F5F8 | 00010000 |                         |

Lo que hace aquí en esta rutina en sencillo, en el offset 0x40986C compara ecx con el tamaño de la sección que creo anteriormente que era de 6300h y de ser igual o mayor te saca de la rutina y si no pasa a copiar el contenido de 0x40241f con ayuda de Memcpy .  
**Copia los primeros n caracteres del objeto apuntado por s2 al objeto apuntado por s1.**

Seguimos:

## 4.1.2 Rutina de Descriptación

```

00 MOV ECX,0x1
TEST ECX,ECX
0001 JE MigAutoP.00409C14
FFFF MOV DWORD PTR SS:[EBP-0x214],0x0755
4001 MOV EDX,DWORD PTR DS:[0x40B054]
CMP EDX,DWORD PTR SS:[EBP+0xC]
00 JB SHORT MigAutoP.00409B77
00 JMP MigAutoP.00409C14
FFFF MOV DWORD PTR SS:[EBP-0x214],0x0755
FFFF MOV DWORD PTR SS:[EBP-0x214],0x0755
FFFF MOV DWORD PTR SS:[EBP-0x214],0x0755
FFFF MOV DWORD PTR SS:[EBP-0x214],0x0755
FFFF MOV DWORD PTR SS:[EBP-0x214],0x0755
FFFF MOV DWORD PTR SS:[EBP-0x214],0x0755
FFFF MOV DWORD PTR SS:[EBP-0x214],0x0755
MOV EAX,DWORD PTR SS:[EBP+0x8]
4001 ADD EAX,DWORD PTR DS:[0x40B054]
MOV ECX,DWORD PTR DS:[EAX]
4001 ADD ECX,DWORD PTR DS:[0x40B054]
MOV EDX,DWORD PTR SS:[EBP+0x8]
4001 ADD EDX,DWORD PTR DS:[0x40B054]
MOV DWORD PTR DS:[EDX],ECX
00 MOV EAX,DWORD PTR DS:[0x40B054]
00 ADD EAX,0x95329
PUSH EAX
MOV ECX,DWORD PTR SS:[EBP+0x8]
4001 ADD ECX,DWORD PTR DS:[0x40B054]
PUSH ECX
00 CALL MigAutoP.00409CB0
ADD ESP,0x8
4001 MOV EDX,DWORD PTR DS:[0x40B054]
ADD EDX,0x4
4001 MOV DWORD PTR DS:[0x40B054],EDX
FF JMP MigAutoP.00409B50
MOV ESP,EBP
POP EBP
RETN
INT3

```

|         |                                 |
|---------|---------------------------------|
| 55      | PUSH EBP                        |
| 8BEC    | MOV EBP,ESP                     |
| 83EC 18 | SUB ESP,0x18                    |
| 8B45 08 | MOV EAX,DWORD PTR SS:[EBP+0x8]  |
| 8945 EC | MOV DWORD PTR SS:[EBP-0x14],EAX |
| 8B4D 0C | MOV ECX,DWORD PTR SS:[EBP+0xC]  |
| 894D FC | MOV DWORD PTR SS:[EBP-0x4],ECX  |
| 8B55 EC | MOV EDX,DWORD PTR SS:[EBP-0x14] |
| 8955 F8 | MOV DWORD PTR SS:[EBP-0x8],EDX  |
| 8B45 FC | MOV EAX,DWORD PTR SS:[EBP-0x4]  |
| 8945 F4 | MOV DWORD PTR SS:[EBP-0xC],EAX  |
| 8B4D F8 | MOV ECX,DWORD PTR SS:[EBP-0x8]  |
| 894D F0 | MOV DWORD PTR SS:[EBP-0x10],ECX |
| 8B55 F4 | MOV EDX,DWORD PTR SS:[EBP-0xC]  |
| 8955 E8 | MOV DWORD PTR SS:[EBP-0x18],EDX |
| 8B45 F0 | MOV EAX,DWORD PTR SS:[EBP-0x10] |
| 8B08    | MOV ECX,DWORD PTR DS:[EAX]      |
| 334D E8 | XOR ECX,DWORD PTR SS:[EBP-0x18] |
| 8B55 F0 | MOV EDX,DWORD PTR SS:[EBP-0x10] |
| 890A    | MOV DWORD PTR DS:[EDX],ECX      |
| 8BE5    | MOV ESP,EBP                     |
| 5D      | POP EBP                         |
| C3      | RETN                            |
| CC      | INT3                            |

Esta rutina (imagen de arriba) hace lo mismo que la anterior un solo que esta va sacando los datos encriptados que guardo anteriormente, va colocando los primeros 4 bytes en ECX (en la imagen de arriba está señalada en gris esa instrucción) y hace un XOR con 95329h y los vuelve a colocar en la

# WHITE PAPER

misma dirección donde se encontraban pero ahora ya descriptados, para encriptar o descriptar el código solo basta usar el mismo **XOR** (bytes,95329h) ya que vimos cómo funciona y que método usaba para encriptar/descriptar los datos que copio anteriormente, coloquemos un BreakPoint al final de la rutina y veamos lo que descripto:

| Antes   | Después   |
|---|---|
| <pre> )6I.f6)P;&lt;jAE7{e :..W:(tL2eA.?fc )S..7&lt;dā-♦eo+of .l.i9(t,leP;Je #%..!;hd39kr# pE #4..p5)Mq7!lā+hn i?IA)S..j!lau60i ā6H.Z6)FP?lPN:gt L..v!tLS*IDS. j?fsD+hn.?l.p5)T .&gt;yPQ*aA)S... Jr #6gA)S..'▼)r#1)A )S..%QI.#QI.#QI. *PI.8PI.)UI.āUI. 0UI.8UI.8UI.0UI. +UI.8UI.ā.Ut+.). #S..f0)P;:jAE/ce :%.!;hd3;kr#pE #11..00(n.7:20/el )S..00(n)f6:2#/el )S..)S..)S..)S.. )C7..f6)P;:jAE/ce </pre> | <pre> .e@.GetProcAddress ss..VirtualAlloc ....UnmapViewOfFile ile.VirtualProtect ct..LoadLibraryEx xA..GetModuleHandleA...CreateFileA leA.SetFilePointer..WriteFile... CloseHandle.GetTempPathA...lstrlenA...lstrcpyA ....♦00.000.000. 400.D00.T00.h00. t00.ā00.ē00.ē00. %00.000.\_t34t. ....GetProcAddress ss..LoadLibraryEx xA..kernel32.dll ....kernel32.dll ..... ...T..8VE.00.00. </pre> |

Antes

Después

Encontramos algo interesante, son nombres de algunas *APIS* lo cual es muy probable las utilizara en conjunto con *GetProcAddress* para sacar las direcciones de las *APIS* e ir las cargando para poder utilizarlas más adelante así también como código que aún no sabemos para que lo ocupara, seguimos y encontramos esto:

## 4.1.3 Guardando Datos Encriptados

|          |          |   |
|----------|----------|---|
| 0012F5C8 | 00000000 | Address = NULL                          |
| 0012F5CC | 00000200 | Size = 200 (512.)                       |
| 0012F5D0 | 00003000 | AllocationType = MEM_COMMIT MEM_RESERVE |
| 0012F5D4 | 00000040 | Protect = PAGE_EXECUTE_READWRITE        |
| 0012F5D8 | 00003000 |   |
| 0012F5DC | 00000200 |   |

| Registers (FPU) |          |
|-----------------|----------|
| EAX             | 003F0000 |
| ECX             | 0012F580 |

Aquí crea una zona vacía en 0x3F0000 con tamaño de 200h para posteriormente llenarla como vimos anteriormente donde guardo los strings encriptados, seguimos más adelante para ver qué datos guardara en esta sección:

# WHITE PAPER

|                                  |                         |
|----------------------------------|-------------------------|
| PUSH EAX                         |                         |
| MOV ECX,DWORD PTR DS:[0x40B058]  | n = 64 (100.)           |
| ADD ECX,DWORD PTR SS:[EBP-0x8EC] | MigAutoP.00401004       |
| PUSH ECX                         |                         |
| MOV EDX,DWORD PTR SS:[EBP-0x8E8] | src = MigAutoP.00401004 |
| ADD EDX,DWORD PTR SS:[EBP-0x8]   |                         |
| PUSH EDX                         |                         |
| CALL <JMP.&msvcrt.memcpy>        | dest = 003F0000         |
| ADD ESP,0xC                      | memcpy                  |

|          |          |                         |
|----------|----------|-------------------------|
| 0012F5E8 | 003F0000 | dest = 003F0000         |
| 0012F5EC | 00401004 | src = MigAutoP.00401004 |
| 0012F5F0 | 00000064 | n = 64 (100.)           |
| 0012F5F4 | 0006F7FF |                         |
| 0012F5F8 | 00010000 |                         |

Estamos en la zona donde va a rellenar la sección vacía que creo anteriormente, pero veamos qué datos guardara seguimos avanzando y vemos esto:

| Hex dump  | ASCII               |
|---|---------------------|
| 00 02 00 00 29 53 09 00 05 53 09 00 ED 61 C1 32 | .0..)S..#S..ŷa+2    |
| ED 61 09 32 00 61 01 32 CD 61 E9 32 80 61 E1 32 | ŷa+2!a02=a02ia02    |
| 80 61 F9 32 90 61 09 00 29 03 09 00 09 53 09 00 | ia+20a..)0...S..    |
| 15 68 16 38 40 68 20 38 3E 68 50 38 06 68 73 38 | Sk..8Mk 8>kP8+ks8   |
| B3 68 90 38 8F 68 B9 38 93 68 31 39 A2 6A A9 39 | k08Ak  80k190j09    |
| 93 6A A6 39 71 6A B7 39 4A 6A E7 39 21 6A 1D 3A | 0j09qjA9Uj09!j#:    |
| 00 68 CD 38 61 6F 6D 3C 02 02 02 02 02 02 02 02 | !h=;aom<00000000    |
| 02 02 02 02 02 02 02 15 6F F4 3C DE 6D C8 3E 85 | 00000000So0<im0>a   |
| 6C 65 3F 7A 6C 70 3F 95 68 90 3F 97 68 BA 3F A0 | le?zlp?0ke?0k  ?a   |
| 68 C6 3F FD 68 F9 3F 29 33 09 00 39 53 09 00 24 | k0?2k-?)0...9S...\$ |
| 63 1E 30 6E 63 55 30 45 63 63 30 1E 63 74 30 A5 | cA0ncU0Ecc0A0ct00   |

## 4.1.4 Rutina de Descriptación

Igual que anteriormente guarda algunos datos que al parecer están encriptados, si seguimos podemos encontrar la rutina que descriptará y podremos ver el contenido real

|         |                                 |
|---------|---------------------------------|
| 8B55 F4 | MOV EDX,DWORD PTR SS:[EBP-0x8C] |
| 8955 E8 | MOV DWORD PTR SS:[EBP-0x18],EDX |
| 8B45 F0 | MOV EAX,DWORD PTR SS:[EBP-0x10] |
| 8B08    | MOV ECX,DWORD PTR DS:[EAX]      |
| 334D E8 | XOR ECX,DWORD PTR SS:[EBP-0x18] |
| 8B55 F0 | MOV EDX,DWORD PTR SS:[EBP-0x10] |
| 890A    | MOV DWORD PTR DS:[EDX],ECX      |
| 8BE5    | MOV ESP,EBP                     |
| 5D      | POP EBP                         |
| C3      | RET                             |

Usa la misma rutina que anteriormente uso para descriptar los datos anteriores, es la misma forma de encriptación/descriptación **XOR**, seguimos y al finalizar la rutina y ver el resultado de las operaciones vemos esto:



WHITE PAPER

WHITE PAPER

WHITE PAPER

WHITE PAPER

WHITE PAPER

WHITE PAPER

WHITE PAPER

WHITE PAPER

WHITE PAPER

# WHITE PAPER

|          |          |                   |
|----------|----------|-------------------|
| 0012F5E8 | 01200200 | dest = 01200200   |
| 0012F5EC | 00300000 | src = 00300000    |
| 0012F5F0 | 00006300 | n = 6300 (25344.) |
| 0012F5F4 | 0006F7FF |                   |
| 0012F5F8 | 00010000 |                   |
| 0012F5FC | 00010000 |                   |
| 0012F600 | 7FFEFFFF |                   |
| 0012F604 | 00000001 |                   |

| Address  | Hex dump  | ASCII              |
|----------|---|--------------------|
| 00300000 | 00 00 3F 00 47 65 74 50 72 6F 63 41 64 64 72 65 | ..?.GetProcAddress |
| 00300010 | 73 73 00 00 56 69 72 74 75 61 6C 41 6C 6C 6F 63 | ss..VirtualAlloc   |
| 00300020 | 00 00 00 00 55 6E 6D 61 70 56 69 65 77 4F 66 46 | ...UnmapViewOfF    |
| 00300030 | 69 6C 65 00 56 69 72 74 75 61 6C 50 72 6F 74 65 | ile.VirtualProte   |
| 00300040 | 63 74 00 00 4C 6F 61 64 4C 69 62 72 61 72 79 45 | ct..LoadLibraryE   |
| 00300050 | 78 41 00 00 47 65 74 4D 6F 64 75 6C 65 48 61 6E | xA..GetModuleHan   |

Ahora con *Mempcy* (imagen arriba) rellena la zona que acaba de crear y coloca dentro del zonavacia+200 mete los datos que descripto que eran los nombres de las API's y parte de código que encontramos anteriormente.

Seguimos y un poco más adelante vemos un RET, si lo cruzamos nos encontramos con esta zona algo interesante y es la que nos llevara a la zona con parte del código que descripto anteriormente:

|          |                 |                             |                   |
|----------|-----------------|-----------------------------|-------------------|
| 00409F59 | SD              | POP EBP                     | kernel32.77441174 |
| 00409F5A | - FF25 44B04000 | JMP DWORD PTR DS:[0x40B044] |                   |
| 00409F60 | SD              | POP EBP                     | kernel32.77441174 |

DS:[0040B044]=01206130

## 4.2 Segunda capa

Vamos a esa dirección y empezamos a analizar ahora en la segunda capa de código.

Comenzando con la segunda capa y seguimos analizando:

## 4.2.1 Obteniendo direcciones API's

|          |                |  |                         |
|----------|----------------|--|-------------------------|
| 009459C0 | EB 09          | JMP SHORT 009459D7                     |                         |
| 009459CE | 8B55 FC        | MOV EDX, DWORD PTR SS:[EBP-0x4]        |                         |
| 009459D1 | 83C2 01        | ADD EDX, 0x1                           |                         |
| 009459D4 | 8955 FC        | MOV DWORD PTR SS:[EBP-0x4], EDX        |                         |
| 009459D7 | 837D FC 0D     | CMP DWORD PTR SS:[EBP-0x4], 0xD        |                         |
| 009459DB | 73 21          | JNB SHORT 009459FE                     |                         |
| 009459DD | 8B45 FC        | MOV EAX, DWORD PTR SS:[EBP-0x4]        |                         |
| 009459E0 | 8B0C85 C402940 | MOV ECX, DWORD PTR DS:[EAX*4+0x9402C4] |                         |
| 009459E7 | 51             | PUSH ECX                               |                         |
| 009459E8 | 8B55 F8        | MOV EDX, DWORD PTR SS:[EBP-0x8]        | kernel32.7C800000       |
| 009459EB | 52             | PUSH EDX                               |                         |
| 009459EC | FF15 40639400  | CALL DWORD PTR DS:[0x946340]           | kernel32.GetProcAddress |
| 009459F2 | 8B4D FC        | MOV ECX, DWORD PTR SS:[EBP-0x4]        |                         |
| 009459F5 | 89048D 4063940 | MOV DWORD PTR DS:[ECX*4+0x946340], EAX |                         |
| 009459FC | EB D0          | JMP SHORT 009459CE                     |                         |
| 009459FE | 8BE5           | MOV ESP, EBP                           |                         |
| 00945A00 | 5D             | POP EBP                                | kernel32.7C800000       |

|                         |                |
|-------------------------|----------------|
| 00 00 00 00 00 00 00 00 | .....          |
| 00 00 00 00 00 00 3F 00 | .....?.        |
| 47 65 74 50 72 6F 63 41 | GetProcAddress |
| 64 64 72 65 73 73 00 00 | ddress..       |
| 56 69 72 74 75 61 6C 41 | VirtualA       |
| 6C 6C 6F 63 00 00 00 00 | lloc....       |
| 55 6E 6D 61 70 56 69 65 | UnmapVie       |
| 77 4F 66 46 69 6C 65 00 | wOfFile.       |
| 56 69 72 74 75 61 6C 50 | VirtualP       |
| 72 6F 74 65 63 74 00 00 | rotect..       |
| 4C 6F 61 64 4C 69 62 72 | LoadLibr       |
| 61 72 79 45 78 41 00 00 | aryExA..       |
| 47 65 74 4D 6F 64 75 6C | GetModul       |
| 65 48 61 6E 64 6C 65 41 | eHandleA       |
| 00 00 00 00 43 72 65 61 | ....Crea       |
| 74 65 46 69 6C 65 41 00 | teFileA.       |
| 53 65 74 46 69 6C 65 50 | SetFileP       |
| 6F 69 6E 74 65 72 00 00 | ointer..       |
| 57 72 69 74 65 46 69 6C | WriteFil       |
| 65 00 00 00 43 6C 6F 73 | e...Clos       |
| 65 48 61 6E 64 6C 65 00 | eHandle.       |
| 47 65 74 54 65 6D 70 50 | GetTempP       |
| 61 74 68 41 00 00 00 00 | athA....       |
| 6C 73 74 72 6C 65 6E 41 | lstrlenA       |
| 00 00 00 00 6C 73 74 72 | ....lstr       |

Es muy fácil entender esta rutina en la cual va cargando los nombres de las API's y las utiliza en conjunto con *GetProcAddress* para ir obteniendo las direcciones de cada una, haciendo un incremento en "ecx" y después por 4 y sumando el offset 0x9402C4 para sacar los nombres poco a poco conforme va incrementando 1 y hace lo mismo de vuelta con el offset 0x946340 para guardar la direcciones devueltas por *GetProcAddress* y las vuelve a acomodar formando una pequeña tabla, que posteriormente va a utilizar, como vemos no tiene demasiadas API's por lo tanto a mi parecer sospecho que descifrá una capa más adelante y seguirá con su labor, seguimos avanzando para ver si encontramos algo interesante.

## 4.2.2 Guardando Datos Encriptados

Seguidamente vemos que con la API *VirtualAlloc* nos creara una zona vacía de size 0x5400 con dirección 0x950000 y después pasa por una rutina que no tiene sentido pero que solo coloca 0 en toda la zona que nos creó lo cual como acabamos de crearla siempre estará vacía por lo tanto es una parte que a mi parecer no tiene mucho sentido, (imagen abajo)

|          |               |                                |                       |
|----------|---------------|--------------------------------|-----------------------|
| 00946110 | 55            | PUSH EBP                       |                       |
| 00946111 | 8BEC          | MOV EBP,ESP                    |                       |
| 00946113 | 6A 40         | PUSH 0x40                      |                       |
| 00946115 | 68 00300000   | PUSH 0x3000                    |                       |
| 0094611A | 8B45 08       | MOV EAX,DWORD PTR SS:[EBP+0x8] |                       |
| 0094611D | 50            | PUSH EAX                       |                       |
| 0094611E | 6A 00         | PUSH 0x0                       |                       |
| 00946120 | FF15 44639400 | CALL DWORD PTR DS:[0x946344]   | kernel32.VirtualAlloc |
| 00946126 | 5D            | POP EBP                        |                       |
| 00946127 | C3            | RETN                           |                       |
| 00946128 | CC            | INT3                           |                       |
| 00946129 | CC            | INT3                           |                       |
| 0094612A | CC            | INT3                           |                       |

|          |          |   |
|----------|----------|---|
| 0012FF80 | 00000000 | Address = NULL                          |
| 0012FF84 | 00005400 | Size = 5400 (21504.)                    |
| 0012FF88 | 00003000 | AllocationType = MEM_COMMIT MEM_RESERVE |
| 0012FF8C | 00000040 | Protect = PAGE_EXECUTE_READWRITE        |
| 0012FF90 | 0012FFC0 |   |
| 0012FF94 | 0094621D | RETURN to 0094621D from 00946110        |
| 0012FF98 | 00005400 |   |

| Registers (FPU) |          | <                 |
|-----------------|----------|-------------------|
| EAX             | 00950000 |                   |
| ECX             | 7C809B49 | kernel32.7C809B49 |

# WHITE PAPER

## SECCION QUE PONE A 0 O LIMPIA EL CONTENIDO DE LA NUEVA SECCION

|      |                |                                 |
|------|----------------|---------------------------------|
| 5A33 | 51             | PUSH ECX                        |
| 5A34 | C745 FC 000000 | MOV DWORD PTR SS:[EBP-0x4],0x0  |
| 5A3B | EB 09          | JMP SHORT 00945A46              |
| 5A3D | 8B45 FC        | MOV EAX,DWORD PTR SS:[EBP-0x4]  |
| 5A40 | 83C0 01        | ADD EAX,0x1                     |
| 5A43 | 8945 FC        | MOV DWORD PTR SS:[EBP-0x4],EAX  |
| 5A46 | 8B4D FC        | MOV ECX,DWORD PTR SS:[EBP-0x4]  |
| 5A49 | 3B4D 10        | CMP ECX,DWORD PTR SS:[EBP+0x10] |
| 5A4C | 73 0D          | JNB SHORT 00945A5B              |
| 5A4E | 8B55 08        | MOV EDX,DWORD PTR SS:[EBP+0x8]  |
| 5A51 | 0355 FC        | ADD EDX,DWORD PTR SS:[EBP-0x4]  |
| 5A54 | 8A45 0C        | MOV AL,BYTE PTR SS:[EBP+0xC]    |
| 5A57 | 8B02           | MOV BYTE PTR DS:[EDX],AL        |
| 5A59 | EB E2          | JMP SHORT 00945A3D              |
| 5A5B | 8B55           | MOV ESP,EBP                     |

Entramos como a una pequeña rutina en la cual si "eax" es mayor al size 5400 entonces nos saca fuera de la rutina, lo cual si seguimos avanzando podremos saber con exactitud el contenido que guarda y que movimientos hace:

|      |               |                                 |
|------|---------------|---------------------------------|
| 623D | 8955 E4       | MOV DWORD PTR SS:[EBP-0x1C],EDX |
| 6240 | 8B45 FC       | MOV EAX,DWORD PTR SS:[EBP-0x4]  |
| 6243 | 3B45 F0       | CMP EAX,DWORD PTR SS:[EBP-0x10] |
| 6246 | 73 53         | JNB SHORT 0094629B              |
| 6248 | 8B4D E4       | MOV ECX,DWORD PTR SS:[EBP-0x1C] |
| 624B | 51            | PUSH ECX                        |
| 624C | 8B55 DC       | MOV EDX,DWORD PTR SS:[EBP-0x24] |
| 624F | 52            | PUSH EDX                        |
| 6250 | E8 0BFCFFFF   | CALL 00945E60                   |
| 6255 | 83C4 08       | ADD ESP,0x8                     |
| 6258 | 8945 E0       | MOV DWORD PTR SS:[EBP-0x20],EAX |
| 625B | 8B45 E0       | MOV EAX,DWORD PTR SS:[EBP-0x20] |
| 625E | 50            | PUSH EAX                        |
| 625F | 8B0D A8649400 | MOV ECX,DWORD PTR DS:[0x9464A8] |
| 6265 | 034D E8       | ADD ECX,DWORD PTR SS:[EBP-0x18] |
| 6268 | 51            | PUSH ECX                        |
| 6269 | 8B15 88639400 | MOV EDX,DWORD PTR DS:[0x946388] |
| 626E | 8B55 FC       | ADD EDX,DWORD PTR SS:[EBP-0x4]  |

# WHITE PAPER

Seguimos avanzando dentro y podemos encontrar lo siguiente:

|          |                |                                 |
|----------|----------------|---------------------------------|
| 00945CB0 | 55             | PUSH EBP                        |
| 00945CB1 | 8BEC           | MOV EBP,ESP                     |
| 00945CB3 | 51             | PUSH ECX                        |
| 00945CB4 | C745 FC 000000 | MOV DWORD PTR SS:[EBP-0x4],0x0  |
| 00945CB8 | EB 09          | JMP SHORT 00945CC6              |
| 00945CBD | 8B45 FC        | MOV EAX,DWORD PTR SS:[EBP-0x4]  |
| 00945CC0 | 83C0 01        | ADD EAX,0x1                     |
| 00945CC3 | 8945 FC        | MOV DWORD PTR SS:[EBP-0x4],EAX  |
| 00945CC6 | 8B4D FC        | MOV ECX,DWORD PTR SS:[EBP-0x4]  |
| 00945CC9 | 3B4D 10        | CMP ECX,DWORD PTR SS:[EBP+0x10] |
| 00945CCC | 73 12          | JNB SHORT 00945CE0              |
| 00945CCE | 8B55 08        | MOV EDX,DWORD PTR SS:[EBP+0x8]  |
| 00945CD1 | 0355 FC        | ADD EDX,DWORD PTR SS:[EBP-0x4]  |
| 00945CD4 | 8B45 0C        | MOV EAX,DWORD PTR SS:[EBP+0xC]  |
| 00945CD7 | 0345 FC        | ADD EAX,DWORD PTR SS:[EBP-0x4]  |
| 00945CDA | 8A08           | MOV CL,BYTE PTR DS:[EAX]        |
| 00945CDC | 880A           | MOV BYTE PTR DS:[EDX],CL        |
| 00945CDE | EB DD          | JMP SHORT 00945CB0              |
| 00945CE0 | 8BES           | MOV ESP,EBP                     |
| 00945CE2 | 5D             | POP EBP                         |
| 00945CE3 | C3             | RETN                            |
| 00945CE4 | CC             | INT3                            |

|              |  |  |  |
|--------------|--|--|--|
| EBP=0012FF88 |  |  |  |
| ESP=0012FF84 |  |  |  |

| Address  | Hex dump                | ASCII    |
|----------|-------------------------|----------|
| 00950010 | 31 03 00 00 E9 03 00 00 | 1...0... |
| 00950018 | 29 04 00 00 E9 03 00 00 | )...0... |
| 00950020 | E9 03 00 00 E9 03 00 00 | 0...0... |
| 00950028 | E9 03 00 00 E9 03 00 00 | 0...0... |
| 00950030 | E9 03 00 00 E9 03 00 00 | 0...0... |
| 00950038 | E9 03 00 00 B1 04 00 00 | 0...0... |
| 00950040 | E7 1A BA 0E E9 AF 09 CD | 0...0... |
| 00950048 | C8 BB 01 4C AC 25 54 68 | 0...0... |
| 00950050 | 00 77 20 70 FB 6A 67 72 | 0...0... |
| 00950058 | C8 68 20 63 C8 69 6E 6F | 0...0... |
| 00950060 | DD 23 62 65 09 76 75 6E | 0...0... |
| 00950068 | 09 6D 6E 20 A5 4A 53 20 | 0...0... |
| 00950070 | C4 6A 64 65 FF 08 0D 0A | 0...0... |

Esta rutina nos guarda datos encriptados, en específico solo guarda 78 bytes y los coloca dentro de la sección que anteriormente fue creada, si nos fijamos bien podemos observar que pareciera ser el comienzo o parte de la cabecera de un .exe (un poco ofuscada por la encriptación), seguirá guardar de 78 bytes hasta llenar por completo la zona o que eax valga 5400 que es el tamaño de la nueva zona creada anteriormente, seguimos avanzando hasta que podamos encontrar alguna rutina que desencripte esto y lograr ver si el contenido lo ocupara par algún destroz, hasta ahora no tenemos al parecer el código que hace cambios en el sistema, sigamos..

## 4.2.3 Rutina de Descriptación

Bien seguidamente llegamos a la zona que nos descripta lo que guardo:

|          |               |                                  |
|----------|---------------|----------------------------------|
| 009462A4 | 8B45 F8       | MOV EAX, DWORD PTR SS:[EBP-0x8]  |
| 009462A7 | 83C0 04       | ADD EAX, 0x4                     |
| 009462AA | 8945 F8       | MOV DWORD PTR SS:[EBP-0x8], EAX  |
| 009462AD | 8B4D F8       | MOV ECX, DWORD PTR SS:[EBP-0x8]  |
| 009462B0 | 3B4D F0       | CMP ECX, DWORD PTR SS:[EBP-0x10] |
| 009462B3 | 73 39         | JNB SHORT 009462EE               |
| 009462B5 | 8B15 88639400 | MOV EDX, DWORD PTR DS:[0x946388] |
| 009462BB | 0355 F8       | ADD EDX, DWORD PTR SS:[EBP-0x8]  |
| 009462BE | 8B02          | MOV EAX, DWORD PTR DS:[EDX]      |
| 009462C0 | 0345 F8       | ADD EAX, DWORD PTR SS:[EBP-0x8]  |
| 009462C3 | 8B0D 88639400 | MOV ECX, DWORD PTR DS:[0x946388] |
| 009462C9 | 034D F8       | ADD ECX, DWORD PTR SS:[EBP-0x8]  |
| 009462CC | 8901          | MOV DWORD PTR DS:[ECX], EAX      |
| 009462CE | 8B55 F8       | MOV EDX, DWORD PTR SS:[EBP-0x8]  |
| 009462D1 | 81C2 E9030000 | ADD EDX, 0x3E9                   |
| 009462D7 | A1 88639400   | MOV EAX, DWORD PTR DS:[0x946388] |
| 009462DC | 0345 F8       | ADD EAX, DWORD PTR SS:[EBP-0x8]  |
| 009462DF | 3310          | XOR EDX, DWORD PTR DS:[EAX]      |
| 009462E1 | 8B0D 88639400 | MOV ECX, DWORD PTR DS:[0x946388] |
| 009462E7 | 034D F8       | ADD ECX, DWORD PTR SS:[EBP-0x8]  |
| 009462EA | 8911          | MOV DWORD PTR DS:[ECX], EDX      |
| 009462EC | EB B6         | JMP SHORT 009462A4               |
| 009462EE | E8 BDF7FFFF   | CALL 00945AB0                    |
| 009462F3 | 8B15 88639400 | MOV EDX, DWORD PTR DS:[0x946388] |

En esta rutina lo único que hace es tomar es sacar de la dirección donde están los bytes encriptados y sacar de 4 bytes (word) y los coloca en "eax" y después le suma 8 y los vuelve a guardar el resultado en la zona de donde lo saco , después a 8 le suma 0x3EA y al resultado 0x3F1 le hace un **XOR** con el contenido de "eax" que actualmente contendría el word que esta descriptando, el resultado lo contiene edx que posteriormente serían los 4 bytes descriptados y los coloca en la misma dirección de donde saco los encriptados, lo cual si revertimos estas mismas operación con restas y utilizando **XOR** podríamos así encriptar otra vez los datos , por lo tanto esta sería fácil la rutina de encriptación/descriptación, si vemos los datos que quedaron al final podemos observar un binario limpio:

# WHITE PAPER

| dress  | Hex dump                | ASCII    |
|--------|-------------------------|----------|
| 950000 | 4D 5A 90 00 03 00 00 00 | MZÉ.♦... |
| 950008 | 04 00 00 00 FF FF 00 00 | ♦... ..  |
| 950010 | B8 00 00 00 00 00 00 00 | @.....   |
| 950018 | 40 00 00 00 00 00 00 00 | @.....   |
| 950020 | 00 00 00 00 00 00 00 00 | .....    |
| 950028 | 00 00 00 00 00 00 00 00 | .....    |
| 950030 | 00 00 00 00 00 00 00 00 | .....    |
| 950038 | 00 00 00 00 C8 00 00 00 | ....b... |
| 950040 | 0E 1F BA 0E 00 B4 09 CD | ¶¶¶¶.+=  |
| 950048 | 21 B8 01 4C CD 21 54 68 | !@@L=!Th |
| 950050 | 69 73 20 70 72 6F 67 72 | is progr |
| 950058 | 61 6D 20 63 61 6E 6E 6F | an canno |
| 950060 | 74 20 62 65 20 72 75 6E | t be run |
| 950068 | 20 69 6E 20 44 4F 53 20 | in DOS   |
| 950070 | 6D 6F 64 65 2E 0D 0D 0A | mode.... |
| 950078 | 24 00 00 00 00 00 00 00 | \$...... |

## 4.2.4 Copiando datos desencryptados

Adelante nos encontramos con otro *VirtualAlloc* que nos creara otra zona pero esta vez de un tamaño diferente 0x9000 con dirección en 0x960000

|          |          |   |
|----------|----------|---|
| 0012FF54 | 7C809B50 | kernel32.7C809B50                       |
| 0012FF58 | FFFFFFFF |   |
| 0012FF5C | 00000000 | Address = NULL                          |
| 0012FF60 | 00009000 | Size = 9000 (36864.)                    |
| 0012FF64 | 00003000 | AllocationType = MEM_COMMIT MEM_RESERVE |
| 0012FF68 | 00000040 | Protect = PAGE_EXECUTE_READWRITE        |
| 0012FF6C | 00950000 |   |
| 0012FF70 | 00003000 |   |

Ahora sigamos hasta encontrar lo que guardara en la zona vacía, hasta que llegamos a aquí:

|        |                |                                  |
|--------|----------------|----------------------------------|
| 945CB4 | C745 FC 000000 | MOV DWORD PTR SS:[EBP-0x4], 0x0  |
| 945CB8 | EB 09          | JMP SHORT 00945CC6               |
| 945CBD | 8B45 FC        | MOV EAX, DWORD PTR SS:[EBP-0x4]  |
| 945CC0 | 83C0 01        | ADD EAX, 0x1                     |
| 945CC3 | 8945 FC        | MOV DWORD PTR SS:[EBP-0x4], EAX  |
| 945CC6 | 8B4D FC        | MOV ECX, DWORD PTR SS:[EBP-0x4]  |
| 945CC9 | 3B4D 10        | CMP ECX, DWORD PTR SS:[EBP+0x10] |
| 945CCC | 73 12          | JNB SHORT 00945CE0               |
| 945CCE | 8B55 08        | MOV EDX, DWORD PTR SS:[EBP+0x8]  |
| 945CD1 | 0355 FC        | ADD EDX, DWORD PTR SS:[EBP-0x4]  |
| 945CD4 | 8B45 0C        | MOV EAX, DWORD PTR SS:[EBP+0xC]  |
| 945CD7 | 0345 FC        | ADD EAX, DWORD PTR SS:[EBP-0x4]  |
| 945CDA | 8A08           | MOV CL, BYTE PTR DS:[EAX]        |
| 945CDC | 8B0A           | MOV BYTE PTR DS:[EDX], CL        |
| 945CDE | EB 0D          | JMP SHORT 00945CB8               |
| 945CE0 | 8BE5           | MOV ESP, EBP                     |
| 945CE2 | 5D             | POP EBP                          |
| 945CE3 | C3             | RETN                             |



# WHITE PAPER

En esa rutina que muestro en la imagen de arriba, lo que hace es que mete 400 bytes de la zona que anteriormente descripto y los mete en esta nueva zona que acaba de crear en la dirección 0x960000, al parecer solo guarda los datos de la cabecera sin guardar mas parte del código, seguimos...

|      |                |                                 |
|------|----------------|---------------------------------|
| 5CB3 | 51             | PUSH ECX                        |
| 5CB4 | C745 FC 000000 | MOV DWORD PTR SS:[EBP-0x4],0x0  |
| 5CB8 | EB 09          | JMP SHORT 00945CC6              |
| 5CBD | 8B45 FC        | MOV EAX,DWORD PTR SS:[EBP-0x4]  |
| 5CC0 | 83C0 01        | ADD EAX,0x1                     |
| 5CC3 | 8945 FC        | MOV DWORD PTR SS:[EBP-0x4],EAX  |
| 5CC6 | 8B4D FC        | MOV ECX,DWORD PTR SS:[EBP-0x4]  |
| 5CC9 | 3B4D 10        | CMP ECX,DWORD PTR SS:[EBP+0x10] |
| 5CCC | 73 12          | JNB SHORT 00945CE0              |
| 5CCE | 8B55 08        | MOV EDX,DWORD PTR SS:[EBP+0x8]  |
| 5CD1 | 0355 FC        | ADD EDX,DWORD PTR SS:[EBP-0x4]  |
| 5CD4 | 8B45 0C        | MOV EAX,DWORD PTR SS:[EBP+0xC]  |
| 5CD7 | 0345 FC        | ADD EAX,DWORD PTR SS:[EBP-0x4]  |
| 5CDA | 8A08           | MOV CL,BYTE PTR DS:[EAX]        |
| 5CDC | 8B0A           | MOV BYTE PTR DS:[EDX],CL        |
| 5CDE | EB 0D          | JMP SHORT 00945CB0              |
| 5CE0 | 8BE5           | MOV ESP,EBP                     |
| 5CE2 | 5D             | POP EBP                         |
| 5CE3 | C3             | RET                             |

En la misma rutina colocara otros 4400 bytes y empieza a colocar únicamente desde la sección .code del binario anterior, localizo la sección .code y la copia en esta nueva sección en la dirección 961000, después vuelve a entrar en esta misma rutina y guarda la sección .rdata con el size de 0x600 en la dirección 0x966000, después la sección .data con de tamaño 200 en la dirección 0x967000, y sigue con la sección .reloc de tamaño 400 en la dirección 0x968000,

Seguimos avanzando y podemos ver que crea otra sección con *VirtualAlloc* pero esta ocasión ocupa la dirección 0x400000 así que pone a 0 toda esa sección con el tamaño de 9000.

|          |          |   |
|----------|----------|---|
| 0012FF5C | 00400000 | Address = MigAutoP.00400000             |
| 0012FF60 | 00009000 | Size = 9000 (36864.)                    |
| 0012FF64 | 00003000 | AllocationType = MEM_COMMIT MEM_RESERVE |
| 0012FF68 | 00000040 | Protect = PAGE_EXECUTE_READWRITE        |
| 0012FF6C | 00950000 |   |
| 0012FF70 | 00003000 |   |

Más adelante podemos ver que utiliza la misma rutina que utilizo para guardar las secciones, la utiliza para poder copiar todo el contenido a de la dirección 0x966000 a la dirección 0x400000 sobrescribiendo su propia cabecera y ahora este mismo binario tiene una cabecera nueva con contenido nuevo.

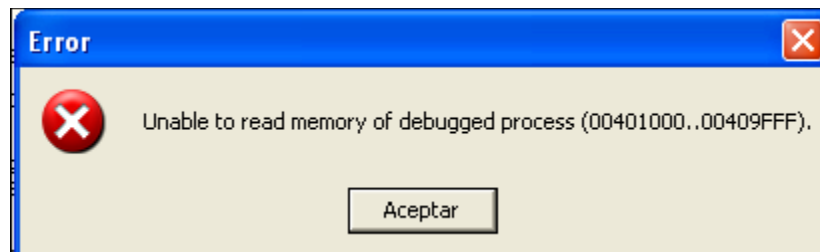
# WHITE PAPER

|    |               |                                  |
|----|---------------|----------------------------------|
| 1C | 894424 04     | MOV DWORD PTR SS:[ESP+084], EHX  |
| 20 | 8B15 B0649400 | MOV EDX, DWORD PTR DS:[0x9464B0] |
| 26 | 52            | PUSH EDX                         |
| 27 | C3            | RETN                             |
| 28 | 8BEE          | MOV ESP, EBP                     |

Saca el entry point y lo coloca en “edx” y hace un “push” al entry point de la zona que acaba de copiar en 0x400000 y pasa por un ret , por lo cual estaríamos ya por caer en la tercer capa.

## 4.3 Tercer capa

Sigamos más adelante y veamos que más tiene, pero en caso de perdernos o de reiniciar para eso ya habíamos hecho un dumped atrás con esa misma información por lo tanto ya tenemos el binario que sospecho es el que se encarga de hacer los destrozos, busquemos strings para ver algo que nos ayude a saber si tenemos mayores referencias y vemos



Nos muestra un error y no podemos ver los strings, pero intentemos con el dumped que teníamos y vemos:

| Address | Disassembly              | Text string   |
|---------|--------------------------|---|
| 01C33   | PUSH dumped1.00406000    | ASCII "%s\MigAutoPlay.exe"                            |
| 01CF1   | MOV ESI,dumped1.004061C8 | ASCII "PheeragIrefvba\Cbyupvxf\Flfgrz"                |
| 01D01   | PUSH dumped1.004061B4    | ASCII "SOFTWARE\MICROSOFT\"                           |
| 01D85   | PUSH dumped1.004061A8    | ASCII "EnableLUA"                                     |
| 01DCF   | PUSH dumped1.00406020    | ASCII "user32.dll"                                    |
| 01DDF   | PUSH dumped1.00406014    | ASCII "wsprintfA"                                     |
| 01E0F   | PUSH dumped1.004061F4    | ASCII "%s\MigAutoPlay.exe"                            |
| 01E26   | MOV ESI,dumped1.0040612C | ASCII "FBSGJNER\Zupebfbsg\Jvaqbjf\PheeragIrefvba\Eha" |
| 01E76   | PUSH dumped1.00406120    | ASCII "MigAutoPlay"                                   |
| 01EDF   | PUSH dumped1.00406020    | ASCII "user32.dll"                                    |
| 01EEF   | PUSH dumped1.00406014    | ASCII "wsprintfA"                                     |
| 01F1F   | PUSH dumped1.004061F4    | ASCII "%s\MigAutoPlay.exe"                            |
| 01F36   | MOV ESI,dumped1.0040612C | ASCII "FBSGJNER\Zupebfbsg\Jvaqbjf\PheeragIrefvba\Eha" |
| 01F86   | PUSH dumped1.00406120    | ASCII "MigAutoPlay"                                   |
| 0253B   | PUSH dumped1.00406258    | ASCII "Ole32.dll"                                     |
| 02552   | PUSH dumped1.00406248    | ASCII "OleAut32.dll"                                  |
| 02562   | PUSH dumped1.00406234    | ASCII "OleSavePictureFile"                            |
| 02576   | PUSH dumped1.0040621C    | ASCII "CreateStreamOnHGlobal"                         |
| 02590   | PUSH dumped1.0040620C    | ASCII "OleLoadPicture"                                |
| 0317E   | PUSH dumped1.00406270    | ASCII "GET"   |
| 0322E   | PUSH dumped1.00406270    | ASCII "GET"   |
| 032ED   | PUSH dumped1.00406298    | ASCII "Urlmon.dll"                                    |
| 03300   | PUSH dumped1.00406284    | ASCII "URLDownloadToFileA"                            |
| 0331B   | PUSH dumped1.00406020    | ASCII "user32.dll"                                    |
| 0332B   | PUSH dumped1.00406014    | ASCII "wsprintfA"                                     |
| 0335B   | PUSH dumped1.00406278    | ASCII "%s\1.jpg"                                      |
| 033FE   | PUSH dumped1.004062A4    | ASCII "google.com"                                    |
| 03410   | PUSH dumped1.004062A4    | ASCII "google.com"                                    |
| 03425   | PUSH dumped1.004062A4    | ASCII "google.com"                                    |
| 03BC0   | PUSH EBP                 | (Initial CPU selection)                               |
| 03BEF   | PUSH dumped1.004062CC    | ASCII "ZwAllLocateVirtualMemory"                      |

Bien vemos que tenemos al bicho después de todo esto lo hemos sacado, ahora falta analizar lo que hace este, ya que a simple vista se pueden ver algunos strings aun encriptados, algunas URL's y demás cosas, seguimos poco a poco, y empezamos a analizar este bicho por donde estábamos...

### 4.3.1 Modificación en el registro de Windows

Seguimos analizando y encontramos otra rutina en la cual empezara a desencriptar con XOR algunos strings para posteriormente usarlos:

| Registers (FPU) |  |
|-----------------|--|
| EAX             | 00000046   |
| ECX             | 0012FF38 ASCII "FBSGJNER\Zupebfbsg\Jvaqbjf\PheeragIrefvba" |
| EDX             | 0012FF38 ASCII "FBSGJNER\Zupebfbsg\Jvaqbjf\PheeragIrefvba" |
| EBX             | 7FFD6000   |
| ESP             | 0012FD00   |
| EBP             | 0012FD04   |
| ESI             | 004061A2 MigAutoP.004061A2                                 |
| EDI             | 0012FF62   |

Toma el primer dígito y le resta 0x34, después duplica el valor del operando en el registro con CDQ, idiv, le suma 0x41, y hace lo mismo con el resto de los caracteres hasta obtener un string desencriptado:

# WHITE PAPER

| Address  | Hex dump                | ASCII    |
|----------|-------------------------|----------|
| 0012FF38 | 53 4F 46 54 57 41 52 45 | SOFTWARE |
| 0012FF40 | 5C 4D 69 63 72 6F 73 6F | \Microso |
| 0012FF48 | 66 74 5C 57 69 6E 64 6F | ft\Windo |
| 0012FF50 | 77 73 5C 43 75 72 72 65 | ws\Curre |
| 0012FF58 | 6E 74 56 65 72 73 69 6F | ntVersio |
| 0012FF60 | 6E 00 12 00 9D 60 94 00 | n...0'0. |
| 0012FF68 | 01 00 00 80 C0 FF 12 00 | 0..Ç' +. |
| 0012FF70 | CB 3B 40 00 C8 00 96 00 | ff;0..0. |

Seguimos avanzando hasta ver donde qué datos guardara en el registro pues aquí esta descryptando la ruta para posteriormente guardar en esa dirección:

|          |          |   |
|----------|----------|---|
| 0012FCDC | 004021B1 | CALL to RegSetValueExA from MigAutoP.004021AE |
| 0012FCE0 | 00000048 | hKey = 0x48                                   |
| 0012FCE4 | 004061A4 | ValueName = "DNS"                             |
| 0012FCE8 | 00000000 | Reserved = 0x0                                |
| 0012FCEC | 00000001 | ValueType = REG_SZ                            |
| 0012FCF0 | 0012FD24 | Buffer = 0012FD24                             |

| Address  | Hex dump                | ASCII    |
|----------|-------------------------|----------|
| 0012FD24 | 43 3A 5C 44 6F 63 75 6D | C:\Docum |
| 0012FD2C | 65 6E 74 73 20 61 6E 64 | ents and |
| 0012FD34 | 20 53 65 74 74 69 6E 67 | Setting  |
| 0012FD3C | 73 5C 41 64 6D 69 6E 69 | s\Admini |
| 0012FD44 | 73 74 72 61 64 6F 72 5C | strador\ |
| 0012FD4C | 45 73 63 72 69 74 6F 72 | Escritor |
| 0012FD54 | 69 6F 5C 4D 69 67 41 75 | io\MigAu |
| 0012FD5C | 74 6F 50 6C 61 79 2E 65 | toPlay.e |
| 0012FD64 | 78 65 00 00 00 00 00 00 | xe.....  |
| 0012FD6C | 00 00 00 00 80 FD 12 00 | ....C²+. |

Esta guardando un path en HKCU software\Microsoft\Windows\currentversion, el path lo obtuvo con GetModuleFileName el cual devuelve el path de donde fue ejecutado el proceso para posteriormente guardarlo en esa llave del registro, seguimos

## 4.3.2 Rutina de Descriptación

Seguimos avanzando poco a poco y llegamos a una rutina algo similar pero con diferentes operaciones para el proceso de descriptación de un string esta vez algo pequeño

| Registers (FPU) |          |                     |
|-----------------|----------|---------------------|
| EAX             | 0012FF58 | ASCII "fipubfg.rkr" |
| ECX             | 0012FEC4 |                     |
| EDX             | 0012FF58 | ASCII "fipubfg.rkr" |

# WHITE PAPER

|             |                                 |  |
|-------------|---------------------------------|--|
| 0FBF11      | MOVSX EDX, BYTE PTR DS:[ECX]    |  |
| 83FA 61     | CMP EDX, 0x61                   | si son mayusculas envia a al offset 0x403890 |
| 7C 26       | JL SHORT MigAutoP.00403890      |  |
| 8B45 08     | MOV EAX, DWORD PTR SS:[EBP+0x8] |  |
| 0FBE08      | MOVSX ECX, BYTE PTR DS:[EAX]    |  |
| 83F9 7A     | CMP ECX, 0x7A                   |  |
| 7F 18       | JG SHORT MigAutoP.00403890      |  |
| 8B55 08     | MOV EDX, DWORD PTR SS:[EBP+0x8] |  |
| 0FBE02      | MOVSX EAX, BYTE PTR DS:[EDX]    |  |
| 83E8 54     | SUB EAX, 0x54                   | resta al primer caracter 0x54                |
| 99          | CDQ                             |  |
| B9 1A000000 | MOV ECX, 0x1A                   |  |
| F7F9        | IDIV ECX                        |  |
| 83C2 61     | ADD EDX, 0x61                   | suma al resultado 0x61                       |
| 8B45 08     | MOV EAX, DWORD PTR SS:[EBP+0x8] |  |
| 8810        | MOV BYTE PTR DS:[EAX], DL       | el resultado lo coloca en su lugar           |
| EB 2F       | JMP SHORT MigAutoP.004038BF     |  |
| 8B4D 08     | MOV ECX, DWORD PTR SS:[EBP+0x8] |  |
| 0FBE11      | MOVSX EDX, BYTE PTR DS:[ECX]    |  |
| 83FA 41     | CMP EDX, 0x41                   |  |
| 7C 24       | JL SHORT MigAutoP.004038BF      |  |
| 8B45 08     | MOV EAX, DWORD PTR SS:[EBP+0x8] |  |
| 0FBE08      | MOVSX ECX, BYTE PTR DS:[EAX]    |  |
| 83F9 5A     | CMP ECX, 0x5A                   |  |
| 7F 19       | JG SHORT MigAutoP.004038BF      |  |
| 8B55 08     | MOV EDX, DWORD PTR SS:[EBP+0x8] |  |
| 0FBE02      | MOVSX EAX, BYTE PTR DS:[EDX]    |  |
| 83E8 34     | SUB EAX, 0x34                   | resta al primer caracter 0x34                |
| 99          | CDQ                             |  |
| B9 1A000000 | MOV ECX, 0x1A                   |  |
| F7F9        | IDIV ECX                        |  |
| 83C2 41     | ADD EDX, 0x41                   | suma al resultado 0x41                       |
| 8B45 08     | MOV EAX, DWORD PTR SS:[EBP+0x8] |  |
| 8810        | MOV BYTE PTR DS:[EAX], DL       | el resultado lo coloca en su lugar           |
| EB 8B       | JMP SHORT MigAutoP.0040384C     |  |
| 8B45 FC     | MOV EAX, DWORD PTR SS:[EBP-0x4] |  |

En esta rutina (imagen arriba) podemos observar que toma cada carácter y compara de la “A” hasta la “Z” si es mayúscula o minúscula, de acuerdo a la a eso hace diferentes operaciones, en este caso todas son minúsculas así que las operaciones que realiza como muestro en la imagen seria resta al primer carácter 0x54 y al resultado le suma 0x61, el cual vuelve a colocar en su lugar y hace lo mismo con los próximos caracteres, si vamos al final de la rutina podremos ver que es lo que descripto:

| Address  | Hex dump                | ASCII     |
|----------|-------------------------|-----------|
| 0012FF58 | 73 76 63 68 6F 73 74 2E | svchost.  |
| 0012FF60 | 65 78 65 00 C0 FF 12 00 | exe. L .  |
| 0012FF68 | FB 3C 40 00 C0 3B 40 00 | '<@. L;@. |

## 4.3.3 Creación del proceso suspendido

Finalmente descifrado “svchost.exe”, el cual es un proceso del sistema que aloja varios servicios de Windows, posiblemente adelante inyectara el proceso o algo similar, seguimos avanzando...

Encontramos algo interesante y es la API que se encargara de crear el proceso svchost = CreateProcessA, aquí es algo importante para poder seguirle el paso y saber qué es lo que hará con él, veamos:

```

0012FE00 00402EF1 CALL to CreateProcessA from MigAutoP.00402EEE
0012FE04 00000000 ModuleFileName = NULL
0012FE08 0012FF58 CommandLine = "svchost.exe"
0012FE0C 00000000 pProcessSecurity = NULL
0012FE10 00000000 pThreadSecurity = NULL
0012FE14 00000000 InheritHandles = FALSE
0012FE18 00000004 CreationFlags = CREATE_SUSPENDED
0012FE1C 00000000 pEnvironment = NULL
0012FE20 00000000 CurrentDir = NULL
0012FE24 0012FEC4 pStartupInfo = 0012FEC4
0012FE28 0012FF14 pProcessInfo = 0012FF14
0012FE2C 7C80236B kernel32.CreateProcessA

```

Hay tenemos los parámetro que utiliza y lo que es de gran importancia es conocer el PID que es el identificador del proceso que creara en modo suspendido, es importante conocerlo para poder seguir el paso a lo que hace en caso de que inyecte código en el mismo, en pProcessInfo nos retornara información de acuerdo al Process\_Information structure en el tercer dword tendremos a nuestro identificador del proceso el cual nos devolverá en hexadecimal:

```
typedef struct _PROCESS_INFORMATION {
```

```
    HANDLE hProcess;
```

```
    HANDLE hThread;
```

```
    DWORD dwProcessId;
```

```
    DWORD dwThreadId;
```

| Address  | Hex dump                | ASCII    |
|----------|-------------------------|----------|
| 0012FF14 | 5C 00 00 00 60 00 00 00 | \...\... |
| 0012FF1C | B8 01 00 00 08 0B 00 00 | @0..i8.. |
| 0012FF24 | 01 00 00 00 00 00 00 00 | 0.....   |

El valor del identificador es 01B8h = 440d :

|                              |     |               |
|------------------------------|-----|---------------|
| svchost.exe                  | 440 | Administrador |
| System                       | 4   | SYSTEM        |
| Proceso inactivo del sistema | 0   | SYSTEM        |

## 4.3.4 Detección del Sistema Operativo

Más adelante invoca a GetVersionEx lo cual recupera información del sistema operativo,

La estructura es la siguiente:

```
typedef struct _OSVERSIONINFO {
    DWORD dwOSVersionInfoSize;
    DWORD dwMajorVersion;
    DWORD dwMinorVersion;
    DWORD dwBuildNumber;
    DWORD dwPlatformId;
    TCHAR szCSDVersion[128];
} OSVERSIONINFO;
```

| Address  | Hex dump                | ASCII     |
|----------|-------------------------|-----------|
| 0012FDB8 | 94 00 00 00 05 00 00 00 | ô...\$... |
| 0012FDC0 | 01 00 00 00 28 0A 00 00 | @...(...  |
| 0012FDC8 | 02 00 00 00 53 65 72 76 | @...Serv  |
| 0012FDD0 | 69 63 65 20 50 61 63 6B | ice Pack  |
| 0012FDD8 | 20 33 00 00 00 00 00 00 | 3.....    |

# WHITE PAPER

|         |               |      |                             |
|---------|---------------|------|-----------------------------|
| 0403996 | 83BD 64FFFFFF | CMP  | DWORD PTR SS:[EBP-0x9C],0x5 |
| 040399D | 75 15         | JNZ  | SHORT MigAutoP.004039B4     |
| 040399F | 83BD 68FFFFFF | CMP  | DWORD PTR SS:[EBP-0x98],0x1 |
| 04039A6 | 72 0C         | JB   | SHORT MigAutoP.004039B4     |
| 04039A8 | C785 5CFFFFFF | MOV  | DWORD PTR SS:[EBP-0xA4],0x1 |
| 04039B2 | EB 0A         | JMP  | SHORT MigAutoP.004039BE     |
| 04039B4 | C785 5CFFFFFF | MOV  | DWORD PTR SS:[EBP-0xA4],0x0 |
| 04039BE | 8B95 5CFFFFFF | MOV  | EDX,DWORD PTR SS:[EBP-0xA4] |
| 04039C4 | 8955 FC       | MOV  | DWORD PTR SS:[EBP-0x4],EDX  |
| 04039C7 | 837D FC 00    | CMP  | DWORD PTR SS:[EBP-0x4],0x0  |
| 04039CB | 74 06         | JE   | SHORT MigAutoP.004039D3     |
| 04039CD | B0 01         | MOV  | AL,0x1                      |
| 04039CF | EB 04         | JMP  | SHORT MigAutoP.004039D5     |
| 04039D1 | EB 02         | JMP  | SHORT MigAutoP.004039D5     |
| 04039D3 | 32C0          | XOR  | AL,AL                       |
| 04039D5 | 8BE5          | MOV  | ESP,EBP                     |
| 04039D7 | 5D            | POP  | EBP                         |
| 04039D8 | C3            | RET  |                             |
| 04039D9 | CC            | INT3 |                             |

hace una comparación con dwMajorVersion sea igual a 5 y dwMinorVersion sea igual a 1 , (imagen arriba) para saber si el sistema operativo es Windows XP y de ser así sale de la rutina con eax valiendo 1

Ejemplo:

*.if dwMajorVersion ==5 Then*

*.if dwMinorVersion == 1 Then*

*Version = Windows XP"*

*Return = True*

|             |      |                         |
|-------------|------|-------------------------|
| 83F8 01     | CMP  | EAX,0x1                 |
| 75 4F       | JNZ  | SHORT MigAutoP.00402AE3 |
| E8 17F2FFFF | CALL | MigAutoP.00401CB0       |

Así podrá decidir qué camino tomar, en este caso al ser Windows XP toma un camino el cual seguiremos avanzando posteriormente haremos pruebas con Windows 7, seguimos

**Con Windows 7:**

Salta a una dirección de memoria lo cual trata de evitar pasar por la API ZwSetInformationProcess que ocasionaría un DoS del sistema en alguna versiones de Windows 7



## 4.3.5 Inyectando en el proceso suspendido

Seguimos Windows XP:

Después de un rato seguir avanzando y pasar por diferentes API's que verificaban si el usuario estaba dentro del grupo administradores y ajustar los permisos, encontramos una zona donde utiliza VirtualAlloc para reservar un espacio vacío en memoria de size 0x6000 en la dirección 0x9B0000:

|          |          |   |
|----------|----------|---|
| 0012FE30 | 00403009 | CALL to VirtualAlloc from MigAutoP.00403006   |
| 0012FE34 | 00000000 | Address = NULL                                |
| 0012FE38 | 00006000 | Size = 6000 (24576.)                          |
| 0012FE3C | 00001000 | AllocationType = MEM_COMMIT                   |
| 0012FE40 | 00000040 | Protect = PAGE_EXECUTE_READWRITE              |
| 0012FE44 | 7C809AE1 | kernel32.VirtualAlloc                         |
| 0012FE48 | 0012FF64 |   |
| 0012FE4C | 00402BC7 | RETURN to MigAutoP.00402BC7 from MigAutoP.004 |

| Registers (FPU) |                 |
|-----------------|-----------------|
| EAX             | 009B0000        |
| ECX             | 7C809B49 kernel |

Después con ReadProcessMemory guarda contenido en esa dirección de memoria creada, en la cual guarda la información a partir del BaseAddress 0x1000000

|          |          |  |
|----------|----------|--|
| 0012FE28 | 00402FCD | CALL to ReadProcessMemory from MigAutoP.00402FCA |
| 0012FE2C | 0000005C | hProcess = 0000005C (window)                     |
| 0012FE30 | 01000000 | pBaseAddress = 0x1000000                         |
| 0012FE34 | 009B0000 | Buffer = 009B0000                                |
| 0012FE38 | 00006000 | BytesToRead = 6000 (24576.)                      |
| 0012FE3C | 0012FF24 | pBytesRead = 0012FF24                            |
| 0012FE40 | 7C802100 | RETURN to kernel32.ReadProcessMemory             |

| Hex dump                | ASCII      |
|-------------------------|------------|
| 4D 5A 90 00 03 00 00 00 | MZ... ..   |
| 04 00 00 00 FF FF 00 00 | ... ..     |
| B8 00 00 00 00 00 00 00 | @.....     |
| 40 00 00 00 00 00 00 00 | @.....     |
| 00 00 00 00 00 00 00 00 | .....      |
| 00 00 00 00 00 00 00 00 | .....      |
| 00 00 00 00 00 00 00 00 | .....      |
| 00 00 00 00 E0 00 00 00 | ....ó...   |
| 0E 1F BA 0E 00 B4 09 CD | ...  .+. = |
| 21 B8 01 4C CD 21 54 68 | !@L=!Th    |
| 69 73 20 70 72 6F 67 72 | is progr   |
| 61 6D 20 63 61 6E 6E 6F | am canno   |

Más adelante Crea una sección con algunas API's de la librería .ntdll ZwCreateSection y posteriormente la lee con ZwMapViewOfsection, creando un mapa de memoria en la dirección

# WHITE PAPER

0x9c0000 el cual lo rellenara con *RtlMoveMemory* y copiara su propio contenido, es decir, 0x400000 , al crear el mapa en memoria cualquier cambio que se le haga a la memoria mapeada se verá reflejada en el contenido real del proceso , por lo tanto el proceso ya esta inyectado.

## 4.3.6 Modificando el entryptpoint

Seguimos avanzando podemos encontrar esta zona y es muy importante:

|         |               |                                 |
|---------|---------------|---------------------------------|
| 040202D | 0F85 3A010000 | JNZ MigAutoP.00402E6D           |
| 0402033 | 8B95 2CFFFFFF | MOV EDX,DWORD PTR SS:[EBP-0x04] |
| 0402039 | 0355 AC       | ADD EDX,DWORD PTR SS:[EBP-0x54] |
| 040203C | C602 68       | MOV BYTE PTR DS:[EDX],0x68      |
| 040203F | 8B45 0C       | MOV EAX,DWORD PTR SS:[EBP+0xC]  |
| 0402042 | 2B45 CC       | SUB EAX,DWORD PTR SS:[EBP-0x34] |
| 0402045 | 0385 0CFFFFFF | ADD EAX,DWORD PTR SS:[EBP-0xF4] |
| 0402048 | 8B8D 2CFFFFFF | MOV ECX,DWORD PTR SS:[EBP-0xD4] |
| 0402051 | 034D AC       | ADD ECX,DWORD PTR SS:[EBP-0x54] |
| 0402054 | 8941 01       | MOV DWORD PTR DS:[ECX+0x1],EAX  |
| 0402057 | 8B95 2CFFFFFF | MOV EDX,DWORD PTR SS:[EBP-0xD4] |
| 040205D | 0355 AC       | ADD EDX,DWORD PTR SS:[EBP-0x54] |
| 0402060 | C642 05 C3    | MOV BYTE PTR DS:[EDX+0x5],0xC3  |
| 0402064 | 8B45 A8       | MOV EAX,DWORD PTR SS:[EBP-0x58] |
| 0402067 | 8985 08FFFFFF | MOV DWORD PTR SS:[EBP-0xF8],EAX |
| 040206D | 8B8D 08FFFFFF | MOV ECX,DWORD PTR SS:[EBP-0xF8] |
| 0402073 | 51            | PUSH ECX                        |

Lo que hace aquí es llegar a la dirección donde se encuentra ubicado el EntryPoint del proceso suspendido de acuerdo a la información de la cabecera es "AdressOfEntryPoint" = 0x2509

|          |                                     |
|----------|-------------------------------------|
| 00002C00 | SizeOfCode = 2C00 (11264.)          |
| 00000A00 | SizeOfInitializedData = A00 (2560.) |
| 00000000 | SizeOfUninitializedData = 0x0       |
| 00002509 | AddressOfEntryPoint = 0x2509        |
| 00001000 | BaseOfCode = 0x1000                 |
| 00004000 | BaseOfData = 0x4000                 |

Lo cual hace la suma a la dirección mapeada y ubicar el entryptpoint que sobrescribirá con una redirección:

| Address  | Hex dump                |   |                          |
|----------|-------------------------|---|--------------------------|
| 009B2509 | 68 10 44 09 00 C3 FF 56 | = | 68 10440900 PUSH 0x94410 |
| 009B2511 | 57 68 A2 2E 00 01 FF 15 |   | C3 RETN                  |

# WHITE PAPER

Bien ya tenemos el EntryPoint que sobrescribe, lo cual es muy importante ya que en adelante solo se esperara algún *ResumeThread* para que continúe la ejecución del proceso inyectado, por lo tanto modificaremos algunos bytes de tal manera que se ejecute un loop infinito con el cual nos dé tiempo y podamos adjuntar el proceso y poder seguir la ejecución del malware en el proceso inyectado, por lo tanto quedaría algo así:

| ress  | Hex dump                |       |                             |
|-------|-------------------------|-------|-----------------------------|
| B2509 | EB FE 44 09 00 C3 FF 56 | 0000  | ADD BYTE PTR DS:[EAX],AL    |
| B2511 | 57 68 A2 2E 00 01 FF 15 | EB FE | JMP SHORT MigAutoP.00407554 |
|       |                         | 0000  | ADD BYTE PTR DS:[EAX],AL    |

Con eso basta para poder hacer el loop, seguimos avanzado y encontramos el *ResumeThread* pero ya tenemos modificado el EntryPoint así que no hay problema se quedara corriendo el loop, si avanzamos un poco mas llegamos a un *ExitProcess* y fin del proceso.

## 5 Proceso inyectado

Adjuntamos ahora el proceso y ponemos pausa a la ejecución y caemos en el loop que colocamos:

|        |       |                                      |
|--------|-------|--------------------------------------|
| 002508 | 90    | NOP                                  |
| 002509 | EB FE | JMP SHORT svchost.<ModuleEntryPoint> |
| 00250B | 44    | INC ESP                              |
| 00250C | 0000  | ADD DWORD PTR DS:[EAX],EAX           |

Colocamos el push 0x94410 que el malware quería escribir y llegamos al entryptoint del injerto que hizo el malware:

|             |              |
|-------------|--------------|
| 68 10440900 | PUSH 0x94410 |
| C3          | RETN         |

|         |             |                            |
|---------|-------------|----------------------------|
| 0094410 | 55          | PUSH EBP                   |
| 0094411 | 8BEC        | MOV EBP,ESP                |
| 0094413 | 83EC 44     | SUB ESP,0x44               |
| 0094416 | 56          | PUSH ESI                   |
| 0094417 | 57          | PUSH EDI                   |
| 0094418 | E8 83E3FFFF | CALL 000927A0              |
| 009441D | 8945 F4     | MOV DWORD PTR SS:[EBP]     |
| 0094420 | 8B45 F4     | MOV EAX,DWORD PTR SS:[EBP] |
| 0094423 | 58          | PUSH EAX                   |

## 5.1 Modificación en el registro de Windows

Seguimos analizando a partir del entryptpoint y avanzamos, un poco más adelante podremos encontrarnos una rutina que descriptará otro string, la rutina es similar a la anteriormente mostrada comparando si son mayúsculas o minúsculas y a partir de ahí hace las operaciones XOR correspondientes, y después ocupara el string descriptado para usarlo con *RegOpenKey* y retirar los valores que anteriormente había guardado

```
ASCII "FBSGJNER\Zvpebfbsg\Jvaqbjf\PheeragIrefvba"
```

|          |          |  |
|----------|----------|--|
| 0007FBD4 | 0009213D | CALL to <i>RegOpenKeyExA</i> from 0009213A           |
| 0007FBD8 | 80000001 | hKey = HKEY_CURRENT_USER                             |
| 0007FBD8 | 0007FE30 | Subkey = "SOFTWARE\Microsoft\Windows\CurrentVersion" |
| 0007FBE0 | 00000000 | Reserved = 0x0                                       |
| 0007FBE4 | 00020019 | Access = KEY_READ                                    |
| 0007FBE8 | 0007FE5C | pHandle = 0007FE5C                                   |
| 0007FBEC | 77DA7842 | ADVAPI32.RegOpenKeyExA                               |
| 0007FBE0 | 0007FE6C |  |

|          |          |   |
|----------|----------|---|
| 0007FBCC | 00092331 | CALL to <i>RegQueryValueExA</i> from 0009232E |
| 0007FBD0 | 0000006C | hKey = 0x6C                                   |
| 0007FBD4 | 000961A4 | ValueName = "DNS"                             |
| 0007FBD8 | 00000000 | Reserved = NULL                               |
| 0007FBD8 | 00000000 | pValueType = NULL                             |
| 0007FBE0 | 0007FE64 | Buffer = 0007FE64                             |
| 0007FBE4 | 0007FE28 | pBufSize = 0007FE28                           |
| 0007FBE8 | 77DA7AAB | RETURN to ADVAPI32.RegQueryValueExA           |

Verifica su existencia con *CreateFile* y como no encuentra nada, sigue su ejecución pasando primero a descriptar otro string:

```
ASCII "FBSGJNER\Zvpebfbsg\Jvaqbjf\PheeragIrefvba\Eha"
```

```
ASCII "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
```

El cual mismo se establecerá en la llave de registro encargada de cargar los programas para iniciar con Windows

|          |          |   |
|----------|----------|---|
| 0007F964 | 000921B1 | CALL to <i>RegSetValueExA</i> from 000921AE |
| 0007F968 | 00000078 | hKey = 0x78                                 |
| 0007F96C | 00096120 | ValueName = "MigAutoPlay"                   |
| 0007F970 | 00000000 | Reserved = 0x0                              |
| 0007F974 | 00000001 | ValueType = REG_SZ                          |
| 0007F978 | 0007FAB4 | Buffer = 0007FAB4                           |
| 0007F97C | 00000047 | BufSize = 47 (71.)                          |
| 0007F980 | 77DAEAD7 | ADVAPI32.RegSetValueExA                     |

```
0007FAB4 ASCII ""C:\Documents and Settings\All Users\Datos de programa\MigAutoPlay.exe""
```

## 5.2 Copiando y eliminando Ficheros

Guarda el path obtenido por *SHGetSpecialFolderPath* y al cual le concateno "MigAutoPlay.exe", en esa dirección aún no se encuentra ningún ejecutable con ese nombre así que debe copiarlo:

```
0007FBE4 00092395 CALL to CopyFileA from 00092392
0007FBE8 0007FE64 ExistingFileName = "C:\Documents and Settings\Administrador\Escritorio\MigAutoPlay.exe"
0007FBEC 0007FD1C NewFileName = "C:\Documents and Settings\All Users\Datos de programa\MigAutoPlay.exe"
0007FBF0 00000000 FailIfExists = FALSE
0007FBF4 7C8286D6 kernel32.CopyFileA
```

Pasando esto, invoca a "sleep" con el parámetro 600 ms

```
0007FBF4 000923BD CALL to Sleep from 000923BA
0007FBF8 00000258 Timeout = 600. ms
0007FBFC 7C802446 kernel32.Sleep
0007FC00 00000000
```

Seguido elimina el binario con *DeleteFile*, lo cual es la muestra principal de donde se había empezado a analizar.

## 5.3 Copiando y eliminando llaves del registro de Windows

Poco más adelante descrypta otro string con la misma rutina que ha utilizado:

```
ASCII "Flfgz\PheeragPbagebyFrg\Pbageby\Fnsr0bbg\"
```

```
ASCII "System\CurrentControlSet\Control\SafeBoot\"
```

En esa llave se encuentran las configuraciones para entrar en "Modo Seguro", este es otro de los movimientos importantes que hará en el sistema, si seguimos vemos:

# WHITE PAPER

|          |          |  |
|----------|----------|--|
| 0007FD9C | 0009213D | CALL to <b>RegOpenKeyExA</b> from 0009213A           |
| 0007FDA0 | 80000002 | hKey = HKEY_LOCAL_MACHINE                            |
| 0007FDA4 | 0007FDE4 | Subkey = "System\CurrentControlSet\Control\SafeBoot" |
| 0007FDA8 | 00000000 | Reserved = 0x0                                       |
| 0007FDAC | 000F003F | Access = KEY_ALL_ACCESS                              |
| 0007FDB0 | 0007FF20 | pHandle = 0007FF20                                   |
| 0007FDB4 | 77D92842 | ADVAPI32.RegOpenKeyExA                               |

|          |          |   |
|----------|----------|---|
| 0007FD7C | 000922BD | CALL to <b>RegCreateKeyExA</b> from 000922BA              |
| 0007FD80 | 80000002 | hKey = HKEY_LOCAL_MACHINE                                 |
| 0007FD84 | 0007FE10 | Subkey = "System\CurrentControlSet\Control\SafeBoot\mini" |
| 0007FD88 | 00000000 | Reserved = 0x0  |
| 0007FD8C | 00000000 | Class = NULL  |
| 0007FD90 | 00000000 | Options = REG_OPTION_NON_VOLATILE                         |
| 0007FD94 | 000F003F | Access = KEY_ALL_ACCESS                                   |
| 0007FD98 | 00000000 | pSecurity = NULL  |
| 0007FD9C | 0007FF54 | pHandle = 0007FF54  |
| 0007FDA0 | 00000000 | pDisposition = NULL                                       |
| 0007FDA4 | 77DAE9E4 | ADVAPI32.RegCreateKeyExA                                  |

|          |          |   |
|----------|----------|---|
| 0007FDC0 | 00091465 | CALL to <b>SHCopyKeyA</b> from 00091462 |
| 0007FDC4 | 00000078 | hSrcKey = 0x78                          |
| 0007FDC8 | 00096118 | SubKey = "Minimal"                      |
| 0007FDCC | 00000074 | hDestKey = 0x74                         |
| 0007FDD0 | 00000000 | Reserved = 0x0                          |
| 0007FDD4 | 77D92842 | ADVAPI32.SHCopyKeyA                     |

|          |          |   |
|----------|----------|---|
| 0007FDC8 | 00091497 | CALL to <b>SHDeleteKeyA</b> from 00091491 |
| 0007FDCC | 00000078 | hKey = 0x78                               |
| 0007FDD0 | 00096118 | SubKey = "Minimal"                        |
| 0007FDD4 | 7C93216A | ntdll.7C93216A                            |

Bien como muestro en las imágenes de arriba crea una entrada nueva en el registro con el nombre de "System\CurrentControlSet\Control\SafeBoot\mini" y copia hay dentro el contenido de "System\CurrentControlSet\Control\SafeBoot\Minimal" para posteriormente borrarla.

Seguimos y hace las mismas operaciones con "System\CurrentControlSet\Control\SafeBoot\NetWork" copiándolo a "System\CurrentControlSet\Control\SafeBoot\net" y posteriormente eliminando la llave, estas son muy importantes ya que son las que se encargan de arrancar el sistema en "Modo seguro" normal y con funciones de red.

## 5.4 Desenscriptando datos de conexión la red

Seguimos avanzando y ahora desenscriptará con la misma rutina que ha venido desenscriptando strings, lo que parece ser una dirección URL de una imagen:

```
000970E4 ASCII "uggc://62.76.191.238/rukei_2bnv/hcybnq/vzt.wct"
```

```
4 000970E4 ASCII "http://62.76.191.238/ehxrv_2oai/upload/img.jpg"
```

Mas adelante empezara con la creación de un Thread en la dirección 093D70

|          |          |   |
|----------|----------|---|
| 0007FF34 | 00094E41 | CALL to <b>CreateThread</b> from 00094E3E |
| 0007FF38 | 00000000 | pSecurity = NULL                          |
| 0007FF3C | 00000000 | StackSize = 0x0                           |
| 0007FF40 | 00093D70 | ThreadFunction = 00093D70                 |
| 0007FF44 | 00000000 | pThreadParm = NULL                        |
| 0007FF48 | 00000000 | CreationFlags = 0                         |
| 0007FF4C | 00000000 | pThreadId = NULL                          |
| 0007FF50 | 7C8106C7 | kernel32.CreateThread                     |

Sigue su ejecución normal e invoca un Sleep con el parámetro 600000ms = 600 segundos, si esperamos a que pasa por esta instrucción, entonces entra la función del Thread en la cual lo que hace es hacer una serie de comprobaciones en los registros o los datos que guardo anteriormente.

## 5.5 Estableciendo controles de la ventana de dialogo

Siguiendo el camino por el que íbamos empieza a crear la ventana con CreateWindowExA lo cual cambiamos los parámetros para que el width height sea de 0x30 y podamos manipular en caso de que se interponga en la pantalla, seguidamente empieza a cargar nombres de algunas APIs para poder usarlas con *GetProcAddress*.

# WHITE PAPER

|      |             |                                  |                                |
|------|-------------|----------------------------------|--------------------------------|
| 4743 | 68 7C640900 | PUSH 0x9647C                     | ASCII "BitBit"                 |
| 4748 | 8B4D 9C     | MOV ECX, DWORD PTR SS:[EBP-0x64] |                                |
| 474B | 51          | PUSH ECX                         |                                |
| 474C | E8 1FD9FFFF | CALL 00092070                    |                                |
| 4751 | 83C4 08     | ADD ESP, 0x8                     |                                |
| 4754 | 8945 F0     | MOV DWORD PTR SS:[EBP-0x10], EAX |                                |
| 4757 | 68 6C640900 | PUSH 0x9646C                     | ASCII "DeleteObject"           |
| 475C | 8B55 9C     | MOV EDX, DWORD PTR SS:[EBP-0x64] |                                |
| 475F | 52          | PUSH EDX                         |                                |
| 4760 | E8 0BD9FFFF | CALL 00092070                    |                                |
| 4765 | 83C4 08     | ADD ESP, 0x8                     |                                |
| 4768 | 8945 E4     | MOV DWORD PTR SS:[EBP-0x1C], EAX |                                |
| 476B | 68 5C640900 | PUSH 0x9645C                     | ASCII "SelectObject"           |
| 4770 | 8B45 9C     | MOV EAX, DWORD PTR SS:[EBP-0x64] |                                |
| 4773 | 50          | PUSH EAX                         |                                |
| 4774 | E8 F7D8FFFF | CALL 00092070                    |                                |
| 4779 | 83C4 08     | ADD ESP, 0x8                     |                                |
| 477C | 8945 F4     | MOV DWORD PTR SS:[EBP-0xC], EAX  |                                |
| 477F | 68 44640900 | PUSH 0x96444                     | ASCII "CreateCompatibleBitmap" |
| 4784 | 8B4D 9C     | MOV ECX, DWORD PTR SS:[EBP-0x64] |                                |
| 4787 | 51          | PUSH ECX                         |                                |
| 4788 | E8 E3D8FFFF | CALL 00092070                    |                                |
| 478D | 83C4 08     | ADD ESP, 0x8                     |                                |
| 4790 | 8945 F8     | MOV DWORD PTR SS:[EBP-0x8], EAX  |                                |
| 4793 | 68 30640900 | PUSH 0x96430                     | ASCII "CreateCompatibleDC"     |
| 4798 | 8B55 9C     | MOV EDX, DWORD PTR SS:[EBP-0x64] |                                |
| 479B | 52          | PUSH EDX                         |                                |
| 479E | E8 0BD9FFFF | CALL 00092070                    |                                |

Seguidamente a crear los botones:

|          |          |   |
|----------|----------|---|
| 0007F1B0 | 00094FE9 | CALL to CreateWindowExA from 00094FE6                         |
| 0007F1B4 | 00000000 | ExtStyle = 0  |
| 0007F1B8 | 00096420 | Class = "Button"  |
| 0007F1BC | 0007F740 | WindowName = "Submit"   |
| 0007F1C0 | 50001000 | Style = WS_CHILD WS_VISIBLE 1000                              |
| 0007F1C4 | 000002EC | X = 2EC (748.)  |
| 0007F1C8 | 0000026B | Y = 26B (619.)  |
| 0007F1CC | 00000096 | Width = 96 (150.)   |
| 0007F1D0 | 00000019 | Height = 19 (25.)   |
| 0007F1D4 | 001A02FE | hParent = 001A02FE ('Dialog Window', class='My Window Class') |
| 0007F1D8 | 00000BB8 | hMenu = 00000BB8  |
| 0007F1DC | 00000000 | hInst = NULL  |
| 0007F1E0 | 00000000 | lParam = NULL   |
| 0007F1E4 | 75005100 | RETURN: HINSTANCE = 0, HINSTANCE = 0                          |



## 5.6 Monitor y cierre de procesos en el sistema

Crea un snapshot de los procesos corriendo actualmente:

|          |          |  |
|----------|----------|--|
| 009BFE50 | 00093A01 | CALL to CreateToolhelp32Snapshot from 000939FE |
| 009BFE54 | 00000002 | Flags = TH32CS_SNAPPROCESS                     |
| 009BFE58 | 00000000 | ProcessID = 0x0                                |
| 009BFE5C | 7C865B1F | kernel32.CreateToolhelp32Snapshot              |
| 009BFE60 | 00093A08 |  |

Seguido empieza a desencriptar de nuevo algunos strings que si nos damos cuenta son algunas cosas que seguramente impedirá ejecutar en el sistema, por ejemplo:

|               |                     |
|---------------|---------------------|
| PUSH 0x97000  | ASCII "taskmgr.exe" |
| CALL 00093840 |                     |
| ADD ESP,0x4   |                     |
| PUSH EAX      |                     |
| CALL 00093770 |                     |
| ADD ESP,0x4   |                     |
| PUSH 0x1      |                     |
| CALL 000923A0 |                     |
| ADD ESP,0x4   |                     |
| PUSH 0x9700C  | ASCII "cmd.exe"     |
| CALL 00093840 |                     |
| ADD ESP,0x4   |                     |
| PUSH EAX      |                     |
| CALL 00093770 |                     |
| ADD ESP,0x4   |                     |
| PUSH 0x1      |                     |
| CALL 000923A0 |                     |
| ADD ESP,0x4   |                     |
| PUSH 0x97014  | ASCII "ertrvgg.rkr" |
| CALL 00093840 |                     |
| ADD ESP,0x4   |                     |
| PUSH EAX      |                     |
| CALL 00093770 |                     |
| ADD ESP,0x4   |                     |
| PUSH 0x1      |                     |
| CALL 000923A0 |                     |
| ADD ESP,0x4   |                     |
| PUSH 0x97020  | ASCII "BylQOT.rkr"  |

Conforme va desencriptando tiene una rutina en la cual va cargando proceso por proceso hasta encontrarse con él y en caso de encontrarlo lo termina:

|          |          |                                   |
|----------|----------|-----------------------------------|
| 009BFE48 | 00093A96 | CALL to OpenProcess from 00093A93 |
| 009BFE4C | 00000001 | Access = TERMINATE                |
| 009BFE50 | 00000000 | Inheritable = FALSE               |
| 009BFE54 | 00000930 | ProcessId = 0x930                 |
| 009BFE58 | 7C8309D1 | kernel32.OpenProcess              |
| 009BFE5C | 009BFFA8 |                                   |
| 009BFE60 | 000937E8 | RETURN to 000937E8 from 00093A70  |
| 009BFE64 | 00000001 |                                   |

|          |          |   |
|----------|----------|---|
| 009BFE50 | 00093AC1 | CALL to <b>TerminateProcess</b> from 00093ABE |
| 009BFE54 | 0000011C | hProcess = 0000011C (window)                  |
| 009BFE58 | FFFFFFFF | ExitCode = FFFFFFFF (-1.)                     |
| 009BFE5C | 7C801E1A | kernel32.TerminateProcess                     |
| 009BFE60 | 009BFFA8 |   |

.....

## 6 Palabras finales

Hasta aquí llego humildemente la primer parte de este análisis de malware, en próximos días se publicara la segunda parte en donde se incluirá a detalle las conexiones remotas, el análisis de los servidores con los resultados obtenidos con la base de datos, las conclusiones finales y la posible solución.

## 7 Agradecimientos

Debo agradecer a: Nahuel Riva (NCR+/CRC!), Christian Navarrete (Chr1x), Erick (+Erisoft)

Por leer el análisis y darme feedback para mejorar las cosas.

## 8 Contacto

Twitter: <https://twitter.com/TorresCrack248>

e-mail: [tora\\_248@hotmail.com](mailto:tora_248@hotmail.com)

Blog: <http://torrescrack.blogspot.com>