Skype (v2.5) Analysis Notes (French)

Author: Ouanilo MEDEGAN

http://www.oklabs.net

Skype : Problèmes

L’analyse du protocole skype se heurte à deux principaux problèmes :

- Un binaire implémentant de nombreuses techniques pour empêcher un audit du code, toute la plate forme Skype tenant dans ce seul fichier binaire exécutable.
- Presque aucune donnée n’est envoyée en clair sur le réseau. Utilisation très avancée de la cryptographie.

Skype : Déprotection du binaire

Le premier objectif est donc d’enlever les protections qui empêchent un audit du binaire de sorte a pouvoir dans un debugger avoir accès aux fonctions de cryptage ainsi qu’aux données destinées à être cryptées pour ensuite etre envoyé sur le réseau. Ainsi nous pourrons donc avoir une vue claire dudit protocole.

Skype : Déprotection du binaire

Protections :

- Le code et les données sensibles sont cryptés.
- Les debuggers sont détectés.
- Quelques centaines d'agents présents dans le code veillent à l'intégrité du code en mémoire.
- L'intégrité du binaire sur le disque dur est vérifiée pour parer a tout « patch ».
- Le code proprement dit est assombri par une politique de code difficile a lire.

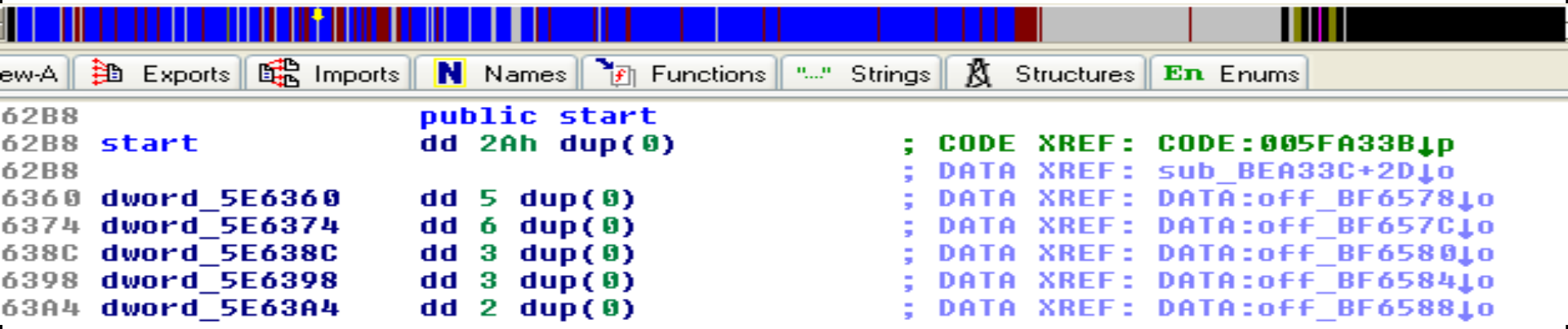
Skype : Déprotection du binaire (Code crypté)

Environ 4Mo du binaire sont encryptés. Ouvert dans un désassembleur (IDA), une bonne partie du code (le code sensible) est interprétée comme étant des données. A son exécution Skype va décrypté cette zone cryptée la rendant ainsi exécutable et exploitable. Cela empêche tout audit statique du code sensible, celui n'étant compréhensible qu'après décryptage par Skype.

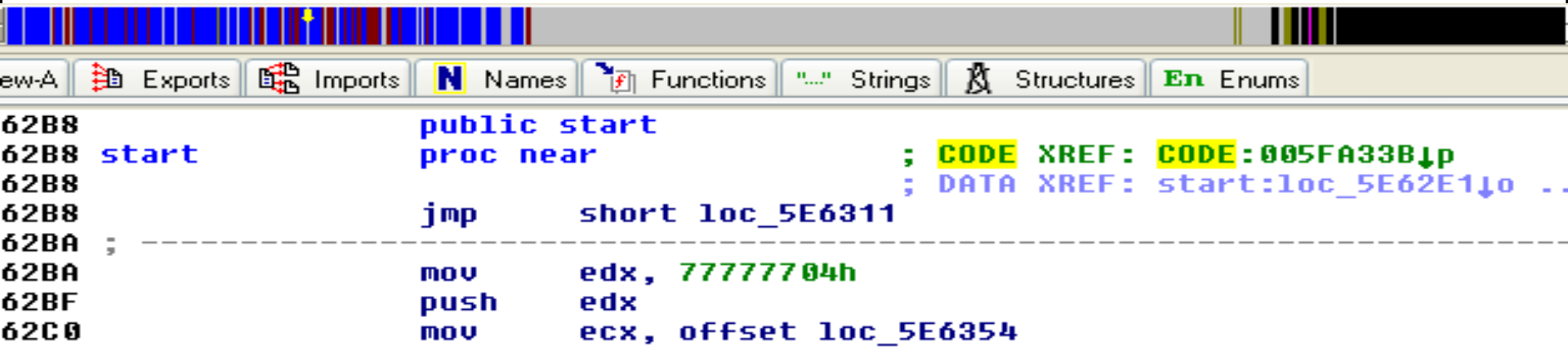
En plus de cela Skype va effacer les 0xF4 (244) premiers octets de son code pour empêcher de pouvoir obtenir un binaire exécutable en « dumpant » la mémoire du process après le décryptage.

Skype : Déprotection du binaire (Code crypté)

Skype Décrypté en Mémoire



Skype Original



Skype : Déprotection du binaire (Code crypté)

```
00BEA378  .  8D45 FC        LEA EAX,[LOCAL.1]
00BEA37B  .  50             PUSH EAX
00BEA37C  .  6A 40          PUSH 40
00BEA37E  .  8B45 D0        MOV EAX,[LOCAL.12]
00BEA381  .  50             PUSH EAX
00BEA382  .  8B45 D4        MOV EAX,[LOCAL.11]
00BEA385  .  50             PUSH EAX
00BEA386  .  E8 E9E081FF    CALL <JMP.&kernel32.VirtualProtect>
00BEA38B  .  85C0           TEST EAX,EAX
00BEA38D  .v 75 0A          JNZ SHORT Skype.00BEA399
00BEA38F  .  B8 08A7BE00    MOV EAX,Skype.00BEA708
00BEA394  .  E8 2BFAFFFF    CALL <Skype.@TCustomDBGrid@CreateWnd>
00BEA399  >  8B45 D4        MOV EAX,[LOCAL.11]
00BEA39C  .  33C9           XOR ECX,ECX
00BEA39E  .  8B55 D0        MOV EDX,[LOCAL.12]
00BEA3A1  .  E8 F29581FF    CALL <Skype.@@FillChar>
00BEA3A6  .  8D45 FC        LEA EAX,[LOCAL.1]
00BEA3A9  .  50             PUSH EAX
00BEA3AA  .  6A 20          PUSH 20
00BEA3AC  .  8B45 D0        MOV EAX,[LOCAL.12]
00BEA3AF  .  50             PUSH EAX
00BEA3B0  .  8B45 D4        MOV EAX,[LOCAL.11]
00BEA3B3  .  50             PUSH EAX
00BEA3B4  .  E8 BBE081FF    CALL <JMP.&kernel32.VirtualProtect>
00BEA3B9  .  85C0           TEST EAX,EAX
00BEA3BB  .v 75 0A          JNZ SHORT Skype.00BEA3C7
00BEA3BD  .  B8 08A7BE00    MOV EAX,Skype.00BEA708
00BEA3C2  .  E8 FDF9FFFF    CALL <Skype.@TCustomDBGrid@CreateWnd>
00BEA3C7  >  C605 DC9ABF0   MOV BYTE PTR DS:[BF9ADC],1
00BEA3CE  .  6A 04          PUSH 4
00BEA3D0  .  68 00100000    PUSH 1000
00BEA3D5  .  A1 309BBF00    MOV EAX,DWORD PTR DS:[BF9B30]
00BEA3DA  .  50             PUSH EAX
00BEA3DB  .  6A 00          PUSH 0
00BEA3DD  .  E8 82E081FF    CALL <JMP.&kernel32.VirtualAlloc>
```

```
pOldProtect
NewProtect = PAGE_EXECUTE_READWRITE
Size
Address
VirtualProtect

ASCII "0ut of memory"

pOldProtect
NewProtect = PAGE_EXECUTE_READ
Size
Address
VirtualProtect

ASCII "0ut of memory"

Protect = PAGE_READWRITE
AllocationType = MEM_COMMIT
Size => 49F000 (4845568.)
Address = NULL
VirtualAlloc
```

```
00BEA43C  .  C745 EC  0FF0   MOV [LOCAL.5],7077F00F
00BEA443  .  8B45 F8         MOV EAX,[LOCAL.2]
00BEA446  .  8D0480          LEA EAX,DWORD PTR DS:[EAX+EAX*4]
00BEA449  .  8B0485 EC9ABF   MOV EAX,DWORD PTR DS:[EAX*4+BF9AEC]
00BEA450  .  C1E8 02         SHR EAX,2
00BEA453  .  48              DEC EAX
00BEA454  .  85C0            TEST EAX,EAX
00BEA456  .v 72 3B           JB SHORT Skype.00BEA493
00BEA458  .  40              INC EAX
00BEA459  .  8945 C8         MOV [LOCAL.14],EAX
00BEA45C  .  C745 E8 0000    MOV [LOCAL.6],0
00BEA463  >  8B45 F4         MOV EAX,[LOCAL.3]
00BEA466  .  8B55 E8         MOV EDX,[LOCAL.6]
00BEA469  .  8B0490          MOV EAX,DWORD PTR DS:[EAX+EDX*4]
00BEA46C  .  3345 EC         XOR EAX,[LOCAL.5]
00BEA46F  .  8B55 F0         MOV EDX,[LOCAL.4]
00BEA472  .  8B4D E8         MOV ECX,[LOCAL.6]
00BEA475  .  890488          MOV DWORD PTR DS:[EDX+ECX*4],EAX
```

```
00BEA635  >  8B55 CC        MOV EDX,[LOCAL.13]
00BEA638  .  A1 681EC000    MOV EAX,DWORD PTR DS:[C01E68]
00BEA63D  .  8B0D 309BBF0   MOV ECX,DWORD PTR DS:[BF9B30]
00BEA643  .  E8 948A81FF    CALL <OLDSkype.@Move>
00BEA648  .  68 00800000    PUSH 8000
00BEA64D  .  6A 00          PUSH 0
00BEA64F  .  A1 681EC000    MOV EAX,DWORD PTR DS:[C01E68]
00BEA654  .  50             PUSH EAX
00BEA655  .  E8 12DE81FF    CALL <JMP.&kernel32.VirtualFree>
```

```
OLDSkype.0049F000

FreeType = MEM_RELEASE
Size = 0
Address => NULL
VirtualFree
```

Skype : Déprotection du binaire (Code crypté)

```
Buffer = malloc(0x49F000);
ZeroMemory(Buffer, 0x49F000);
ZeroMemory(&si, sizeof(si));
si.cb = sizeof(si);
hProcess = OpenProcess(PROCESS_VM_READ,FALSE, 0xF0C);
ReadProcessMemory(hProcess, (LPVOID)0x01FB0000, Buffer, 0x49F000,NULL);
result = fopen("Skype.exe", "wb");
memcpy(BigBuffer + 3429792, Buffer, 0x49F000);
fwrite(BigBuffer, Size, 1, result);
fclose(result);
```

```
00BEA635  >  8B55 CC       MOV EDX,[LOCAL.13]
00BEA638  .  A1 681EC000   MOV EAX,DWORD PTR DS:[C01E68]
00BEA63D  .  8B0D 309BBF0( MOV ECX,DWORD PTR DS:[BF9B30]      Skype.0049F000
00BEA643  .  90            NOP
00BEA644  .  90            NOP
00BEA645  .  90            NOP
00BEA646  .  90            NOP
00BEA647  .  90            NOP
00BEA648  .  68 00800000   PUSH 8000                          FreeType = MEM_RELEASE
00BEA64D  .  6A 00         PUSH 0                             Size = 0
00BEA64F  .  A1 681EC000   MOV EAX,DWORD PTR DS:[C01E68]
00BEA654  .  50            PUSH EAX                           Address => NULL
00BEA655  .  E8 12DE81FF   CALL <JMP.&kernel32.VirtualFree>   VirtualFree
```

Skype : Déprotection du binaire (Anti-Debuggers)

Skype détecte si il est debugger en utilisant un méthode plus ou moins connue mais pas très efficace. Cette méthode consiste à essaye via un CreateFile de charger les drivers de SoftIce et Cie. Si cela réussi l'exécution est arretée avec un message d'alerte.

Une autre technique est utilisée. Voir plus loin.

```
005E6311  > E8 26010000  CALL Skype.005E643C
005E6316  .  84C0        TEST AL,AL
005E6318  .v 74 1C       JE SHORT Skype.005E6336
005E631A  .  6A 00       PUSH 0                                  ; Style = MB_OK|MB_APPLMODAL
005E631C  .  FF35 7065BF0 PUSH DWORD PTR DS:[BF6570]             ; Title = "Skype"
005E6322  .  FF35 7465BF0 PUSH DWORD PTR DS:[BF6574]             ; Text = "Error: Skype is not compatible with debuggers like SoftICE .."
005E6328  .  6A 00       PUSH 0                                  ; hOwner = NULL
005E632A  .  E8 252AE2FF CALL <JMP.&user32.MessageBoxA>          ; MessageBoxA
005E632F  .  6A 01       PUSH 1                                  ; ExitCode = 1
005E6331  .  E8 D61DE2FF CALL <JMP.&kernel32.ExitProcess>        ; ExitProcess
```

Skype : Déprotection du binaire (Agents d'intégrités)

La protection la plus efficace dans Skype. En effet, des portions de codes (similaires) sont dispersées dans tout le binaire. Nombre aujourd'hui recensé : 223 !

Chacun de ses bouts de code assure l'intégrité du code en mémoire, en calculant des « checksums » de portions du code. Ces checksums serviront à déterminer des adresses dans le code ou l'exécution sera redirigée, ou des adresses de données primordiales pour l'exécution du code.

Aucun de ses agents ne ressemble a un autre ni part la taille, ni par les instructions d'initialisation, ni par le stockage du résultat final etc..

Skype : Déprotection du binaire (Agents d'intégrités)

La protection la plus efficace dans Skype. En effet, des portions de codes (similaires) sont dispersées dans tout le binaire. Nombre aujourd'hui recensé : 223 !

Chacun de ses bouts de code assure l'intégrité du code en mémoire, en calculant des « checksums » de portions du code. Ces checksums serviront à déterminer des adresses dans le code ou l'exécution sera redirigée, ou des adresses de données primordiales pour l'exécution du code.

Aucun de ses agents ne ressemble a un autre ni part la taille, ni par les instructions d'initialisation, ni par le stockage du résultat final etc..

Skype : Déprotection du binaire (Agents d'intégrités)

Skype : Déprotection du binaire (Agents d'intégrités)

Grace au moteur de recherche de séquences de commande imprécises de OllyDbg on va grace a leurs empreintes communes retrouver chacun de ses agents.

```
MOV R32, CONST
ANY 8
MOV R32, R32
ANY 8
MOV R32, [R32+CONST]
JMP OFFSET
```

```
XOR R32, CONST
ANY 8
ANY 8
ANY 8
MOV R32, [R32+CONST]
JMP OFFSET
```

```
XOR R32, R32
ANY 8
MOV R32, CONST
ANY 8
DEC R32
JNZ OFFSET
```

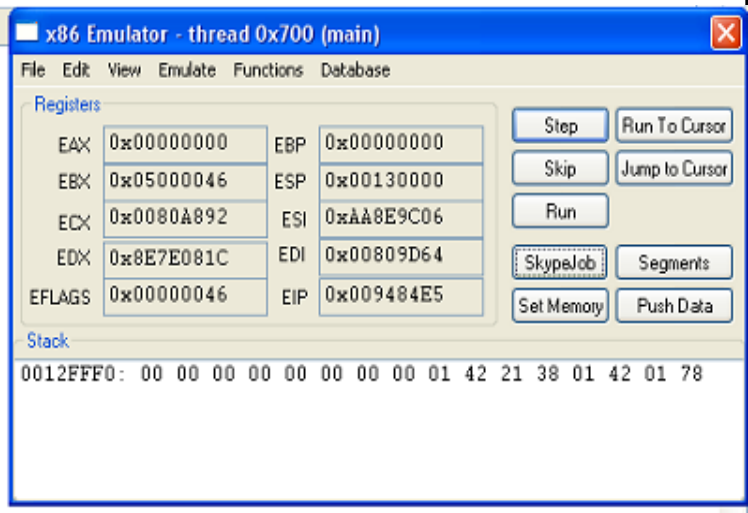
```
XOR R32, R32
ANY 8
MOV R32, [R32+CONST]
ANY 8
DEC R32
JNZ OFFSET
```

Skype : Déprotection du binaire (Agents d'intégrités)

Ensuite on va modifier un émulateur x86 (x86emu) sous forme de plugin IDA qui a bout utilité de pouvoir émulé toute instruction x86 indépendamment du système.
Donc on le modifie de sorte qu'il se positionne sur chacun des agents repérés, qu'il calcule pour chaque agent le checksum attendu, la fin de l'agent ainsi que le registre où sera stocké le checksum pour cet agent.
Celui va nous générer un tableau de structures en c, où chaque structure défini le cœur de l'agent sa fin et une instruction assembleur qui place dans le registre destiné a accueillir la valeur du checksum, cette valeur finale.

Skype : Déprotection du binaire (Agents d'intégrités)

```
005484B4    009484B4: sub_948310+1A4
end at 0x8ef8a1 (checksum = 0x25480F72, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x8efd29
valuer found at 0x8efd4a : xor, register_2
end at 0x8efd55 (checksum = 0xA4C2184C, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x8f01b6
valuer found at 0x8f01d8 : sub, register_2
end at 0x8f01e3 (checksum = 0x7442D802, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x8f0b49
valuer found at 0x8f0b70 : add, register_2
end at 0x8f0b7b (checksum = 0x3A326175, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x8f11a1
valuer found at 0x8f11bb : add, register_3
end at 0x8f11c8 (checksum = 0x4A8FFB13, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x905450
valuer found at 0x905471 : add, register_7
end at 0x90547e (checksum = 0x9C89E0AF, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x924cde
valuer found at 0x924d02 : xor, register_6
end at 0x924d0f (checksum = 0x92E232D3, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x928aec
valuer found at 0x928b16 : sub, register_3
end at 0x928b21 (checksum = 0x7A13A572, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x928e78
valuer found at 0x928e99 : xor, register_2
end at 0x928ea6 (checksum = 0xC026EBB8, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x93ae8e
valuer found at 0x93aeac : xor, register_7
end at 0x93aeb9 (checksum = 0x1131E7ED, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x93d1e0
valuer found at 0x93d204 : add, register_3
end at 0x93d20f (checksum = 0x64A2D76A, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x93f37e
valuer found at 0x93f3a1 : add, register_6
end at 0x93f3ae (checksum = 0xBBE3E8C3, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x940266
valuer found at 0x940287 : sub, register_6
end at 0x940294 (checksum = 0xA22EB245, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x940577
valuer found at 0x940591 : xor, register_3
end at 0x94059d (checksum = 0x2D33939C, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x940a3c
valuer found at 0x940a5f : xor, register_0
end at 0x940a6c (checksum = 0x456DE1A1, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x940ba3
valuer found at 0x940bc1 : sub, register_6
end at 0x940bce (checksum = 0xDEFFE75A, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x946a4d
valuer found at 0x946a67 : sub, register_6
end at 0x946a72 (checksum = 0xAA8E9C06, wasjnz = 1, CF = 64, tracereg = -1)
begin at 0x9484b4
valuer found at 0x9484da : sub, register_2
end at 0x9484e5 (checksum = 0x8E7E081C, wasjnz = 1, CF = 64, tracereg = -1)
work done for 223 snipets
```



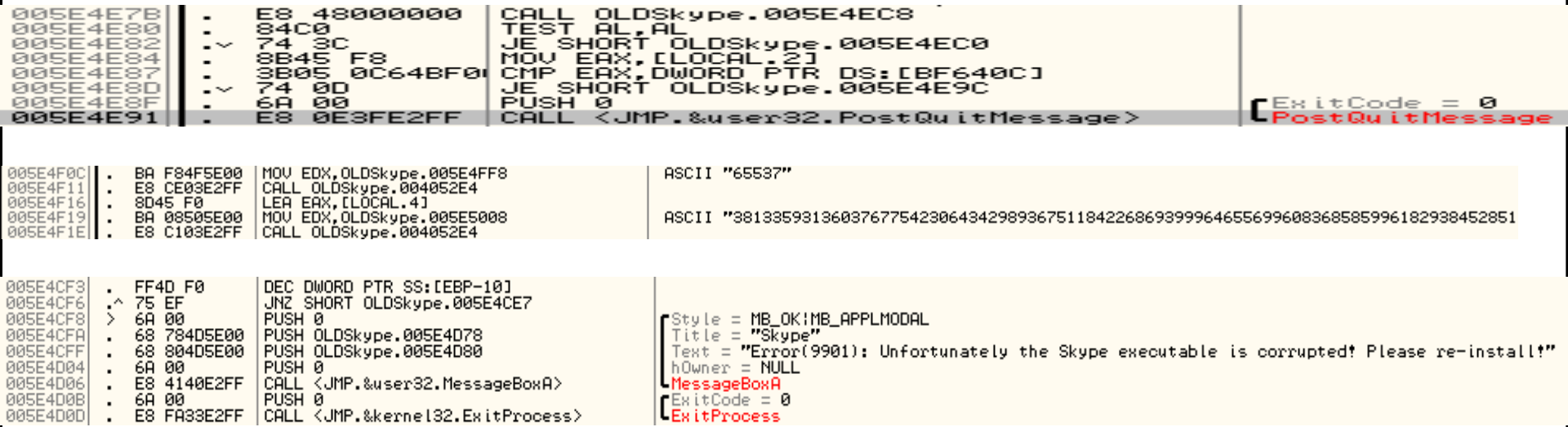
Skype : Déprotection du binaire (Agents d'intégrités)

Une fois ce tableau générée il nous sert à ecrire un plugin pour OllyDbg qui va utiliser les données de chacune de ces structures là pour générer un « patch » pour chacun de ses agents.

```
008986F3  .  BB 723E0000    MOV EBX,3E72                Chk #63
008986F8  .  81EB 59C784FF  SUB EBX,FF84C759
008986FE  .  BA 1B940D00    MOV EDX,0D941B
00898703  .  83C3 00        ADD EBX,0
00898706  .  81F2 F260890(  XOR EDX,8960F2
0089870C  .  8BF2           MOV ESI,EDX
0089870E  .  81C6 5E0B7BFF  ADD ESI,FF7B0B5E
00898714  .  C7C2 0B716C37  MOV EDX,376C710B            Patch ICI
0089871A  .  90             NOP                         Plus de Calcul de Checksum
0089871B  .  90             NOP                         Execution plus rapide :)
0089871C  .  90             NOP
0089871D  .  90             NOP
0089871E  .  90             NOP
0089871F  .  90             NOP
00898720  .  90             NOP
00898721  .  90             NOP
00898722  .  90             NOP
00898723  .  90             NOP
00898724  .  81EA 0B716C37  SUB EDX,376C710B
```

Skype : Déprotection du binaire (Intégrité du binaire)

Certaines procédures vérifient si l'image du binaire sur le disque dur a changé en utilisant une clé publique et détecte aussi si un debuggage est en cours. L'exécution est arrêtée plus ou moins subtilement, les cas échéants.



Skype : Conclusion

A cette étape, l’analyse du protocole proprement dit peut à priori commencer car avec les modifications sur le binaire on peut mener a termes toute sorte de scénario. Patch, breakpoints, Appels avec debugger, Chats..

L’analyse du protocole commence par la lecture des documents déjà publiés a ce sujet.

Skype REGISTER – Première Partie

- La phase de REGISTER de Skype se décompose en deux parties :

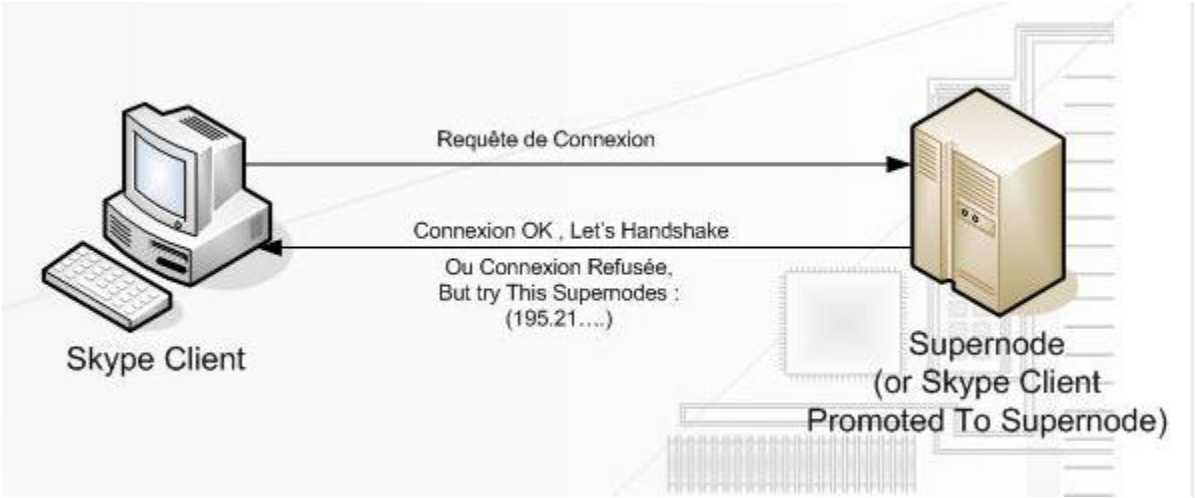
1 – Recherche d’un supernode qui va manager notre session.

2 – Authentification

Skype REGISTER – Première Partie : Procédure

- La recherche d'un supernode manager se fait par UDP
- Le client dispose d'un Host Cache « ..\shared.xml » qui est une liste de supernodes qu'il a mis à jour avec le temps
- La toute première fois, le Host Cache est vide, le client va donc utiliser une liste de supernodes HardCodés (195.215.8.145:33033 / 64.246.48.23:33033 ...)

Skype REGISTER – Première Partie : Procédure



Skype REGISTER – Première Partie : Procédure

- Le supernode peut soit accepté de managé le client, soit refusé et lui envoyé une liste de supernodes à tester.
- La liste d'hôtes va servir à mettre à jour le Host Cache.
- L'entité « HostScanner » du client va continuer à envoyé des « probes » à tous les hôtes connus ou reçus jusqu'à ce qu'un supernode accepte la connection et établisse un handshake.

Skype REGISTER – Première Partie : Procédure

73		18:11:31	Skype Debug	0x7c80996d [kernel32.dll]	Skype.exe [pid=2716, tid=3768]	HostScanner: sending probe (addr=75.80.133.206:58760 time=2.0s)
74		18:11:31	Skype Debug	0x00a3916e	Skype.exe [pid=2716, tid=3768]	CommLayer: Sending packet #e414 to 75.80.133.206 using UDP
75		18:11:31	sendto	0x008da404	Skype.exe [pid=2716, tid=3768]	0.0.0.0:50024: Sent 35 bytes to 75.80.133.206:58760
76		18:11:32	recvfrom	0x008da528	Skype.exe [pid=2716, tid=3768]	0.0.0.0:50024: Received 18 bytes from 75.80.133.206:58760
77		18:11:32	Skype Debug	0x7c80e3dc [kernel32.dll]	Skype.exe [pid=2716, tid=3768]	CommLayer: Packet #63f0 received from 75.80.133.206 using UDP
78		18:11:32	Skype Debug	0x7c80e3dc [kernel32.dll]	Skype.exe [pid=2716, tid=3768]	CommLayer: Reply to #e413 cmd $28
79		18:11:32	Skype Debug	0x00a5dd2b	Skype.exe [pid=2716, tid=3768]	HostScanner: probe accept received (addr=75.80.133.206:58760 time=2.0s)
80		18:11:32	Skype Debug	0x7c80e3dc [kernel32.dll]	Skype.exe [pid=2716, tid=3768]	CommLayer: Deleting packet #e414
81		18:11:32	Skype Debug	0x7c8024b7 [kernel32.dll]	Skype.exe [pid=2716, tid=3768]	HostScanner: fallback+proxy-connecting (addr=75.80.133.206:58760 time=2.2s)
82		18:11:32	Skype Debug	0x00a39260	Skype.exe [pid=2716, tid=3768]	TCP: OUT #1 75.80.133.206:58760 State CONNECTING
83		18:11:32	Skype Debug	0x7c920732 [ntdll.dll]	Skype.exe [pid=2716, tid=3768]	TCP: OUT #1 75.80.133.206:58760 connecting (timeout=300000 now=24585562)
84		18:11:32	Skype Debug	0x7c920732 [ntdll.dll]	Skype.exe [pid=2716, tid=3768]	FallbackConnManager: connecting to 75.80.133.206:58760
85		18:11:32	Skype Debug	0x00a342f3	Skype.exe [pid=2716, tid=3768]	FallbackConn: 75.80.133.206:58760/direct connecting to 75.80.133.206:58760...
86		18:11:32	connect	0x008d9747	Skype.exe [pid=2716, tid=3768]	124.251.195.2:49922: connecting to 75.80.133.206:58760
87		18:11:33	Skype Debug	0x7c80b9e6 [kernel32.dll]	Skype.exe [pid=2716, tid=3768]	FallbackConn: 75.80.133.206:58760/https connecting to 75.80.133.206:443...
88		18:11:33	connect	0x008d9747	Skype.exe [pid=2716, tid=3768]	156.251.195.2:49922: connecting to 75.80.133.206:443
89		18:11:33	Skype Debug	0x0076b561	Skype.exe [pid=2716, tid=3768]	FallbackConn: 75.80.133.206:58760/https sending https handshake
90		18:11:33	send	0x008d9aa4	Skype.exe [pid=2716, tid=3768]	127.0.0.1:1508: Sent 72 bytes to 75.80.133.206:443
91		18:11:33	recv	0x008d9b48	Skype.exe [pid=2716, tid=3768]	127.0.0.1:1508: Received 134 bytes from 75.80.133.206:443
92		18:11:33	Skype Debug	0x7c80ee67 [kernel32.dll]	Skype.exe [pid=2716, tid=3768]	FallbackConn: 75.80.133.206:58760/https connected, handing over
93		18:11:33	Skype Debug	0x7c8025f0 [kernel32.dll]	Skype.exe [pid=2716, tid=3768]	TCP: OUT #1 75.80.133.206:58760 connected -> 75.80.133.206:443
94		18:11:33	Skype Debug	0x00444554	Skype.exe [pid=2716, tid=3768]	TCP: OUT #1 75.80.133.206:58760 State CONNECTED
95		18:11:33	closesocket	0x008d94fd	Skype.exe [pid=2716, tid=3768]	0.0.0.0:1507: connection to 75.80.133.206:58760 closed
96		18:11:33	send	0x008d9aa4	Skype.exe [pid=2716, tid=3768]	127.0.0.1:1508: Sent 73 bytes to 75.80.133.206:443
97		18:11:34	Skype Debug	0x0085e2b6	Skype.exe [pid=2716, tid=3768]	CommLayer: Sending packet #e416 to 75.80.133.206 using TCP
98		18:11:34	send	0x008d9aa4	Skype.exe [pid=2716, tid=3768]	127.0.0.1:1508: Sent 27 bytes to 75.80.133.206:443
99		18:11:34	recv	0x008d9b48	Skype.exe [pid=2716, tid=3768]	127.0.0.1:1508: Received 201 bytes from 75.80.133.206:443
100		18:11:34	Skype Debug	0x00a39260	Skype.exe [pid=2716, tid=3768]	CommLayer: Packet #6456 received from 75.80.133.206 using TCP
101		18:11:34	Skype Debug	0x00a39260	Skype.exe [pid=2716, tid=3768]	CommLayer: Reply to #e415 cmd $31
102		18:11:34	Skype Debug	0x0075ef38	Skype.exe [pid=2716, tid=3768]	HostScanner: client accept received (addr=75.80.133.206:58760 time=4.0s)
103		18:11:34	Skype Debug	0x0075ef38	Skype.exe [pid=2716, tid=3768]	HostScanner: stackVersion=34 stackTimestamp=608211059 port=58760 (addr=75.80.133.206:58760 time=4.0s)

Skype REGISTER – Première Partie : Transport des données

- Durant toute la communication, les charges utiles des paquets Skype sont cryptés avec un RC4

Skype REGISTER – Première Partie : Transport des données

Transaction ID : Random (2 Octets)

Packet Type : (1 Octet)

- 0xX2 : Paquet chiffré
- 0xX3 : Renvoi avec nouveau chiffrement
- 0xX7 : NACK

Vecteur d'initialisation : (4 Octets)

Généré aléatoirement (Random TRES costaud, basé sur SHA, GetTickCount, QueryPerformanceCounter, GetPointer, etc..) puis mis à jour.

CRC32 : (4 Octets)

CRC32 (salt) des données non chiffrées

Skype REGISTER – Première Partie : Transport des données

- Les données sont chiffrées avec RC4. La clé du RC4 est générée à partir d'un « seed » via une fonction Seed2RC4Key

- SEED = 

Skype REGISTER – Première Partie : Transport des données

- A l'envoi du premier paquet, le client ne connait pas son IP Publique. Il utilise alors « 0.0.0.0 » pour générer son seed.
- Le supernode lui n'arrivera pas à décrypter le paquet et renvoie donc un paquet de type NACK contenant l'ip publique du client.
- Le client met donc à jour cette donnée pour la génération d'un seed plus cohérent.

Skype REGISTER – Première Partie : Transport des données

42		18:11:29	sendto	0x008da404	Skype.exe [pid=2716, tid=3768]	0.0.0.0:50024: Sent 21 bytes to 195.215.8.145:33033
43		18:11:29	EncryptMessage	0x77e8e4e6 [RPCRT4.dll]	Skype.exe [pid=2716, tid=3224]	
44		18:11:29	DecryptMessage	0x77e8e61f [RPCRT4.dll]	Skype.exe [pid=2716, tid=3224]	
45		18:11:29	EncryptMessage	0x77e8e4e6 [RPCRT4.dll]	Skype.exe [pid=2716, tid=3224]	
46		18:11:29	DecryptMessage	0x77e8e61f [RPCRT4.dll]	Skype.exe [pid=2716, tid=3224]	
47		18:11:29	Skype Debug	0x7c91fb71 [ntdll.dll]	Skype.exe [pid=2716, tid=3224]	ProxyDetect: Automatic proxy detect failed (detect disabled 0)
49		18:11:29	Skype Debug	0x71993ed8 [mswsock.dll]	Skype.exe [pid=2716, tid=3768]	FallbackConnManager: proxy resolved to https 0.0.0.0:0 socks 0.0.0.0:0
51		18:11:29	recvfrom	0x008da528	Skype.exe [pid=2716, tid=3768]	0.0.0.0:50024: Received 11 bytes from 195.215.8.145:33033
52		18:11:29	Skype Debug	0x7c9206f0 [ntdll.dll]	Skype.exe [pid=2716, tid=3768]	UDP: NACK #e40a resending (00000000->3e2709fb)
53		18:11:29	sendto	0x008da404	Skype.exe [pid=2716, tid=3768]	0.0.0.0:50024: Sent 26 bytes to 195.215.8.145:33033
54		18:11:29	recvfrom	0x008da528	Skype.exe [pid=2716, tid=3768]	0.0.0.0:50024: Received 53 bytes from 195.215.8.145:33033
55		18:11:29	Skype Debug	0x719f951e [WS2_32.dll]	Skype.exe [pid=2716, tid=3768]	CommLayer: Packet #cd1a received from 195.215.8.145 using UDP

```
...: #42
000: e4 0a 02 be 66 df 5f ec 03 1c 9e 8d 34 75 c6 64  ....f._.....4u.d
010: 09 c6 36 a5 eb                                   ..6..

...: #51
000: e4 0a 37 3e 27 09 fb cf f6 dd 5f                 ..7>'....._

...: #53
000: e4 0a 43 01 cf f6 dd 5f c3 d7 08 91 ec 03 1c 9e  ..C...._........
010: 73 c3 7c a7 bc a4 c6 c3 c5 b8                    s.|.......
```

Skype REGISTER – Première Partie : TODO

- Reverse de l'engine Seed2RC4Key
  (Pour l'instant on injecte du code dans Skype pour obtenir une clé à partir d'un seed)

- Compréhension complète des payloads cryptés

- Un début de communication sera donc fin implémentée. On pourra passer à la phase 2 du register, l'authentification sécurisée.

Skype REGISTER – Suite de l’analyse

● Avant l’authentification, le client Skype doit d’abord se faire accepter par un supernode qui va manager la session.

Cette phase consiste en le scan du réseau à le recherche de supernodes actifs (probes), la connexion à ces supernodes et l’acceptation de « client management » par celui-ci.

Skype REGISTER

- Une fois connecté à un supernode (via un port custom, le port 443 ou le port 80), le client et le supernode procèdent à un échange de clés RC4 pour installer le système de cryptage qui prévaudra tout le temps de la session.

```
>> ....· #170
>> 0000: 5d 55 02 c8 80 99 00 00 00 01 00 00 00 03 29 03  ]U..............).
>> 0010: 6f 54 b5 62 eb a0 91 8e 27 ac 2d 7a 23 78 89 26  oT.b....'.-z#x.&
>> 0020: df 04 a5                                         ...

<< ....· #171
<< 0000: ca 64 d6 4a 83 b3 00 00 00 01 00 00 00 03 6f 03  .d.J..........o.
<< 0010: 99 f6 6f 54 b5 62 eb a0 91 8e 27 ac 2d 7a 23 78  ..oT.b....'.-z#x
<< 0020: 89 26 df 04 a5 92 5b 50 81 be 97 5c 1d aa 93 28  .&....[P...\...(
<< 0030: 79 56 4f b4 95 c2 cb 00 71 ee 07 0c 0d da 03 d8  yVO.....q.......
<< 0040: 69 86 bf 64 85 f2                               i..d..
```

Skype REGISTER

- Si la connection s'effectue par le port 443, le client et le supernode procèdent d'abord a un Handshake HTTPS.

```
>> ....· #165
>> 0000: 80 46 01 03 01 00 2d 00 00 00 10 00 00 05 00 00   .F....-.........
>> 0010: 04 00 00 0a 00 00 09 00 00 64 00 00 62 00 00 08   .........d..b...
>> 0020: 00 00 03 00 00 06 01 00 80 07 00 c0 03 00 80 06   ................
>> 0030: 00 40 02 00 80 04 00 80 a8 f9 d6 cf 34 15 42 4b   .@..........4.BK
>> 0040: 80 f1 6e 87 8c 8d 5a 83                           ..n...Z.

<< ....· #166
<< 0000: 16 03 01 00 4a 02 00 00 46 03 01 40 1b e4 86 02   ....J...F..@....
<< 0010: ad e0 29 e1 77 74 e5 44 b9 c9 9c b4 31 31 5e 02   ..).wt.D....11^.
<< 0020: dd 77 9d 15 4a 96 09 ba 5d a8 70 20 1c a0 e4 f6   .w..J...].p.....
<< 0030: 4c 63 51 ae 2f 8e 4e e1 e6 76 6a 0a 88 d5 d8 c5   LcQ./.N..vj.....
<< 0040: 5c ae 98 c5 e4 81 f2 2a 69 bf 90 58 00 05 00      \.......*i..X...
```

Skype REGISTER

- Après l'echange de clés le client envoie au supernode une commande « Client Accept » pour se faire enregistrer par le supernode. Si le client est accepté le supernode lui envoie avec l'accusé d'acceptation, des stats du reseaux.

```
>> ....- #174
>> 0000: 34 f0 f4 15 f2 01 f0 f3 42 54 0f 3f d0 67 8e d4  4.......BT.?.g..
>> 0010: 98 f6 c7 8f 5f a8 35 d4 01 4e 28                 ...._.5..N(

<< ....- #175
<< 0000: 38 22 40 17 fb 01 f0 f3 42 b1 25 75 95 77 98 12  8"@.....B.%u.w..
<< 0010: 18 7c 10 0d c7 bf 88 fb 09 27 3e 0f 74 58 22 41  .|.......'>.tX"A
<< 0020: 07 59 42 b0 9f 30 2b ee a1 1e f9 02 42 f6 43 86  .YB..0+.....B.C.
<< 0030: 84 3d 87 e5 14 2d eb 3b d9 d0 4b 43 cd 66 27 5f  .=...-.;..KC.f'_
<< 0040: 0c 34 b3 b8 c4 17 84 86 88 fd                    .4........
```

```
CommLayer: Packet #2240 received from 24.226.94.170 using TCP
CommLayer: Reply to #f0f3 cmd $31
HostScanner: client accept received (addr=24.226.94.170:1804 time=8.2s)
HostScanner: stackVersion=37 stackTimestamp=612181303 port=1804 (addr=24.226.94.170:1804 time=8.2s)
CommLayer: Deleting packet #f0f4
CommLayer: Packet #2241 received from 24.226.94.170 using TCP
CommLayer: cmd $11
Localnode: CommandReceived(cmd=$11) from 24.226.94.170:1804
Localnode: network stats received (7560409 clients online)
```

Skype REGISTER

- Le processus de register est implémenté jusqu'à l'échange des clés. Le client accepte est en cours.
- L'implémentation se heurte à un problème : Encodage de la liste des objets passés en paramètres au fonctions.
- Exemple : Les 4 paramètres de la fonction « Client Accept » (NodeID, UpTime..) peuvent etre spécifiés de deux manières différentes.

-Facon « Etendue » : spécifiée par l'octet 0x42 en début de liste

-Facon « Basique » : spécifiée par l'octet 0x41 en début de liste

Skype REGISTER

```
>> .,.,. #174
>> 0000: 34 f0 f4 15 f2 01 f0 f3 42 54 0f 3f d0 67 8e d4  4.......BT.?.g..
>> 0010: 98 f6 c7 8f 5f a8 35 d4 01 4e 28                 ...._.5..N(
```

```
>> .,.,. #174
>> 0000: 3c c0 e9 19 f2 01 c0 e8 41 04 01 0d 28 4e 01 d4  <.......A...(N..
>> 0010: 35 a8 5f 8f 00 10 e8 86 03 00 01 1a 00 23 00     5._..........#.
```

Skype REGISTER

Phase suivante de l’implémentation :

-Utiliser le système basique et finir le client accept

-Décoder les stats du réseau

-Entamer la génération du « blob » d’authentification