Castle in the Skype

Fabrice DESCLAUX

serpilliere(at)rstack.org / fabrice.desclaux(at)eads.net
EADS Corporate Research Center — DCR/STI/C
SSI Lab
Suresnes, FRANCE

SSTIC, 31 mai, 2006

Motivation de l'étude

Étude Skype

- Skype n'est pas une application autorisée sur notre réseau
- Skype utilise un protocole obscurci
- Skype refuse de communiquer les détail du protocole
- Notre étude visait à comprendre le fonctionnement du protocole pour le maîtriser

EADS CCR

Pourquoi Skype paraît-il dangereux?

Au niveau réseau

Que voit l'administrateur réseau? (paranoïaque)

- Tout à l'air obscurci
- Utilisation de réseaux P2P
  - Nombre de machines connectées important
  - Identification des destinataires difficile
- Réutilisation automatique des accès aux proxy
- Trafic important même quand on ne l'utilise pas

Conclusion

$\Longrightarrow$ Impossible de distinguer un trafic normal d'une exfiltration de données (trafic chiffré sur des ports étranges, activité nocturne)

$\Longrightarrow$ Brouille les signes d'une intrusion

Comment Skype camoufle-t-il son code?

Chiffrement du code

- Le code sensible est "chiffré"
- Une procédure de déchiffrement est insérée au début du programme
- Elle déchiffre le code et passe la main

Chiffrement du code

```
mov     eax, offset base_adresse
...
add     eax, ds:ptr_code_chiffre[edx*4]
...
mov     eax, ds:ptr_code_dechiffre[eax*4]
add     eax, ds:memoire_allouee
...
mov     dword ptr [ebp-14h], 7077F00Fh
...
mov     eax, ds:taille_code[eax*4]
```

Comment Skype camoufle-t-il son code?

Structure de description des couche de chiffrement

```
struct memory_location
{
unsigned int start_alloc;
unsigned int size_alloc;
unsigned int start_file;
unsigned int size_file;
unsigned int protection_flag;
}
```

Comment Skype camoufle-t-il son code?

Description des zones

ZONE 1
```
dd 1000h
dd 250000h
dd 1000h
dd 250000h
dd 20h
```

ZONE 2
```
dd 251000h
dd 49000h
dd 251000h
dd 49000h
dd 2
```

ZONE 3
```
dd 29A000h
dd 13C000h
dd 29A000h
dd 3D000h
dd 4
```

ZONE 4
```
dd 3D6000h
dd 2000h
dd 2D7000h
dd 2000h
dd 4
```

Comment Skype camoufle-t-il son code?

Déchiffrement des zones

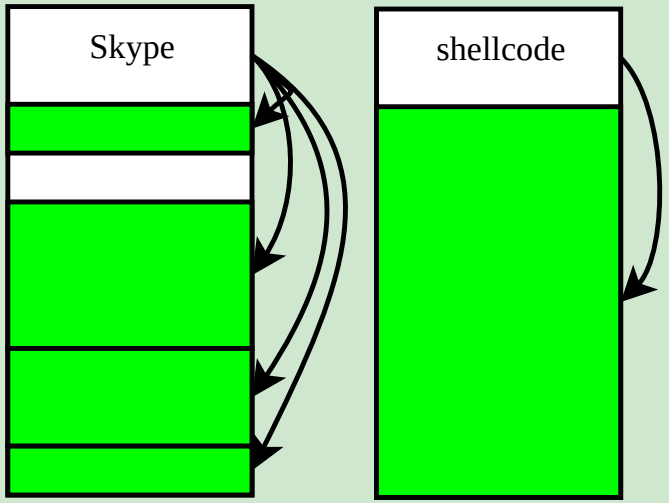
```
decipher_loop:
mov     eax, [eax+edx*4]
xor     eax, [ebp-14h]
mov     [edx+ecx*4], eax
...
mov     eax, [eax+edx*4]
xor     eax, [ebp-14h]
mov     [ebp-28h], eax
add     dword ptr [ebp-14h], 71h
inc     dword ptr [ebp-18h]
dec     dword ptr [ebp-34h]
jnz     short decipher_loop
```

Parallèle avec les malwares

Shellcode chiffré

- Le code binaire est souvent chiffré
- Une mini procédure est insérée dans le shellcode
- Elle déchiffre la charge utile dans la pile de l'application

Déchiffrement



Procédure de déchiffrement:
Chaque partie sensible sera déchiffrée en mémoire au moment de l'exécution

Partie claire

Partie chiffrée

Exemple de shellcode

Chiffrement du code

```
; Exemple de shellcode encodé
call dummy
dummy:
pop edx
sub dl, -25   ; récupération d'eip

debut_dechiffre:
xor ecx,ecx
sub cx, -0x15F ; taille du payload

boucle_dechiffrement:
    xor byte [edx], 0x99 ; décodage du payload
    inc edx
    loop boucle_dechiffrement

shellcode_chiffre:
db \xdd\xeb\xd6\xd0\xdd\xca\xb9\xda
db \xf6\xeb\xc9\xf6\xcb\xf8\xcd\xd0
db \xd6\xd7\xb9\xcb\xec\xd5\xdc\xc3
db ...
```

Effacement des traces

Ecrasement du code de déchiffrement

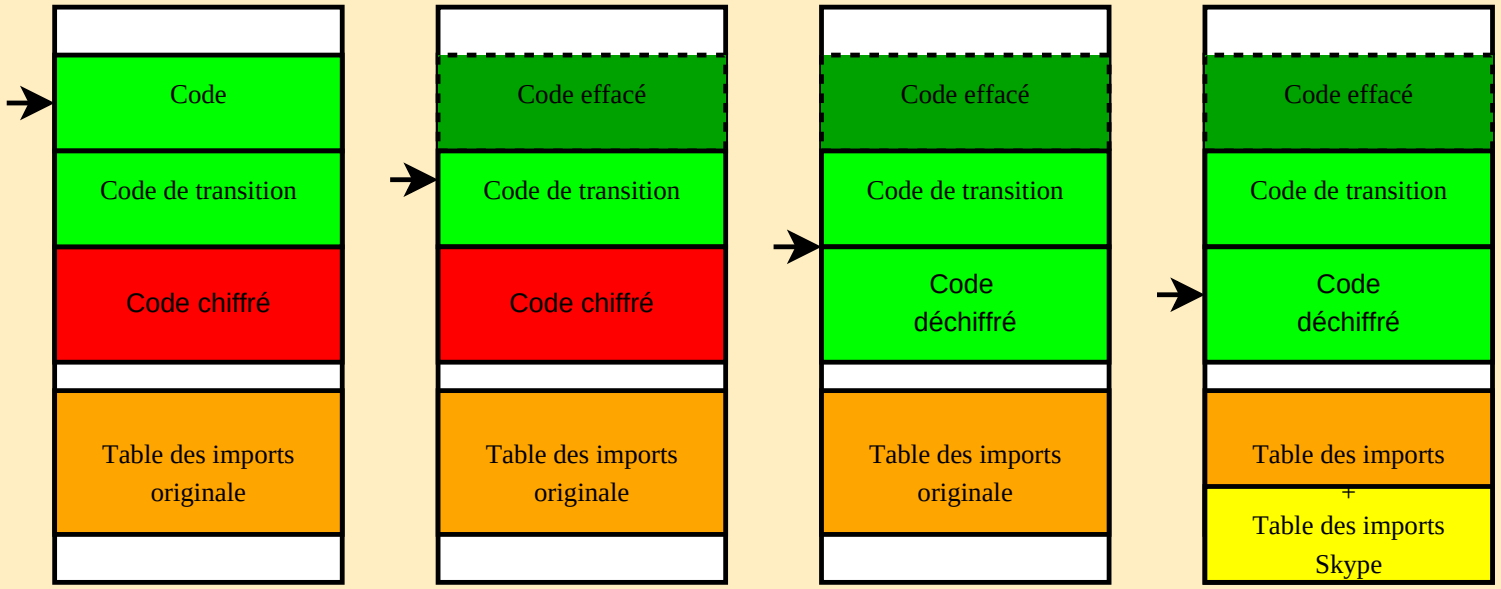
- Le code est déchiffré en mémoire
- Skype peut effacer de la mémoire le code responsable du déchiffrement
- Ici, Skype le remplace par des 0x00

Déchiffrement

```
; Exemple d'effacement de code
; (Anti dumping)
mov     edi, offset debut_zone_a_effacer
mov     ecx, offset fin_zone_a_effacer
sub     ecx, edi
xor     eax, eax ; on remplace le code par \x00
rep stosb
```

Récapitulatif

État du binaire au lancement



Imports cachés

Structure de description des imports

```
struct
{
  char* nom;
  int * ordinal;
  unsigned char* adresse;
}
```

Import de DLL, de fonction

```
dd offset aWinmm_dll     ; "WINMM.dll"
dd 0
dd 0

dd offset aWaveinreset   ; "waveInReset"
dd 0
dd 3D69D0h

Ordinal 3
dd 0
dd 3
dd 3D6A90h
```

Anti débogueur

Détection des débogueurs

Le binaire tente de détecter les débogueurs connus

- Tentative de chargement de modules de débogueurs
- Si succès, le débogueur est présent
- Sinon, on retente plus tard
- Ici, il s'agit de *Softice*

Déchiffrement

```
mov eax, offset str_Siwvid ; "\\\\.\\Siwvid"
call test_driver
test al, al
```

Anti débogueur: les malwares

Dans les malware

- Ils possèdent aussi cette fonctionnalité
- Cela ralentit la sortie d'une signature
- $\Longrightarrow$ La vie du virus est prolongée!

Worm.Win32_Bropia-N

Worm.Win32_Mytob-AR

- Ce ver détecte Ollydbg
- Et l'arrête s'il est présent

Camouflage des chaînes de caractères sensibles

Le binaire ne laisse rien transparaître

- Les chaînes de caractères sensibles sont chiffrées
- Puis déchiffrées à la volée

Chaînes de caractères chiffrées/claires

```
db 'B494A6545B414B4D',0                    \\.\SICE
db 'B49AACA6B9A3FD7C636E',0                \\.\Siwvid
db 'B49BAB5DB7BD80CA4C',0                  \\.\NTICE
db 'B49D5D8BCC4638F9666B5C5B4E5D5B',0      \\.\SiwvidSTART
```

Côté malware

Shellcode clair

```
j\x0bX\x99Rfh-c\x89\xe7h/sh\x00h/bin\x89\xe3R\xe8\n
\x00\x00\x00/bin/bash\x00WS\x89\xe1\xcd\x80
```

Shellcode chiffré

```
3\xc9\x83\xe9\xf4\xd9\xee\xd9t$\xf4[\x81s\x13\xa5\x
d1L!\x83\xeb\xfc\xe2\xf4\xcf\xda\x14\xb8\xf7\xb7$\x0
c\xc6X\xabI\x8a\xa2$!\xcd\xfe.H\xcbX\xafsM\xdbL!\xa5
\xfe.H\xcb\xfe.@\xd6\xb9Lv\xf6X\xad\xec%\xd1L!$
```

Détection par mesure de temps

Mesure de temps

- Une procédure consomme un certain temps (fixe)
- Si un attaquant trace le programme, le temps consommé est plus important
- $\Longrightarrow$ On peut détecter les débogueurs

Mesure de temps

```
call    gettickcount
mov     gettickcount_result , eax
```

Représaille

Mesure de représaille

- Tuer l'application serait facilement détectable
- Ici, l'application crée une zone de non retour
- L'état du processeur est randomisé
- Et le débogueur ne peut pas en sortir, ni remonter au code chargé de la détection

Trappe pour le débogueur

Randomisation des registres

```
pushf
pusha
mov     save_esp , esp
mov     esp , ad_alloc?
add     esp , random_value
sub     esp , 20h
popa    ; Randomize les registres
jmp     random_mapped_page
```

Trappe pour le débogueur

Contournement

- La page contenant les octets de code aléatoires peut être retrouvée
- Ses caractéristiques sont particulières: RWX
- On peut retrouver sa création, la surveiller et trouver le détecteur de débogueurs

Intégrité du binaire

Test d'intégrité du binaire

- Un attaquant peut encore modifier le binaire
- Des sommes de contrôles ont donc été rajoutées
- Chaque somme de contrôle vérifie une partie de code et une autre somme de contrôle.
- Celles-ci sont insérées au hasard dans le code et générées différemment les unes des autres

Chez les Spywares

- Gator est un spyware attaché à un shareware
- Quand on lance le shareware, celui-ci vérifie que Gator est bien lancé
- Si Gator a été supprimé, le shareware refuse de se lancer

Exemple de somme de contrôle

```
start:
    xor     edi, edi
    add     edi, 0x688E5C
    mov     eax, 0x320E83
    xor     eax, 0x1C4C4
    mov     ebx, eax
    add     ebx, 0xFFCC5AFD
loop_start:
    mov     ecx, [edi+0x10]
    jmp     lbl1
    db 0x19
lbl1:
    sub     eax, ecx
    sub     edi, 1
    dec     ebx
    jnz     loop_start
    jmp     lbl2
    db 0x73
lbl2:
    jmp     lbl3
    dd 0xC8528417, 0xD8FBBD1, 0xA36CFB2F, 0xE8D6E4B7, 0xC0B8797A
    db 0x61, 0xBD
lbl3:
    sub     eax, 0x4C49F346
```

Intégrité du binaire

Retrouver les sommes de contrôles

Même si elles ont été générées différements, elles ont des points communs

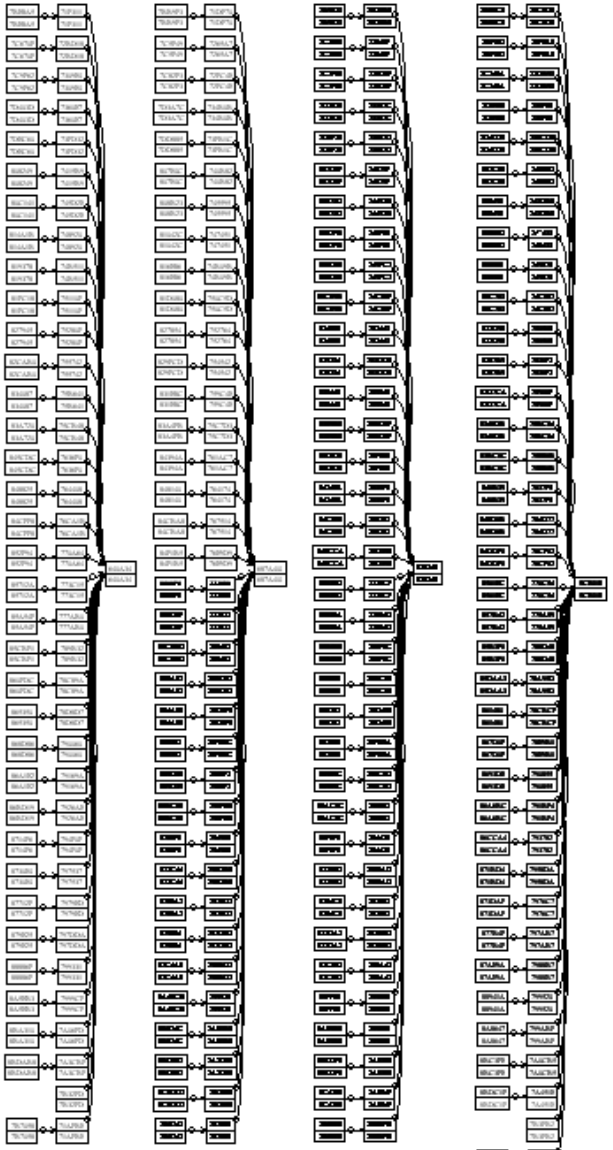
- Une initialisation de pointeur (sur du code)
- Une boucle
- Un calcul arithmétique en guise de somme

Suppression des contrôles

- On peut donc les distinguer dans le code
- Calculer leur vrai somme (grâce à un émulateur par exemple)
- Et les enlever du code final

EADS CCR

Intégrité du binaire



Obscurcissement de code

Principe

- La partie de code responsable du déchiffrement des paquets est sensible
- Le code est ici obscurci
- Tout est mis en oeuvre pour ralentir sa compréhension
- Ajout de code mort, de calculs dynamiques d'adresses, ...

Randomisation des registres

```
mov     eax, 9FFB40h
sub     eax, 7F80h
mov     edx, 7799C1Fh
mov     ecx, [ebp-14h]
call    eax ; sub_9F7BC0
neg     eax
add     eax, 19C87A36h
mov     edx, 0CCDACEF0h
mov     ecx, [ebp-14h]
call    eax  ; eax = 009F8F70
```

Obscurcissement de code

Contre mesure

- Ici, on peut retrouver le code C
- On utilise des variables teintées
- On propage les informations obtenues sur les variables
- Et on regénère les expressions finales

```
/*****************************************************/
void sub_162590(unsigned int *TAB, unsigned int IN_KEY)
{
unsigned int tmp_var_1;
unsigned int tmp_var_0;

tmp_var_0 = TAB [ 56 ] ;
tmp_var_1 = ( TAB [ 60 ] ^ ( ( tmp_var_0 >= 0x291B9650 ) ? ( TAB [ 8 ] ) : ( tmp_var_0
TAB [ 60 ] = tmp_var_1 ;
}
/*****************************************************/
void sub_163670(unsigned int *TAB, unsigned int IN_KEY)
{
unsigned int tmp_var_0;

tmp_var_0 = ( ( TAB [ 0 ] + 0x4376FF7 ) ^ TAB [ 12 ] ) ;
TAB [ 12 ] = tmp_var_0 ;

}
```

Obscurcissement réseau

Flux réseaux

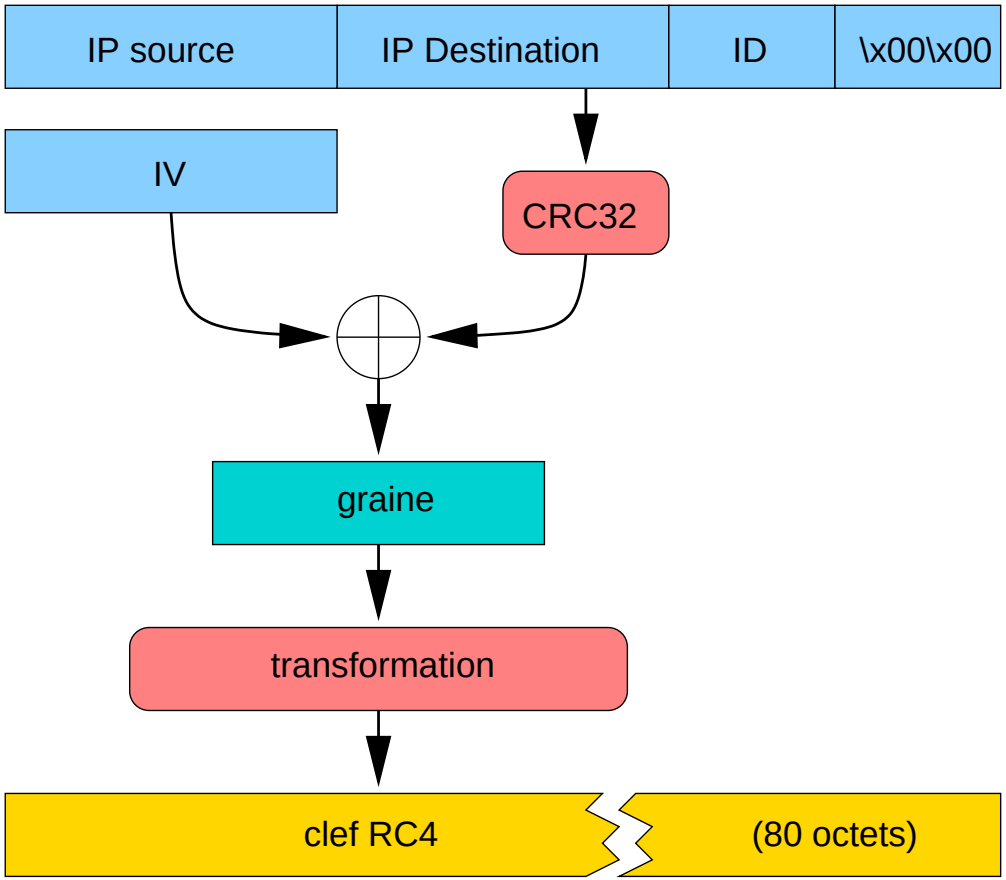
- Tout ceci ne serait pas complet si les paquets laissaient transpirer des données
- Les paquets sont obscurcis
- Ils sont chiffrés avec du RC4
- La clef de déchiffrement peut être retrouvée en connaissant les IP/PORT du paquet

Backdoor

Certaines backdoors sont capables d'utiliser du *ssl* pour camoufler les données exfiltrées par un attaquant

EADS CCR

Principe de l'obscurcissement réseau



Conclusion

Jusqu'ou iront-ils?

Camouflage de code

- Les techniques utilisées sont clairement mises en place pour éviter qu'un attaquant étudie le code
- Peut on laisser une liberté totale au développeur pour protéger un code?
- Doit-on banir les techniques permettant d'étudier le comportement d'un code?

RootKit Sony

- Que serait devenu le rootkit Sony s'il n'avait pas été découvert?

Conclusion
Jusqu'ou iront-ils?

Camouflage de code

- Les techniques utilisées sont clairement mises en place pour éviter qu'un attaquant étudie le code
- Peut on laisser une liberté totale au développeur pour protéger un code?
- Doit-on banir les techniques permettant d'étudier le comportement d'un code?

RootKit Sony

- Que serait devenu le rootkit Sony s'il n'avait pas été découvert?

EADS CCR

Questions?


WALT DISNEY HOME ENTERTAINMENT
PRESENTS A STUDIO GHIBLI FILM
CASTLE
IN THE SKYPE

References

F. Desclaux, *RR0D: the Rasta Ring 0 Debugger*

`http://rr0d.droids-corp.org/`

P. Biondi, *Scapy*

`http://www.secdev.org/projects/scapy/`