28/09/1999

ASPack – le compresseur décompressé par Psyché

Mesdames et Messieurs bonjour ;)

Christal avait demandé des informations sur la compression ASPack il y a peu, avant de rédiger un tut sur le soft Genius, lui-même défi du mois d'octobre 99 de La Main Rouge.

J'ai un peu bossé avec lui sur cette cible (j'adore les compresseurs ☺) et nous en sommes arrivés à la même conclusion lui et moi, à savoir que le loader du programme était également compressé (pour plus d'infos je vous renvoie à son tut et au mien sur Remote Selector et la compression en général).
Il fallait donc patcher à deux endroits différents. D'abord pour modifier le loader décompressé et l'amener au patch proprement dit et ensuite pour insérer ledit patch dans le programme cible.
Donc, à l'adresse où le loader redonne la main au soft principal il fallait obliger le programme à aller à notre patch, celui-ci modifiant les adresses du programme principal/cible pour le cracker.
Bref, je vous laisse relire le tut de Christal.

Il y avait juste un petit blème dans l'histoire.
Pour greffer son propre code à celui du loader il faut quelques octets non utilisés que l'on va parasiter.
Ceci fonctionne sans problème avec les compresseurs comme UPX ou Shrinker qui ont un beau code bien clair et en prime des octets libres, mais quand le loader est compressé, il faut ramer pour trouver de la place (un compresseur qui laisserait des " blancs " serait à revoir illico ☺).

Je pense avoir détaillé chaque octet du loader sans y trouver le moindre espace vide ☹ Ennuyant ça !

Mon pote Christal a résolu ce problème en sacrifiant l'icône du soft et en plaçant son patch dans la mémoire qui y était consacrée (qui bien entendu ne sera jamais exécutée – ben oui c'est du gfx !).
L'idée se défend tout à fait (à la guerre comme à la guerre ☺) mais j'avoue que ça ne me plaisait pas tout à fait … ça ne faisait " pas propre " ;)
Je me suis donc replongé dans le code du loader ASPack en tentant de le comprendre et trouver une autre solution.

De fil en aiguille (bien que je préfère le tricot ;)) je me suis rendu compte que ASPack ne dérogeait pas à la règle que j'avais mentionnée avec Remote Selector, à savoir qu'un compresseur doit être suffisamment général que pour être appliqué à n'importe quel soft.
Donc, pas de " truc " spécifique à l'un ou l'autre … et par extension, le loader est le même pour tous les softs compressés avec ASPack.

Mon but est donc de créer un loader “ modifié ”, applicable à tous les programmes compressés par ASPack et dans lequel on aura la place pour insérer le patch que l’on souhaite en fonction de la cible.
Mais voyons d’abord comment ça fonctionne.

Il faut savoir pour commencer que le loader ASPack occupe une section entière nommée .adata, le code de celui-ci débutant à l’offset 0 depuis l’adresse de début de cette section (raw offset sur disque ou virtual offset en mémoire).
D’abord, on trouve un morceau de code en clair (de l’offset 0 à l’offset @BA) qui va servir à décompresser la suite du loader.

```
:00000000 60                    pushad
:00000001 E800000000            call 00000006
:00000006 5D                    pop ebp
:00000007 81ED0A4A4400          sub ebp, 00444A0A
:0000000D BB044A4400            mov ebx, 00444A04
:00000012 03DD                  add ebx, ebp
:00000014 2B9DB1504400          sub ebx, dword ptr [ebp+004450B1]
:0000001A 83BDAC50440000        cmp dword ptr [ebp+004450AC], 00000000
:00000021 899DBB4E4400          mov dword ptr [ebp+00444EBB], ebx
:00000027 0F8517050000          jne 00000544
:0000002D 8D85D1504400          lea eax, dword ptr [ebp+004450D1]
:00000033 50                    push eax
:00000034 FF9594514400          call dword ptr [ebp+00445194]
:0000003A 8985CD504400          mov dword ptr [ebp+004450CD], eax
:00000040 8BF8                  mov edi, eax
:00000042 8D9DDE504400          lea ebx, dword ptr [ebp+004450DE]
:00000048 53                    push ebx
:00000049 50                    push eax
:0000004A FF9590514400          call dword ptr [ebp+00445190]
:00000050 8985B9504400          mov dword ptr [ebp+004450B9], eax
:00000056 8D9DEB504400          lea ebx, dword ptr [ebp+004450EB]
:0000005C 53                    push ebx
:0000005D 57                    push edi
:0000005E FF9590514400          call dword ptr [ebp+00445190]
:00000064 8985BD504400          mov dword ptr [ebp+004450BD], eax
:0000006A 8B85BB4E4400          mov eax, dword ptr [ebp+00444EBB]
:00000070 8985AC504400          mov dword ptr [ebp+004450AC], eax
:00000076 6A04                  push 00000004
:00000078 6800100000            push 00001000
:0000007D 689A040000            push 0000049A
:00000082 6A00                  push 00000000
:00000084 FF95B9504400          call dword ptr [ebp+004450B9]
:0000008A 8985B5504400          mov dword ptr [ebp+004450B5], eax
:00000090 8D9DCF4A4400          lea ebx, dword ptr [ebp+00444ACF]
:00000096 50                    push eax
:00000097 53                    push ebx
:00000098 E8C8040000            call 00000565 <- décompresse le loader
:0000009D 8BC8                  mov ecx, eax <- ecx = 49a
:0000009F 8DBDCF4A4400          lea edi, dword ptr [ebp+00444ACF]
:000000A5 8BB5B5504400          mov esi, dword ptr [ebp+004450B5]
:000000AB C1F902                sar ecx, 02
:000000AE F3                    repz
:000000AF A5                    movsd
:000000B0 8BC8                  mov ecx, eax
:000000B2 83E103                and ecx, 00000003
:000000B5 F3                    repz
:000000B6 A4                    movsb
:000000B7 8B85B5504400          mov eax, dword ptr [ebp+004450B5]
:000000BD 6800800000            push 00008000 <- début de la zone compactée
:000000C2 6A00                  push 00000000
:000000C4 50                    push eax
:000000C5 FF95BD504400          call dword ptr [ebp+004450BD]
:000000CB 8D0E                  lea ecx, dword ptr [esi]
:000000CD 8537                  test dword ptr [edi], esi
:000000CF 4C                    dec esp
:000000D0 44                    inc esp
:000000D1 07                    pop es
:000000D2 50                    push eax
:000000D3 C3                    ret
```

Le call à l’offset @98 va décompresser toute la partie débutant à l’offset @BD dans une mémoire allouée à cet effet un peu plus haut (le call à l’offset @5E).

Ensuite, ESI pointera vers cette zone contenant le code décompressé et EDI pointera vers l'offset @BD. Tout ceci pour transférer 49A (= 1178) octets de ESI vers EDI.
Ici prend fin la restauration du loader.

L'idée est donc de récupérer ces 1178 octets et remplacer la partie compressée par celle décompressée dans la section .adata.
En faisant cela, tout le code nécessaire à cette décompression deviendra inutile et nous pourrons utiliser cet espace (de l'offset @96 - en fait @90 irait aussi- à l'offset @CA) à des fins inavouables.
Si nécessaire, il y a moyen de récupérer encore quelques octets depuis l'offset @76 car ce qui suit est le traîtement des erreurs éventuelles qui auraient pu survenir au moment de la réservation d'une zone mémoire (le call [EBP+445190]) qui servira à la décompression du loader ET du programme cible (!). C'est un peu risqué mais au pire ça crashe en cas de problème … à vous de voir. Donc considérons plutôt @90 (je continuerai avec @96 dans les listings ci-dessous car je n'ai remarqué que plus tard que @90 n'était pas crucial … en plus je suis fainéant, je n'avais pas envie de tout refaire ☺).

Y a plus qu'à … Enfin presque …. ☺
Si on détaille un peu plus, on se rend compte que le début du loader utilise 4 octets de cette zone pour transmettre des données (il me semble que c'est l'Image Base qui y est stockée).
A l'offset @21, EBX est stocké et ensuite récupéré dans EAX à l'offset @6A.
Il nous faudra tenir compte de ce petit détail en prenant soin de restaurer les valeurs qui se trouvent à cet endroit après décompression (ce que j'ai fait à l'offset @96).

```
:00000000 60                    pushad
:00000001 E800000000            call 00000006
:00000006 5D                    pop ebp
:00000007 81ED0A4A4400          sub ebp, 00444A0A
:0000000D BB044A4400            mov ebx, 00444A04
:00000012 03DD                  add ebx, ebp
:00000014 2B9DB1504400          sub ebx, dword ptr [ebp+004450B1]
:0000001A 83BDAC50440000        cmp dword ptr [ebp+004450AC], 00000000
:00000021 899DBB4E4400          mov dword ptr [ebp+00444EBB], ebx
:00000027 0F8517050000          jne 00000544
:0000002D 8D85D1504400          lea eax, dword ptr [ebp+004450D1]
:00000033 50                    push eax
:00000034 FF9594514400          call dword ptr [ebp+00445194]
:0000003A 8985CD504400          mov dword ptr [ebp+004450CD], eax
:00000040 8BF8                  mov edi, eax
:00000042 8D9DDE504400          lea ebx, dword ptr [ebp+004450DE]
:00000048 53                    push ebx
:00000049 50                    push eax
:0000004A FF9590514400          call dword ptr [ebp+00445190]
:00000050 8985B9504400          mov dword ptr [ebp+004450B9], eax
:00000056 8D9DEB504400          lea ebx, dword ptr [ebp+004450EB]
:0000005C 53                    push ebx
:0000005D 57                    push edi
:0000005E FF9590514400          call dword ptr [ebp+00445190]
:00000064 8985BD504400          mov dword ptr [ebp+004450BD], eax
:0000006A 8B85BB4E4400          mov eax, dword ptr [ebp+00444EBB]
:00000070 8985AC504400          mov dword ptr [ebp+004450AC], eax
:00000076 6A04                  push 00000004
:00000078 6800100000            push 00001000
:0000007D 689A040000            push 0000049A
:00000082 6A00                  push 00000000
:00000084 FF95B9504400          call dword ptr [ebp+004450B9]
:0000008A 8985B5504400          mov dword ptr [ebp+004450B5], eax
:00000090 8D9DCF4A4400          lea ebx, dword ptr [ebp+00444ACF]
:00000096 C785BB4E440085DB7471  mov dword ptr [ebp+00444EBB], 7174DB85 <- restaure le bon
                                                                          dword
:000000A0 EB29                  jmp 000000CB <- on saute le code qui décompresse le loader
:000000A2 4A                    dec edx
:000000A3 44                    inc esp
```

```
:000000A4 008BB5B55044          add byte ptr [ebx+4450B5B5], cl
:000000AA 00C1                  add cl, al
:000000AC F9                    stc
:000000AD 02F3                  add dh, bl
:000000AF A5                    movsd
:000000B0 8BC8                  mov ecx, eax
:000000B2 83E103                and ecx, 00000003
:000000B5 F3                    repz
:000000B6 A4                    movsb
:000000B7 8B85B5504400          mov eax, dword ptr [ebp+004450B5]
:000000BD 6800800000            push 00008000
:000000C2 6A00                  push 00000000
:000000C4 50                    push eax
:000000C5 FF95BD504400          call dword ptr [ebp+004450BD]
:000000CB 8D85374C4400          lea eax, dword ptr [ebp+00444C37] ← on arrive ici
:000000D1 50                    push eax
:000000D2 C3                    ret
```

Bilan de l'histoire … 41 octets libres pour nous amuser.
Il faut maintenant repérer la fin du loader, c-à-d là où le programme principal reprend la main. Bref, ici :

```
:00000558 61                    popad
:00000559 7508                  jne 00000563
:0000055B B801000000            mov eax, 00000001
:00000560 C20C00                ret 000C
:00000563 50                    push eax
:00000564 C3                    ret
```

EAX contient l'Entry Point du programme principal. Le loader place EAX sur la Pile et le RET suivant va rechercher sur la pile l'adresse à laquelle il est sensé rebrancher le soft (astucieux comme méthode, non ?).
A nous maintenant d'amener le loader vers nos patches éventuels avant de lancer le programme cible.

```
:00000558 61                    popad
:00000559 E944FBFFFF            jmp 000000A2 <- saute vers notre espace libre

:0000055E 0000                  add byte ptr [eax], al
:00000560 C20C00                ret 000C
:00000563 50                    push eax
:00000564 C3                    ret
- - - - - - - -

:00000090 8D9DCF4A4400          lea ebx, dword ptr [ebp+00444ACF]
:00000096 C785BB4E440085DB7471  mov dword ptr [ebp+00444EBB], 7174DB85
:000000A0 EB29                  jmp 000000CB
:000000A2 50                    push eax   <- ICI
:000000A3 C3                    ret
:000000A4 00000000000000000000  BYTE 10 DUP(0)
:000000AE 00000000000000000000  BYTE 10 DUP(0)
:000000B8 00000000000000000000  BYTE 10 DUP(0)
:000000C2 000000000000000000    BYTE  9 DUP(0)

:000000CB 8D85374C4400          lea eax, dword ptr [ebp+00444C37]
:000000D1 50                    push eax
:000000D2 C3                    ret
```

Je n'ai sciemment rien mis ici car c'est ce sont des considérations générales, j'ai simplement remis le push eax et le RET qu'il ne faudra bien sûr pas oublier si on veut que la cible s'exécute ☺.
En définitive, nous avons de cette manière 39 octets à notre disposition, ce qui, dans la plupart des cas, est amplement suffisant.

Voyons à présent comment procéder.

Il va nous falloir d'abord un exemplaire de la section .adata décompressée.
ProcDump nous fournira ça rapidement :

1. lancer la cible
2. lancer ProcDump
3. cliquer avec le bouton de droite sur le nom de notre cible
4. Dump (full)
5. Sauver l'exécutable dumpé

Il nous faut maintenant savoir où commence .adata dans cette version décompressée.
1. lancer ProcDump
2. PE Editor
3. SECTIONS
4. Regarder le Raw Offset (offset sur le HD) de la section .adata

De quoi avons-nous besoin ?
De la partie allant de l'offset @96 jusqu'à @557 (@CB + 49A [= nb d'octets décompressés]) à partir du début de .adata (= raw offset de .adata).
Grâce à n'importe quel éditeur héxa il y a moyen d'extraire le morceau qui nous intéresse et en faire un fichier " aspack.bin " (par exemple) par simple copier-coller.
Nous avons maintenant 4C1 (557-96) octets dans notre fichier " aspack.bin ".

Pour le moment nous avons une simple copie d'un morceau de .adata.
Il va falloir lui apporter les modifications que j'ai faites plus haut aux offsets :
@96->A1 (sera l'offset 0 de aspack.bin puisqu'on a sauvé à partir de @96)
@A2->A3 (offset @0C [@A2-@96] de aspack.bin)
et @559->55D (offset @4C3 [@559-@96] de aspack.bin).

On sauve le tout et nous sommes maintenant en possession d'une version " standard ", c-à-d qu'il suffit par copier-coller de remplacer le code de .adata depuis l'offset @96 (attention : raw offset + @96) par notre aspack.bin.
L'exécution ne devrait poser aucun problème …

Vous pourrez dès à présent utiliser aspack.bin pour insérer vos patches en plaçant les bons octets à l'offset @C (@A2-@96) dans aspack.bin en n'oubliant PAS de remettre le PUSH EAX et le RET à la fin du patch BIEN SUR !

Il y aurait bien sûr moyen d'automatiser la modification par un petit programme pas trop compliqué.
Il devrait d'abord repérer le raw offset de .adata. Il suffit de chercher les lettres 'adata' au début du soft (header) et 15 octets plus loin on trouve un dword correspondant au raw offset => stocker cette valeur.
Ouvrir aspack.bin.
Placer le pointeur du fichier cible sur le raw offset de .adata + 96 (héxa).
Copier les 49A octets de aspack.bin dans le programme cible à partir du pointeur.
Fermer le tout et hop !
Je n'ai malheureusement pas le temps de l'écrire moi-même mais si ça vous tente … on pourra toujours insérer le listing ci-dessous ☺

Bonne réflexion et A+
Psyche
Fred_psyche@hotmail.com