Analyse du malware de Clément, Rémi et Zakaria

...

Emy LIEUTAUD, Gautier KLAM & Yunqiao ZHANG

Déroulement de l'analyse

1. Exécution du programme
2. Utilisation d'IDA
    a. Debugger
    b. Messages / Chaînes de caractères
    c. Clé(s)
3. Vérification des clés trouvées

Exécution du programme

Exécution du programme

Paramètre avec des caractères non hexadécimaux :

```
C:\Documents and Settings\Administrateur\Bureau\ProjetLezghamSalaunBrun>Projet.e
xe ffaaziohdf
ffaaziohdf
```

⇒ fonctionnement normal : pas de vérification

Paramètre avec plus de 64 caractères :

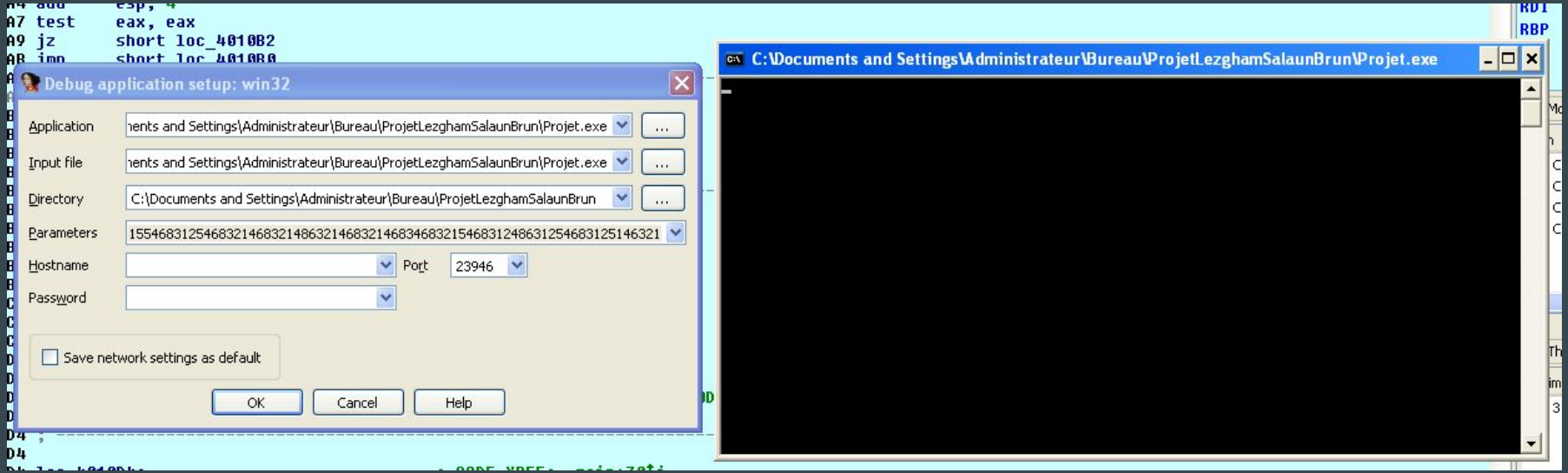
```
C:\Documents and Settings\Administrateur\Bureau\ProjetLezghamSalaunBrun>Projet.e
xe ffaaffaaff3541531213512174812782875287236758937698352897238765235678235678923
7893285235678693287235682367823563289 23
trop long
```

⇒ message d'erreur

Analyse avec IDA

Vérification du debugger

Lancement du debugger avec une clé trop longue :



⇒ on ne voit pas le message “trop long” : le programme vérifie si un debugger est présent

IsDebuggerPresent

```
.text:00401074                push    esi
.text:00401075                mov     esi, ds:_strrev
.text:0040107B                push    edi
.text:0040107C                mov     edi, [ebp+argv]
.text:0040107F                lea     eax, [ebp+flOldProtect]
.text:00401085                push    eax             ; lpflOldProtect
.text:00401086                push    40h             ; flNewProtect
.text:00401088                push    6               ; dwSize
.text:0040108A                push    esi             ; lpAddress
.text:0040108B                call    ds:VirtualProtect
.text:00401091                push    offset alkefuzehfezbfnklze ; "lkefuzehfezbfnklze"
.text:00401096                mov     dword ptr [esi], 82F6EF68h
.text:0040109C                mov     word ptr [esi+4], 0C37Ch
.text:004010A2                call    esi ; _strrev
.text:004010A4                add     esp, 4
.text:004010A7                test    eax, eax
.text:004010A9                jz      short loc_4010B2
.text:004010AB                jmp     short loc_4010B0
```

strrev : fonction qui inverse une chaîne de caractères

401075 : strrev stockée dans esi

4010A2 : esi appelé (soit strrev)

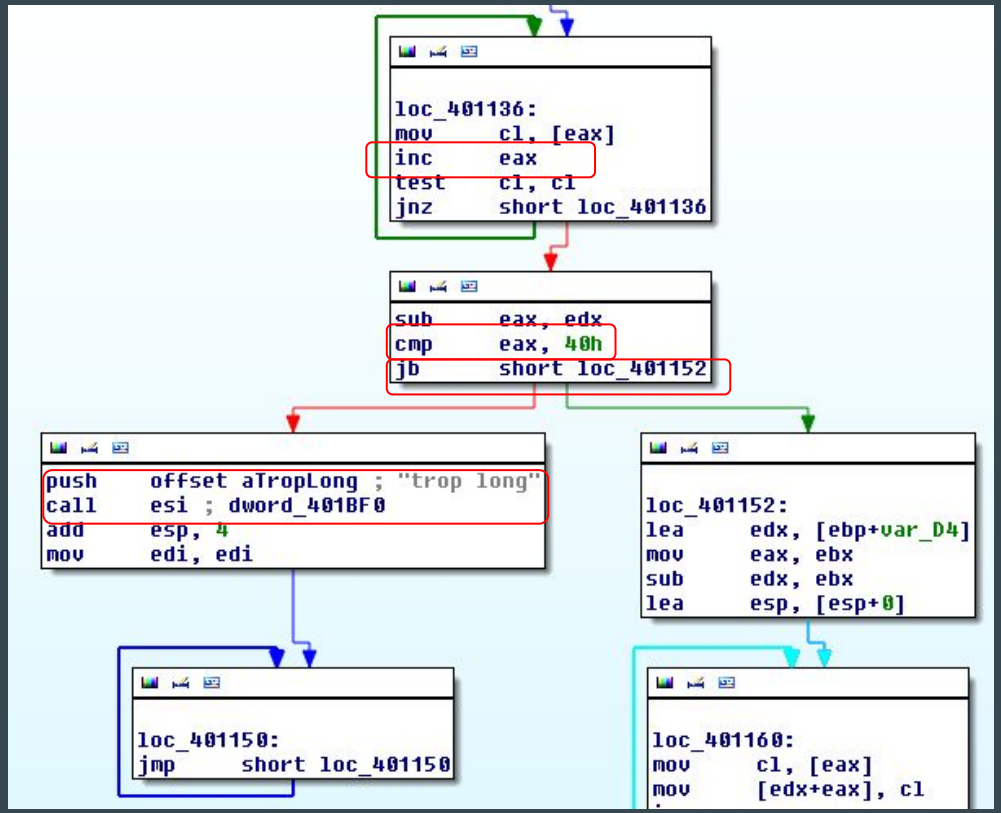
401096 et 40109C : modification de l'adresse de esi (VirtualProtect appelé en 40108B)

68 EFF6827C C3 revient à :

```
push IsDebuggerPresent
ret
```

⇒ obfuscation de fonction : strrev est en fait IsDebuggerPresent

Message “trop long”

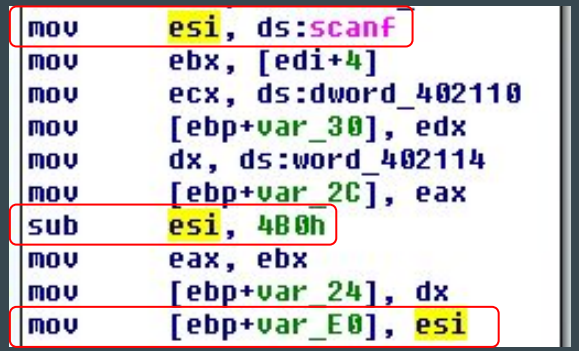


401138 : eax est incrémenté pour chaque caractère de la clé saisie

40113F : comparaison entre eax et 40h (= $64_{(10)}$)

401142 : jb = jump below :
Si eax < 64 ⇒ condition vraie : suite du programme
Sinon ⇒ condition fausse : affichage du message “trop long”

Affichage avec printf



```
mov     esi, ds:scanf
mov     ebx, [edi+4]
mov     ecx, ds:dword_402110
mov     [ebp+var_30], edx
mov     dx, ds:word_402114
mov     [ebp+var_2C], eax
sub     esi, 4B0h
mov     eax, ebx
mov     [ebp+var_24], dx
mov     [ebp+var_E0], esi
```

4010DF : esi devient scanf

adresse de scanf : 78B05B64

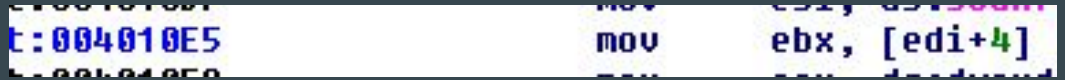
4010FB : soustraction de 4B0h à esi soit à l'adresse de scanf
78B05B64-4B0 = 78B056B4 = adresse de printf

⇒ esi et [ebp+var_E0] deviennent printf

Comparaison des clés - récupération de la clé saisie

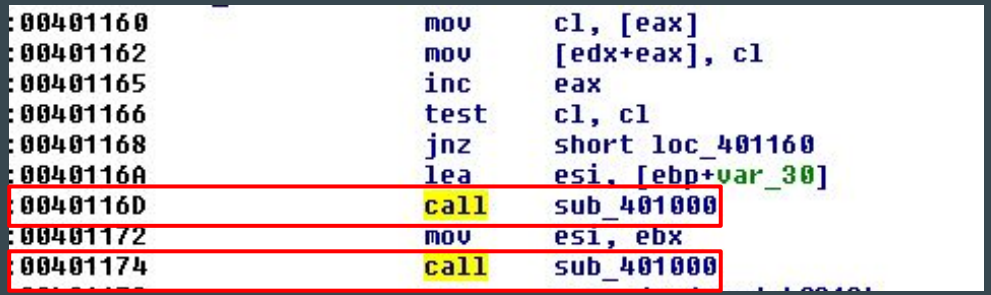
40107C : on récupère l'adresse des paramètres (arguments)

argv = ["Projet.exe", cle_saisie]



4010E5 : on récupère l'adresse de argv[1], soit de la clé saisie

Comparaison des clés - hachage



call sub_401000 : fonction de hachage

40116D : on hache esi soit [ebp+var_30] (dword “zlkfnprnvjzuy”)

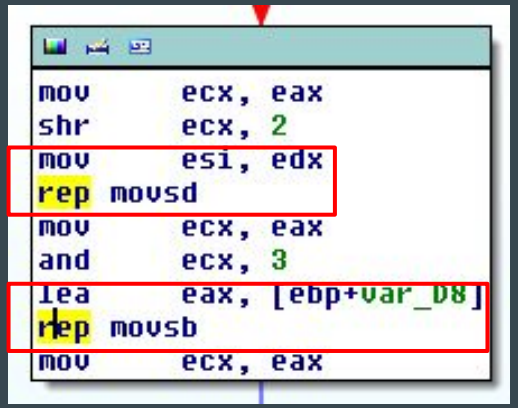
401174 : on hache la clé saisie par l’utilisateur (esi soit ebx soit [edi+4] soit [ebp+argv+4])

Comparaison des clés - création du message final

```
00401179        mov     eax, ds:dword_402124
0040117E        mov     cl, ds:byte_402128
00401184        mov     edx, '}'
00401189        mov     [ebp+var_20], eax
0040118C        mov     [ebp+var_1C], cl
0040118F        mov     word ptr [ebp+var_D8], dx
00401196        xor     eax, eax
00401198        jmp     short loc_4011A0
```

401189 : [ebp+var_20] = CLE{

40118F : [ebp+var_D8] = }

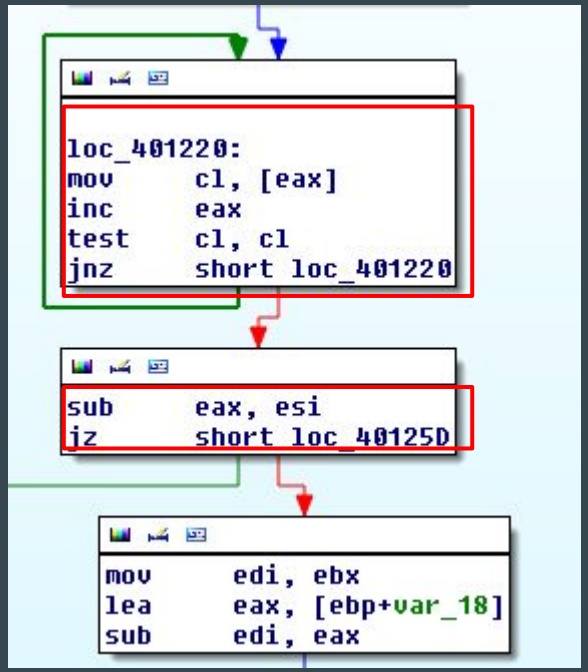


4011D4 : rep movsd → on copie le haché de “zlkfnprnvjzuy” ([ebp+var_30])

4011E1 : rep movsb → on copie “}” ([ebp+var_D8])

⇒ création de la chaîne de caractère affichée lors de la saisie de la bonne clé : CLE{hash_zlkfnprnvjzuy}

Comparaison des clés - longueur de la clé

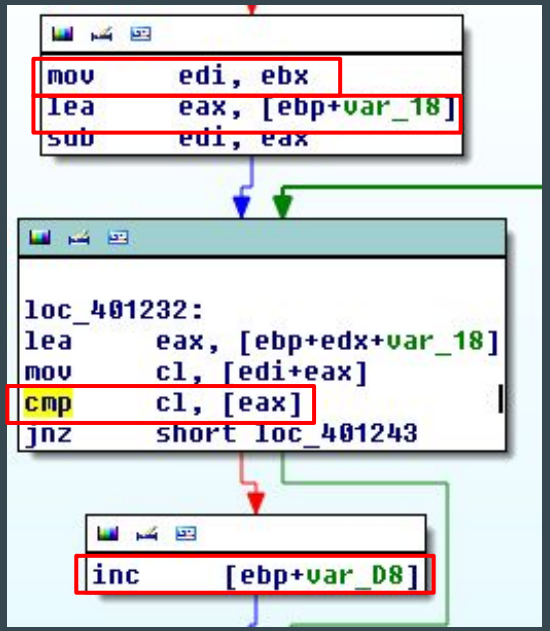


loc_401220 : traitement des chaînes de caractères qui se terminent par un octet nul (représenté par une valeur de zéro)

Si le caractère NULL a été lu ⇒ condition fausse : la chaîne est terminée
Sinon ⇒ condition vraie : continuer à lire les caractères

2ème bloc : on regarde si la chaîne est supérieure à 0
Si elle vaut 0 → on saute l'étape de comparaison avec le hash
Sinon → on va comparer avec le hash

Comparaison des clés - comparaison des hachés



4011F7 : edi = la clé saisie par l'utilisateur

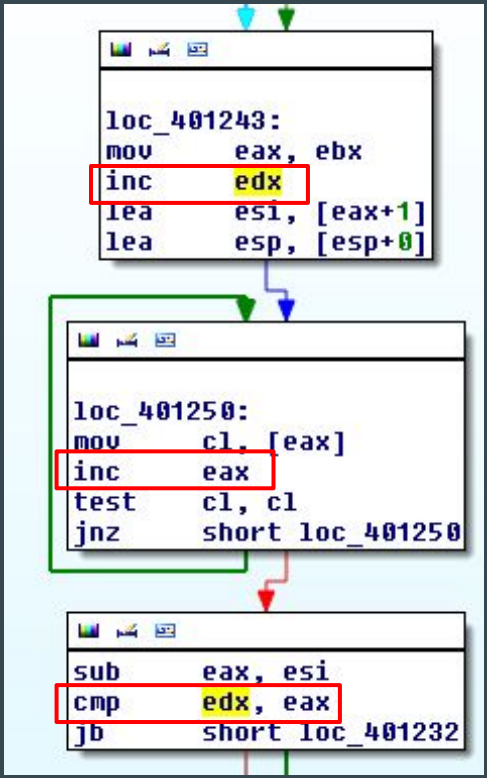
[ebp+var_18] = 20 valeurs initialisées :

```
:0040110D   mov   [ebp+var_18], 4011512h
:00401114   mov   [ebp+var_14], 7040A10h
:0040111B   mov   [ebp+var_10], 4151812h
:00401122   mov   [ebp+var_C], 19020A07h
:00401129   mov   [ebp+var_8], 15011213h
```

401239 : comparaison entre le caractère haché de la clé saisie et le caractère haché de la vraie clé

Si les hachés des caractères sont les mêmes ⇒ on incrémente le compteur (qui compte le nombre de caractères validés)

Comparaison des clés - comparaison des hachés



401245 : on incrémente un compteur qui compte le nombre de caractères comparés (edx)

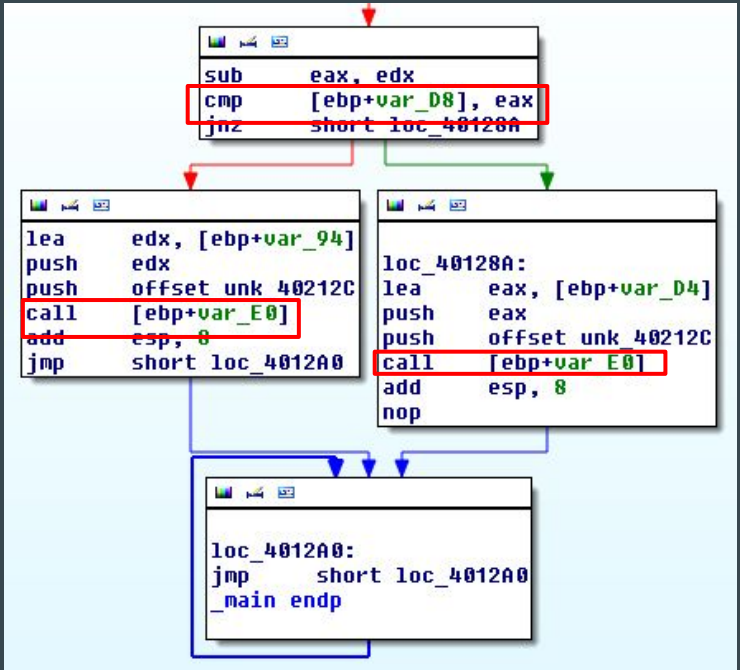
401252 : on incrémente un compteur qui compte la longueur de la clé saisie (eax)

401259 : edx et eax

Si on n'a pas encore comparé tous les caractères ⇒ on refait la boucle

Si toute la clé a été comparée ⇒ on affiche les résultats

Comparaison des clés - affichage message

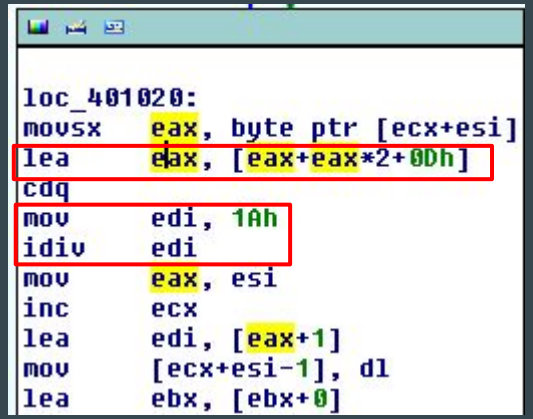


40126B : comparaison entre le nombre de caractères valides ([ebp+var_D8]) et la longueur de la clé saisie (eax)

Si elles sont différentes : on affiche la clé saisie non hachée
Si elles sont identiques : on affiche le message avec les 12 caractères hachés
(CLE{☼↓—♣♂◄♣❤‼☼})

call [ebp+var_E0] : appeler printf

Fonction de hachage - *sub_401000*



```
loc_401020:
movsx   eax, byte ptr [ecx+esi]
lea     eax, [eax+eax*2+0Dh]
cdq
mov     edi, 1Ah
idiv    edi
mov     eax, esi
inc     ecx
lea     edi, [eax+1]
mov     [ecx+esi-1], dl
lea     ebx, [ebx+0]
```

- Multiplication de la valeur ASCII du caractère par 3 (équivalent à eax + eax*2)
- Ajout de 13 (0Dh)
- Modulo 26 (1Ah)

⇒ Chaque caractère (c) est remplacé par (3*c+13)%26

Brute force du paramètre

```python
key = [0x12, 0x15, 0x01, 0x04,
       0x10, 0x0a, 0x04, 0x07,
       0x12, 0x18, 0x15, 0x04,
       0x07, 0x0a, 0x02, 0x19,
       0x13, 0x12, 0x01, 0x15]

compute_cache = {}
def compute(c):
    if compute_cache.get(c) is None:
        compute_cache[c] = (ord(c)*3+0x0d) % 0x1a
    return compute_cache[c]

if __name__ == "__main__":
    res = []
    chars = "0123456789abcdefghijklmnopqrstuvwxyz"
    for k in key:
        possible_chars = []
        for c in chars:
            if compute(c) == k:
                possible_chars.append(c)
        res.append(possible_chars)

    for i in range(max(len(i) for i in res)):
        print("".join(" " if i >= len(l) else l[i] for l in res))
```

script Python

compute(c) → fonction de hachage du caractère c

Création d'une boucle for permettant de comparer le hash de la vraie clé avec le hash d'un caractère (lettre ou chiffre)

affichage des possibilités

Clé possibles :

```
ab015312acb123986a0b
  deigef   efgmlj d
```

Vérification des clés trouvées

Test de 7 clés différentes précédemment obtenues par le script python :

```
C:\Documents and Settings\Administrateur\Bureau>Projet.exe ab015312acb123986a0b
CLE{☼↓▬♀♂◄♀♥‼☼}^C
C:\Documents and Settings\Administrateur\Bureau>
C:\Documents and Settings\Administrateur\Bureau>Projet.exe abd15312acb123986a0b
CLE{☼↓▬♀♂◄♀♥‼☼}^C
C:\Documents and Settings\Administrateur\Bureau>Projet.exe abde5312acb123986a0b
CLE{☼↓▬♀♂◄♀♥‼☼}^C
C:\Documents and Settings\Administrateur\Bureau>Projet.exe abde5312acbef3986adb
CLE{☼↓▬♀♂◄♀♥‼☼}^C
C:\Documents and Settings\Administrateur\Bureau>Projet.exe abde53e2acbef3986adb
CLE{☼↓▬♀♂◄♀♥‼☼}^C
C:\Documents and Settings\Administrateur\Bureau>Projet.exe abde53efacbef3986adb
CLE{☼↓▬♀♂◄♀♥‼☼}^C
C:\Documents and Settings\Administrateur\Bureau>Projet.exe abdei3efacbef3m86adb
CLE{☼↓▬♀♂◄♀♥‼☼}^C
```

ab015312acb123986a0b
abd15312acb123986a0b
abde5312acb123986a0b
abde5312acbef3986adb
abde53e2acbef3986adb
abde53efacbef3986adb
abdei3efacbef3m86adb

⇒ différentes clés valables (hexadécimales et non-hexadécimales

Conclusion

- Vérification si un débugger est présent
- Accepte une clé < 64 caractères (hexadécimaux et non hexadécimaux)
- Différentes clé possibles : 2^13 = 8192 clés
- Très faible sécurité : on peut contourner la sécurité avec 1 caractère

Merci pour votre attention !