Writeups finale SigSegV2

T0t0r0, team Legend88

Je m'excuse comme toujours pour la forme (un jour AK va me tuer :x)

Introduction

Ces challenges sont issus du CTF final de SigSegV2, événement organisé par RTFM dans les locaux de l'Epitech à côté de Paris et s'étant déroulé entre ~ 22h et 10h10 du matin dans la nuit du samedi/dimanche 30 novembre/1er décembre (même si j'ai ragequit/arrêté par fatigue bien avant).

I. Reverse

1. Baby Android (résolu par AK, je l'ai fait dans la semaine)

On décompile baby.apk avec jadx. On va dans les sources, on remarque dans le Manifest.xml ou directement en allant dans com que nous avons une application sigsegv avec la classe MainActivity où nous avons une fonction flagOn qui compare une string en base64 au résultat de chall.encode(flag). La fonction encode est présente dans la classe Chall avec le code simpliste suivant :

```
public class Chall {
   private String _key = "SigSegV2";

   public String b64Encode(String text) throws UnsupportedEncodingException {
      return Base64.encodeToString(text.getBytes("UTF-8"), 0);
   }

   public String encode(String flag) throws UnsupportedEncodingException {
      return b64Encode(new String(otherEncode(flag.getBytes(), this._key.getBytes())));
   }

   private byte[] otherEncode(byte[] a, byte[] key) {
      byte[] out = new byte[a.length];
      for (int i = 0; i < a.length; i++) {
         out[i] = (byte) (a[i] ^ key[i % key.length]);
      }
      return out;
   }
}
```

La fonction encode en base64 le résultat de l'appel à la fonction otherEncode sur le flag et la clé présente dans la même classe plus haut _key = "SigSegV2".

La fonction otherEncode XORe chaque caractère du flag avec un caractère de la clé (quand on dépasse la taille de la clé, on recommence au début de la clé grâce au modulo).

Petit rappel trivial (si vous avez déjà fait des CTFs ou un minimum de logique, passez votre chemin) :

On sait qu'à partir de A XOR B = C on peut retrouver A car quelque soit N, N XOR N = 0 (1) et N XOR 0 = N (2) et on a l'associativité (3)
A XOR B = C (A XOR B) XOR B = A XOR (B XOR B) par (3)
= A XOR 0 par (1)
= A par (2)
donc on a bien A XOR B = C <=> A =C XOR B = B XOR C par commutativité en appliquant XOR B des deux côtés
Dans notre cas on a pour simplifier flag ^ key_string_modulo = base64_decoded_string
donc flag = key_string_modulo ^ base64_decoded_string où key_string_modulo est connu et par décodage élémentaire l'autre élément l'est également

On peut donc résoudre le challenge en full static avec un code sale mais rapide en python2 avant qu'il meurt :

```
import base64 as b
string = "IAAAIAAAIEkRXV8KOlM4diFZVhc6IWZADFxUJxA3Kw=="
key_str="SigSegV2"
flag=""

letters = b.b64decode(string)

for i in range(len(letters)):
    letter = letters[i]
    key = key_str[i%len(key_str)]
    flag += chr(ord(letter)^ord(key))

print flag
```

Notre précieux : sigsegv{B48Y_4nDr01D_F0r_53tuP}

II. Forensic

Petits tips généraux quand on a des dump mémoire et/ou qu'on doit utiliser volatility (je les avais notés sur le Discord de Legend88 pour un autre membre donc autant en faire profiter toute personne n'ayant jamais touché à volatility) :

- toujours avoir des cheatsheets sous les yeux au cas où (autant en Windows on commence à les connaître, autant pour un Linux ou un Mac on est plus embêté). Par exemple :
https://digital-forensics.sans.org/media/volatility-memory-forensics-cheat-sheet.pdf
https://downloads.volatilityfoundation.org/releases/2.4/CheatSheet_v2.4.pdf

- lire le MISC parlant de volatility, il y a quelques petits tips sympas, notamment utiliser un fichier .volatilityrc pour se simplifier la vie en ligne de commande au niveau des arguments à déclarer

- récupérer des profils Linux (sic) et Mac (on sait jamais sur un malentendu ça peut servir)

- récupérer des plugins sympathiques :
* pour des volumes chiffrés (truecrypt (rip), filevault,... je vous laisse chercher)
* récupérer des cred (mimikatz)
* récupérer plus facilement des historiques (chromehistory, firefoxhistory)

1. **Je rim et je ram** de jenaye (résolu après la fin du challenge)

NB : Je n'ai pas réussi ce chall dans le temps imparti, je ne trouvais pas le password, j'ai oublié de repasser sur notepad après avoir vu les liens vers les pastebin like (j'ai bien ragé)

On télécharge un fichier yolo2.raw.

Après avoir un peu regardé son contenu et vu son nom, on dirait bien un dump. On lance volatility.

volatility -f yolo2.raw imageinfo nous donne son profil : Win2008SP2x64

Ensuite on déroule les commandes habituelles :
volatility -f yolo2.raw --profile=Win2008SP2x64 pslist
De même avec filescan

Dans pslist on découvre un notepad. On dumpe le process (memdump en précisant le PID avec -p). Je fais un strings sur le dump du processus et grep sur sigsegv\{ mais rien. Je me rappelle ensuite que notepad gère un peu bizarrement son espace (16 bits LE) et on fait donc un strings -e l (ce que j'ai peut-être oublié de faire a posteriori pour trouver le password) mais toujours rien.

Avec filescan, on trouve user.txt et user.txt.txt, user.lnk et yolo.lnk. J'ai essayé d'en dump certains (commande dumpfiles en précisant l'offset avec -Q par exemple), dans l'un on a un faux flag qui ne respecte par le format à cause d'une petite transposition/switch. Je vérifie dans la MFT avec la commande mftparser, je ne trouve rien de plus.

Avec la commande iehistory, on trouve des URLs intéressantes ou plutôt spéciales : une sur cryptobin.co et l'autre sur privatebin.net. On s'intéressera par la suite à cryptobin.co/71q7y462.

On retrouve la même URL dans le presse-papier (clipboard).

On va sur l'un des deux sites, on voit qu'on a un texte chiffré, qu'il faut un mot de passe. En décodant le texte en base64, on voit qu'on a de l'AES CCM avec une taille de clé legit de 256 bits, bon bah on se dit qu'il va falloir retrouver le mot de passe quelque part :/

Toutes les commandes standards (cachedump, lsadump, mimikatz,...) ne donnent rien.

Il fallait retourner sur le dump du notepad, faire un strings -e l et grep -i sur password ou rtfm :
password : RTFM_YELAA

Et on peut ainsi retourner sur le site et déchiffrer notre précieux :
sigsegv2{Pow3r_oF_Vol}

2. 10 questions about my system de noraj (le 9 a été résolu après la fin du CTF)

Bon alors déjà, on voit que le profil est donné dans l'énoncé (openSUSE Leap 15.1) et que c'est un openSUSE, ça ça sous-entend souvent qu'il va falloir faire son propre profil. Youpi avec ma RAM horrible. Bon bah let's go.

On crée une VM openSUSE grâce à une ISO récupérée sur le site officiel

On veut créer un profil volatility, donc on veut récupérer de bons System.map et module.dwarf (un profil c'est en gros une distribution et un kernel spécifique).

Je l'avais déjà fait pour un autre dump mais ça faisait longtemps, j'ai repris mes marques avec un vieuxx post d'evild3ad et le writeup de StormXploit lors du HackSécuReims. Si vous voulez plus de détails, PM sur Discord ou Twitter, je créerais peut-être un article pour aller plus dans le détail s'il y a de la demande.

Bref, rien de compliqué ici, ça prend juste du temps suivant votre RAM et il faut pas se planter sur la version du kernel/noyau suivant ce que vous avez installé sur votre VM (même si normalement vous avez la bonne).

On met ce profil au bon endroit sur votre machine (/usr/lib/python2.7/dist-packages/volatility/plugins/overlays/linux de mon côté). On regarde comment volatility l'a appelé :

```
volatility --info | grep Profile | grep -i opensuse
Volatility Foundation Volatility Framework 2.6
Linuxopensusex64      - A Profile for Linux opensuse x64)
```

Bon, on peut donc commencer à répondre aux questions. Il faut se rappeler que les commandes vont êtres spécifiques à Linux mais que certaines ressemblent à celles de Windows ; une cheatsheet à côté fait toujours du bien au cas où.

Les flags devaient être sous la forme : sigsegv{sha1(flag)}. Je ne vais pas mettre le SHA1 à chaque fois, vous êtes grands. En plus au moment où je vous parle je viens de voir que la team Cryptis a déjà fait un WU sur le sujet donc vous pouvez voir les flags là-bas ; mais bon vu que j'avais déjà bien commencé ce writeup, je vais le finir.

Je pense que pour certaines questions on aurait pu faire des strings/grep triviaux.

Sinon, il y a eu un problème sur les points de ces challenges qui sont restés anormalement élevés.
En citant noraj sur le discord RTFM:
“
PS : pour les points des challenges on est désolé. La formule de décroissance était [CENSURE]. Les 10 questions forensics et les 9 hashs a cracker étaient censé descendre jusqu'à 5 points min. et les autres challs facile descendre à 50 pts min. Là rien n'a descendu en dessous de 484 pts... Peut être que @Th3_l5D pourrait faire des simulations avec une vrai formule
“

Question 1

Quelle est la commande que noraj a utilisé à `2019-11-19 22:57:38 UTC+0000`?

On nous demande une commande, on va donc regarder l'historique des commandes dans le bash et grep sur la date donnée :

volatility -f chall.raw –profile=Linuxopensusex64 linux_bash

Question 2

Question n°2: Quelle version de gcc a été utilisée pour compiler le kernel? (string entière)

On essaie de retrouver la bannière dans dmesg/les logs kernel après avoir utilisé la commande linux_dmesg et avoir redirigé l'output dans un fichier :

cat linux_dmesg | grep -i linux
[0.0] Linux version 4.12.14-lp151.28.32-default (geeko@buildhost) (gcc version 7.4.1 20190905 [gcc-7-branch revision 275407] (SUSE Linux)) #1 SMP Wed Nov 13 07:50:15 UTC 2019 (6e1aaad)

On a donc gcc version 7.4.1 20190905 [gcc-7-branch revision 275407] (SUSE Linux)

Question 3

Quel est le message de debug à `1105416124.1`?

On utilise la même commande linux_dmesg et on grep.

Question 4

Quelle est l'adresse IP de eth0 et son adresse MAC? (concatène la réponse)

On utilise la commande linux_ifconfig et on grep sur eth0.

Question 5

Quelle est la 3ème bibliothèque chargée par sshd?

On utilise la commande linux_library_list, on grep sur sshd et on compte.

Question 6

Quels sont le système de fichier et les options de montage de `/tmp`? (concatène la réponse)

On utilise linux_mount et on grep sur tmp en regardant la 2ème colonne (ou alors on cat linux_mount | grep tmp | awk '{print $2}' pour trouver le nom exact d'abord)

Question 7

Quels sont le nom/pid du processus qui utilise le socket UNIX 18707 ?

On nous parle de socket, donc de réseau : on va regarder les connexions ouvertes/actives avec linux_netstat

Question 8

Quelle est la commande entière du pid 364?

On peut faire un linux_pslist mais on aura juste le nom du processus, pas la commande entière. Pour cela, j'ai utilisé linux_psaux

Question 9

Quel est l'invocation id de bash?

Je ne savais pas ce qu'était cet "invocation id" ; à mon grand désarroi, personne autour de moi ne savait. Dans la semaine, j'ai balancé toutes les commandes linux volatility connues sur les cheatsheets les plus courantes puis j'ai grep sur le répertoire (oui c'est sale mais ça a payé) et il fallait donc utiliser la commande linux_psenv pour avoir les variables d'environnement.

Question 10

Quel est le PPID of qmgr process?

On peut voir cela facilement avec linux_pslist

III. Crypto

1. Alice au pays des block-ciphers de JcVd

Bon, pas le temps de faire un writeup, on vous file le script fait par AK (elle dort là :( ) :

L'idée principale c'est qu'avec différentes clés, on obtient le même texte chiffré. On peut donc avoir une fonction de déchiffrement indépendante de la clé.

```
ciphertext = '0dede85ca916c63e83eefb630ff1c6802fd38478eb62683ce9b69763dbafca80'
c = ['68636d62627672786f626e656e616771',
     '6870666a7a796f6c67737477696c6772',
     '63796a7476616a72676a7373796f6969',
     '786d6965787569637969716e6d787a',
     '6777766d6e747571656a656b667a6c75',
     '7a6b6c6a636c6f6972747a7371636d65',
     '6f6c7376747a737471637a636e61796d',
     '686e6c746266686b7a6b796c707a6c66',
     '7476646b646a78677571656561726c79',
     '757974736a756165706f747472627479',
     '6c6e6c696d6e70767a72737565766973',
     '6a787a727465756e7362637374747368']
r =  [97, 115, 27, 44, 92, 55, 27, 73, 120, 13, 112, 1]

# Good luck!


c = [int('0x'+x, 16) for x in c]

def _rol(val, bits, bit_size):
    return (val << bits % bit_size) & (2 ** bit_size - 1) | \
           ((val & (2 ** bit_size - 1)) >> (bit_size - (bits % bit_size)))

def _ror(val, bits, bit_size):
    return ((val & (2 ** bit_size - 1)) >> bits % bit_size) | \
           (val << (bit_size - (bits % bit_size)) & (2 ** bit_size - 1))

#__ROR4__ = lambda val, bits: _ror(val, bits, 32)
#__ROL4__ = lambda val, bits: _rol(val, bits, 32)

ROTR = lambda val, bits: _ror(val, bits, 128)
ROTL = lambda val, bits: _rol(val, bits, 128)



def key_schedule(key):
    k = [0]*12
    k[0] = key
```

```
    for i in range(1, 12):
        if (i%2) == 1:
            k[i] = ROTR(k[i-1], r[i-1])
        else:
            k[i] = ROTL(k[i-1], r[i-1])
    return k

def encryption(plaintext, k):
    s = plaintext
    for i in range(11):
        s = s ^ k[i]
        s = ROTR(s, r[i])
        s = s ^ c[i]
    s = s ^ k[11]
    return s

# with known key
def decryption(ciphertext, key):
    s = ciphertext
    k = key_schedule(key)
    s = s ^ k[11]
    for i in range(10,-1,-1):
        s = s ^ c[i]
        s = ROTL(s, r[i])
        s = s ^ k[i]
    return s



key = int('0b'+'1'*128, 2)
part1 = decryption(0x0dede85ca916c63e83eefb630ff1c680, key)
part2 = decryption(0x2fd38478eb62683ce9b69763dbafca80, key)

print(bytes.fromhex(hex(part1)[2:]).decode('utf-8') + bytes.fromhex(hex(part2)[2:]).decode('utf-8'))
# sigsegv{sUr3_r0ll_uR_0wN_crYpt0}
```