Reverse Engineering

Übung 1 - Gruppe 3

Ludwig Karpfinger	Armin Jeleskovic	Valentin Altemeyer
ludwig.karpfinger@hm.edu	a.jeleskovic@hm.edu	valentin.altemeyer@hm.edu

Aufgabe 1 REMnux installieren

Aufgabe 2 REMnux Tool Check

a) Aus welchen Quellen kann ein Tool von REMnux stammen?

Remnux kann wie jede andere Linux-Distro Software aus den angegebenen Quellen installieren. (abgesehen von sonstigen Paketmanagern, wie *snap, flatpack, AppImage*) Folgender Befehl zeigt die Repos an:

```
$ sudo grep -Erh ^deb /etc/apt/sources.list*
```

Es fällt auf, dass ein spezielles Remnux Repo vorhanden ist namens:

http://ppa.launchpad.net/remnux/stable/ubuntu. Diese Repo wurde durch *remnux.sls* hinzugefügt[1]

Die Besonderheit bei Remnux ist, dass der *Remnux Installer* automatisch Software installiert, konfiguriert und aktualisiert. Die Eigenschaften von Software, wie Download-Quelle, Installation Path, Hashnummer, Rechte, Abhängigkeiten und Configs, werden durch sogenannte *state files* bestimmt. Diese Files befinden sich auf GitHub und werden durch den *Remnux Installer* geladen.

[1] https://github.com/REMnux/salt-states/blob/master/remnux/repos/remnux.sls

Remnux nutzt gemäß den Remnux Docs[2] folgende Installationsquellen:

- pip
- gems
- npm
- apt Repos

b) Welche Tools stammen nicht aus Open Source Quellen?

Das Github Repo wurde im Ordner */Documents/* gecloned.

Lösung mit *grep*:

```
$ grep -r --include="*.sls"
> -Eio "source:\s(http|https)://[a-zA-Z0-9./?=_%:-]*"
> /home/remnux/Documents/salt-states/remnux/
> | grep -v "github"
> | cut -d'/' -f10
> | uniq -u
```

Output:

```
snapshots.mitmproxy.org
www.netresec.com
didierstevens.com
www.nowrap.de
bitbucket.org
www.mitec.cz
www.nowrap.de
www.cert.at
www.netresec.com
www.didierstevens.com
radare.mikelloc.com
```

Erklärung:

Es wird *command-chaining* verwendet. Das Tool *grep* sucht mittels *regex* nach allen URLs in allen *.sls* files. Ergebnisse, die den String *github* beinhalten, werden ausgeschlossen. Der *cut* Befehl filtert beim Output die Domains heraus. Der *uniq* Befehl eliminiert alle doppelten Ergebnisse.

[2] `https://docs.remnux.org/behind-the-scenes/technologies/debian-packages`

Lösung mit *Yara*:

```
rule not_open_source
{
    strings:
        $url = /source: (http|https):\/\/[a-zA-Z0-9.\/?=_%:-]*/ nocase
        $gh = "github.com" nocase

    condition:
        $url and not $gh
}
```

Das Github Repo *salt-states-master* wurde in den Ordner */home/remnux/Desktop/uebung1/* geklont.

Die Regel *not_open_source.yara* wurde in der Kommandozeile wie folgt ausgeführt:

```
$ yara -s -r
> not_open_source.yara
> /home/remnux/Desktop/uebung1/salt-states-master
```

Die Regel wird rekursiv auf alle Dateien im geklonten Github Repo angewandt (primär *.sls* Dateien). Es werden alle Dateien gefiltert in denen eine Source-URL (*$url*) enthalten ist, die nicht den String *github.com* (*$gh*) enthält.

Aufgabe 3 File Classification

a) samples.zip - not stripped

bin a

- Es handelt sich um eine AMD 64bit ELF Datei im little-endian Format, die durch GCC kompiliert wurde.
- Clamscan hat keine bekannten Viren entdeckt.
- signsrch hat keine Pattern zur Verschlüsselung, Encoding, Kompression gefunden
- Magic Bytes: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
- Es handelt sich um den Linux cp Command

bin b

- Es handelt sich um eine AMD 64bit ELF Datei im little-endian Format, die durch GCC kompiliert wurde.
- Der *Unix.Tool.Pnscan-8031486-0* Virus wurde gefunden
- signsrch hat keine Pattern zur Verschlüsselung, Encoding, Kompression gefunden
- Es handelt sich um den TCP Portscanner Pnscan

bin c

- Es handelt sich um eine AMD 64bit ELF Datei im little-endian Format, die durch GCC kompiliert wurde.
- Der *Unix.Trojan.Mirai-7100807-0* Virus wurde gefunden
- signsrch hat keine Pattern zur Verschlüsselung, Encoding, Kompression gefunden
- Virus ruft C++ Compiler und Linux Programme, wie Watchdog, auf

b) binaries.zip - stripped

bin a Doppelgänger

Yara file (64 byte Header) == Output *bin 3173*:

```
rule bin_a
{
  strings:
    $id0 = {7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00}
    $id1 = {02 00 3e 00 01 00 00 00 d0 25 40 00 00 00 00 00}
    $id2 = {40 00 00 00 00 00 00 00 80 77 02 00 00 00 00 00}
    $id3 = {00 00 00 00 40 00 38 00 09 00 40 00 1f 00 1c 00}
  condition:
    all of them
}
```

bin b Doppelgänger

Yara file (objdump -drS aus .text) == Output *bin 2723*:

```
rule bin_b
{
  strings:
    $rcx = {48 c7 c1 90 50 40 00}
    $rdi = {48 c7 c7 60 44 40 00}
    $call = {e8 e7 fd ff ff}
  condition:
    all of them
}
```

bin c Doppelgänger

Yara file (objdump -drS aus .text) == Output *bin 1207*:

```
rule bin_c
{
  strings:
    $rcx = {49 c7 c0 80 f3 40 00}
    $rdi = {48 c7 c7 50 c3 40 00}
    $call = {e8 d7 fd ff ff}
  condition:
    all of them
}
```

Aufgabe 4 - Firmware Identifikation

a) Dump Analyse

Die Entropie gibt Aufschluss darüber, dass die Firmware komprimiert und/oder verschlüsselt. Die Schwachstelle bei der XOR-Verschlüsselung ist diese Formel:

```
x^0 = x
```

Es lässt sich somit das Muster - *88 44 A2 D1 68 B4 5A 2D* - finden.

Über den Hexeditor wird die entschlüsselte Datei als *decrypt.bin* abgespeichert.

b) Extrahieren

Das Squashfs Filesystem ist xz compressed und 5739914 bytes groß.

```
$ binwalk decrypt.bin
$ dd if=decrypt.bin skip=1900672 of=linux bs=1
$ unsqushfs linux
```

Mit dem *dd* Befehl wird die Filesystem Partition herausgeschnitten und schließlich mit unsquashfs decompressed.

c) Gerätetyp und Version

Die Analyse der Datei */sbin/dbox init* ergibt, dass es sich wahrscheinlich um einen Router handelt.

Bei der Version von *Squashfs* handelt es sich wahrscheinlich um Version 1.0.0 (hier bin ich mir nicht ganz sicher, da ich keine konkrete Versionsangabe gefunden habe)