volatileEve

Maximilian Li (2941769), Victor Schümmer (2930796)

# **1. Einleitung**

Im Rahmen des P4-Praktikums *Implementierung in Forensik und Mediensicherheit* sollte ein Tool zur automatischen Generierung von Arbeitsspeicher-Abbildern (memory dumps) zu Trainingszwecken implementiert werden. In den dumps sollen ähnlich wie beim *Capture the Flag[[1]](#footnote-2)* (CTF) randomisierte Geheimnisse (flag) versteckt sein, die es zu finden gilt.

Eine derartige flag könnte zum Beispiel folgendermaßen versteckt sein:

In einem Texteditor ist ein eine Datei geöffnet, die scheinbar willkürliche Zeichenfolgen enthält. In einem Browser ist eine Google-Anfrage zum Thema Verschlüsselung geöffnet. Durch Entschlüsselung der Zeichenfolge mit einem Passwort, welches in einer weiteren Textdatei geöffnet ist, ergibt sich das Geheimnis.

Die flags sollen dabei randomisiert sein, damit Teilnehmer die Lösung nicht von anderen erfahren können und es muss außerdem eine (grobe) Lösung generiert werden, die auch den konkreten Wert der flag enthält. Die dumps sollen auf einem Windows(-7)-System generiert werden und kompatibel mit dem Speicherforensik-Tool *volatility* sein, während das zu implementierende Tool auf Linux laufen soll.

Zu diesem Zweck haben wir *volatileEve* (in Anlehnung an das Analyse-Werkzeug *volatility* und den archetypischen Angreifer Eve) entwickelt.

# **2. Konzept**

Da Anwender in der Regel ungern einen dump ihres eigenen Systems verbreiten wollen, und damit mehr Kontrolle über den Speicherinhalt gewährleistet ist, bringt *volatileEve* seine eigene Windows-7-Virtual-Machine mit. Diese wird von *volatileEve* in einen Zustand mit geöffneten Programmen, Registry-Einträgen etc. gebracht und anschließend ein Speicherabbild erzeugt.

Um vielfältige und flexible Szenarien zu ermöglichen, haben wir uns für einen modularen Aufbau entschieden. Auf der Hostmaschine wird dafür aus Anweisungen ein Skript generiert, welches anschließend auf die VM verschoben und dort ausgeführt wird. Sobald das Skript fertig ausgeführt wurde meldet die VM dem Hostsystem, dass sie fertig ist, woraufhin das Hostsystem den Zustand der VM einfriert und ein Speicherabbild erstellt.

Um einen leichteren Einstieg in die Software zu ermöglichen, sind einige vorgefertigte Szenarien verfügbar (Tabelle 1). Die Einschätzung soll den Schwierigkeitsgrad in ein ungefähres Verhältnis setzen und ist nicht absolut zu sehen.

|  |  |  |  |
| --- | --- | --- | --- |
| **Szenario** | **Beschreibung** | **Hilfreiche Tools** | **Einschätzung** |
| autologin.py | Ein Nutzer verwendet überall dasselbe Passwort, welches es herauszufinden gilt. In Windows ist Autologin aktiviert und das Passwort lässt sich somit in der Registry in Klartext finden (easy-mode) oder im lsadump (hard-mode, ungetestet). Ein Hinweis ist die Google-Suche in Internet Explorer. Mit dem Passwort kann das beiliegende Zip entschlüsselt werden. | iehistory, printkey, lsadump | 2/5 easy,  5/5 hard |
| clipboard.py | Es ist ein Textfile geöffnet, indem sich die Flag offensichtlich befinden müsste, doch es wurde anscheinend entfernt. Die Flag befindet sich im Clipboard. | mftparser, filescan, clipboard | 2/5 |
| network.py | Auf dem System läuft Netcat und Wireshark. Die Flag wurde an Netcat auf Port 42042 gesendet. Zwei Möglichkeiten zur Lösung:  - Strings auf den Speicherdump von wireshark anwenden und mit grep nach dem Port suchen  - Mit dem Tool bulk\_extractor lassen sich TCP-Pakete als PCAP-File extrahieren und anschließend mit Wireshark analysieren. | netscan, bulk\_extractor, wireshark | 4/5 |
| password.py | Ein Nutzer verwendet überall dasselbe Passwort, welches es herauszufinden gilt. Eine in Notepad geöffnete Email-Datei deutet auf das Windows-Passwort hin. Dieses lässt sich gehasht finden und ist garantiert so schlecht, dass es sich mit Online-Tools leicht knacken lässt. Mit dem Passwort kann das beiliegende Zip entschlüsselt werden. | hashdump | 3/5 |
| picture.py | Bild in Bildbetrachter geöffnet, extrahierbar z.B. über Analyse der RAW-Daten mit Gimp. Dazu benötigt es die Breite der Bildschirmauflösung,  welche sich erraten lässt. | screenshot, memdump, GIMP | 4/5 |
| terminal\  History.py | Ein Zipfile wurde mit 7z über eine cmd-Konsole erstellt und verschlüsselt. Das Passwort lässt sich im Klartext in der cmd-history finden. | cmdscan | 1/5 |
| marathon.py | Kombination aus vielen der vorherigen Szenarien: wie bei picture ist ein Bild geöffnet, welches man zunächst durch Rekonstruktion des Bildschirminhalts entdecken muss – jedoch lässt sich mit dem Inhalt zunächst nichts anfangen. Es handelt sich um einen Registry-Pfad, auf den dreimal Rotation-Cipher angewendet wurde. Wie weit jeweils rotiert wurde kann man 3 Hinweisen entnehmen: Dem Fenstertitel eines notepad-Fensters (screenshot), dem Clipboard-Inhalt (clipboard) und dem Inhalt einer Zip-Datei, die man zunächst aus dem Speicherdump extrahieren und anschließend mit dem korrekten Passwort entschlüsseln muss (filescan, dumpfiles, cmdscan). Unter dem Registrypfad findet man schließlich die finale Flag. (printkey) | ehistory, printkey, mftparser, filescan, clipboard,  screenshot, memdump, GIMP | 5/5 |
| obfuscation.py | Kombination aus picture und network – statt einer gibt es 15 offene Netzwerkverbindungen, über jede wurde ein Secret übertragen. Welches das echte ist kann man einem geöffneten Bild mit der passenden Port-Nummer entnehmen | netscan, bulk\_extractor, wireshark,  screenshot, memdump, GIMP | 5/5 |

Tabelle 1: Übersicht über vorgefertigte Szenarien

Ein Template zum Erstellen neuer Szenarien findet sich in *challenges/template.py.*

# **3. Implementierung**

## 3.1 volatileEve.sh

Der Einstiegspunkt ist das Bash-Skript *volatileEve.sh*. Dieses setzt zunächst die VM mit *virsh* auf einen „sauberen“ Snapshot zurück, um einen problemlosen Durchlauf zu gewährleisten und zu verhindern, dass sich Restdaten im Speicher ansammeln. Anschließend wird ein Python-Skript aufgerufen, welches für die Erstellung der nötigen Dateien für das entsprechende Szenario verantwortlich ist. Die generierten Dateien werden dann über das *SCP*-Protokoll auf die VM geschoben und die VM neu gestartet.

Nach dem Neustart versetzt sich die VM in den Zustand der gedumpt werden soll und signalisiert durch das Anlegen einer Datei, dass sie fertig ist. Das Bash-Skript prüft via *SSH* alle 2 Sekunden, ob die VM fertig ist, friert dann ihren Zustand ein und generiert mit virsh einen Speicherdump der VM.

Bei Bedarf kann vor dem Einfrieren noch das von python generierte Zusatzskript *additional.sh* ausgeführt werden. Dies wird z.B. von den Netzwerk-Szenarien verwendet um von außen Daten an die VM zu übermitteln.

## 3.2 Das python-Framework

Für die Generierung der benötigten Dateien existieren Wrapper-Klassen für verschiedene Windowsanwendungen, die eine aufs wesentliche beschränkte Konfigurationsmöglichkeit der zu startenden Anwendungen bieten. Aus dem so erstellten Modell wird anschließend automatisch ein *AutoIt*-Skript (\*.au3-File) generiert. *AutoIt* ist eine Software, die Automatisierung von Abläufen auf Windows-Systemen ermöglicht und auf der mitgelieferten VM bereits vorinstalliert ist.

Zusätzlich zu dem *AutoIt*-Skript werden Bilder und weitere benötigte Dateien generiert und im Verzeichnis *output* abgelegt, welches dann auf die VM geschoben wird. In einem weiteren Verzeichnis liegen zudem Dateien, die nicht für die VM bestimmt sind, für manche Szenarien zum Beispiel ein verschlüsseltes Zip-Archiv, dessen Schlüssel im Dump versteckt ist, sowie eine Textdatei mit automatisch generierten Lösungshinweisen.

### 3.3 Die VM

Die Windows-7-VM liegt als *qemu/kvm*-Image vor. Auf der VM sind benötigte Programme wie *Firefox*, *IrfanView* und *7Zip* für die Szenarien, sowie *Cygwin* mit *sshd* und *AutoIt* vorinstalliert. Das automatische Update wurde deaktiviert, um zu verhindern, dass der Update-Prozess mit den Szenarien interferiert.

Nach dem Systemstart wird per Group-Policy automatisch eine Batchdatei gestartet, welche *AutoIt* mit der au3-Datei für das Szenario ausführt.

# **4. Installation**

Zur Verwendung von *volatileEve* muss Python in Version 3.5 oder neuer, sowie *qemu/kvm* und *virsh* auf dem System installiert sein. Für manche Szenarien wird zudem *netcat* und *zip* benötigt.

Zur Installation von VM und Snapshot kann das beiliegende Skript

installVM.sh

ausgeführt werden. Das Speicherabbild ist so konfiguriert, dass es nur mit *KVM* läuft, da ohne die Beschleunigung durch *KVM* kein stabiler Ablauf gewährleistet werden kann.

Anschließend sollte der *SSH*-Key installiert werden, der zur Verbindung genutzt wird. Dazu die Datei *id\_eve* in das *.ssh*-Verzeichnis des ausführenden Nutzers kopieren und folgendes der Datei *config* hinzufügen:

Host <local ip of vm>

IdentityFile <path to id\_eve>

# **5. Verwendung**

Zur Verwendung sollte zunächst die lokale IP-Adresse der VM in Erfahrung gebracht werden.

*volatileEve* kann folgendermaßen über ein Terminal ausgeführt werden:

volatileEve.sh <number of dumps> <local ip of vm> <path to challenge>

Ein Beispielaufruf aus dem Verzeichnis *volatileEve* könnte also so aussehen:

volatileEve.sh 10 192.168.0.1 ./challenges/password.py

1. https://de.wikipedia.org/wiki/Capture\_the\_Flag#Computersicherheit [↑](#footnote-ref-2)