



Technische Hochschule Ingolstadt

Seminararbeit/Whitepaper

Malwareanalyse

angefertigt von

Name:

Dominik Gunther Florian Schlecht

Matrikelnummer:

00032209

Betreuer:

Technische Hochschule Ingolstadt: Prof. Hahndel

Ingolstadt, 17. Juli 2015

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Seminararbeit selbstständig und ohne fremde Hilfe verfasst habe.

Die verwendeten Quellen sowie die verwendeten Hilfsmittel sind vollständig angegeben. Wörtlich übernommene Textteile, übernommene Bilder und Zeichnungen sind in jedem Einzelfall kenntlich gemacht.



Ingolstadt, 17. Juli 2015

Inhaltsverzeichnis

1. Einleitung	1
2. Verwendete Tools und Infrastruktur	1
2.1. Physikalisches Betriebssystem	1
2.2. Virtualisierungslösung	1
2.3. Virtuelles Betriebssystem	1
2.4. Weitere Tools	2
2.4.1. Resource Hacker	2
2.4.2. Dependency Walker	2
2.4.3. PEView	2
2.4.4. RegShot	2
2.4.5. Process Monitor	2
2.4.6. Process Explorer	3
2.4.7. Autoruns	3
2.4.8. IDA Pro Free	3
2.5. Webseiten	3
2.5.1. Virustotal	3
2.5.2. Malwr	3
2.5.3. Immunity Debugger	4
3. Infektionsweg	4
4. Statische Analyse	4
4.1. Identifikation des Samples	4
4.2. Virustotal	5
4.3. Resource Hacker	6
4.4. Dependency Walker	8
4.5. PEView	9
4.6. IDA Pro Free	10
4.7. Fazit zur statischen Analyse	11
5. Dynamische Analyse	11
5.1. Malwr	12
5.2. Live-Analyse	12
5.3. Immunity Debugger	14
5.4. Weiter Analyse	15
5.5. Fazit zur dynamischen Analyse	17
6. Fazit	17
A. Appendix	18
A.1. Version Info	18
A.2. Abbildungen	19

1. Einleitung

Im Rahmen dieser Seminararbeit wird im Folgenden ein Malware untersucht, die per Spam verbreitet wurde. Dabei wird die Malware vorerst statisch und anschließend dynamisch analysiert. Die Einzelergebnisse sind jeweils den Abschnitten 4.7 und 5.5 zu entnehmen.

2. Verwendete Tools und Infrastruktur

In diesem Abschnitt werden die verwendeten Tools sowie die Infrastruktur dargestellt.

2.1. Physikalisches Betriebssystem

Als Grundsystem wurde ein *Linux*-System verwendet. Dies hat den Vorteil, dass ein Großteil der sich derzeit im Umlauf befindlichen Malware für *Windows* konzipiert ist und somit nur ein geringes Risiko besteht, dass sich das Grundsystem mit der Malware infiziert. Als unkompliziertes, wandelbares und trotzdem hoch modifizierbares System wurde *Debian 8* gewählt. Versuche mit zum Beispiel *Gentoo* zeigten Probleme mit der verwendeten Virtualisierungslösung.

2.2. Virtualisierungslösung

Es gibt viele Vorteile für die Nutzung einer Virtualisierungslösung bei der Malwareanalyse, jedoch auch Nachteile. Vorteilhaft ist vor allem das Erstellen von sogenannten Snapshots, welche einen bestimmten Zustand eines Systems aufzeichnen und es möglich machen, diesen später wieder herzustellen. Zudem wird das Host-System vor der Malware geschützt. Ein Nachteil ist, dass moderne Malware immer häufiger überprüft, ob sie in einer virtuellen Umgebung ausgeführt wird. Falls ja, werden oft andere Funktionen ausgeführt, um die ursprüngliche Funktion zu verschleiern. Insgesamt überwiegen aber die Vorteile den Nachteil. Falls die Malware auf die virtuelle Umgebung prüft, muss getestet werden, ob die Prüffunktion über den Disassembler oder Debugger deaktiviert oder umgangen werden kann.

Als Virtualisierungslösung wurde *VMWare Workstation 11* genutzt. Diese bietet gerade im Bereich der Netzwerk-Modifikation weitere Möglichkeiten gegenüber der kostenlosen Variante *Virtualbox* von *Oracle*. Die Workstation kann von der offiziellen Webseite heruntergeladen und für 30 Tage kostenlos getestet werden.

2.3. Virtuelles Betriebssystem

Als virtuelles Betriebssystem wurde *Windows 7 Pro* verwendet. *Windows 7* bietet sich an, da es derzeit eines der meist verbreiteten Betriebssysteme ist und Malware meistens für *Windows* konzipiert ist.

Zudem wurde 32-bit als Architektur gewählt, um die Kompatibilität mit Tools wie *Cuckoo-Sandbox* sowie mit der aktuellen Malware sicherzustellen. Außerdem wurden sowohl das *UAC*, Updates sowie die Firewall deaktiviert, um der Malware eine möglichst einfache Umgebung zu bieten. Die *VMWare*-Tools wurden absichtlich nicht installiert, da dies einer Malware eine sehr einfache Möglichkeit bieten würde, die Umgebung zu erkennen.

Diese Konfiguration wird als Grund-Image verwendet.

2.4. Weitere Tools

Neben der oben genannten Umgebung wurden zudem folgende Tools verwendet, um die Malware zu analysieren.

2.4.1. Resource Hacker

Resource Hacker ist ein Tool für *Windows*, welches Bestandteile einer ausführbaren Datei aufgliedert und darstellt. So können zum Beispiel schnell Icons, Menü-Elemente, Strings oder ähnliches identifiziert und extrahiert werden.¹

2.4.2. Dependency Walker

Der *Dependency Walker* (Version 2.2)² ist eine freie Software, welche 32-bit und 64-bit *Windows*-Anwendungen (*exe*, *DLLs* und weitere) analysiert und Abhängigkeiten darstellt. Für jede Abhängigkeit werden Import und Export-Funktionen angezeigt. Dies erlaubt eine grobe Einschätzung, welche Funktionalitäten das Programm bietet.

2.4.3. PEView

PEView zeigt Informationen über ausführbare Dateien unter *Windows* an (welche im *PE/-COFF*-Format vorliegen, was unter *Windows* der Standard ist). Hier können verschiedene Informationen wie die Import- und Exporttabellen, Textsegmente oder ähnliches untersucht werden. Da *PEView* diese jedoch nur sehr rudimentär aufgliedert, kann die manuelle Analyse einige Zeit dauern.

2.4.4. RegShot

Das Programm *RegShot* erlaubt es, den derzeitigen Zustand der *Windows-Registry* zu sichern und später mit einem anderen Stand zu vergleichen. Dies ist für die dynamische Analyse sehr praktisch, da man schnell die, zum Beispiel durch Malware, veränderten Einträge sehr schnell identifizieren kann. Diese geben wiederum wichtige Informationen über die Funktionsweise der Malware oder liefern *Indicators of Compromise* (IOC). Eine Analyse der Registry ohne *RegShot* ist möglich, wäre aber sehr aufwendig.

2.4.5. Process Monitor

Das Tool *Process Monitor* ist Teil der von *Microsoft* veröffentlichten *Sysinternal-Suite*³. Es zeigt Zugriffe (unter anderem Festplatte, Netzwerk, Registry) aller aktuell laufenden Programme auf und erlaubt die Filterung dieser. Ebenso kann es bereits bei Systemstart mit der Protokollierung beginnen und das Verhalten von Programme aufzeichnen.

¹<http://www.angusj.com/resourcehacker/>

²<http://www.dependencywalker.com/>

³<https://technet.microsoft.com/en-us/sysinternals/bb842062.aspx>

2.4.6. Process Explorer

Das Tool *Process Explorer* ist ebenfalls Teil der *Sysinternal-Suite*. Es ist eine Art erweiterter *Windows Task-Manager* und bietet Funktionen wie das Überprüfen der Signaturen von laufenden Prozessen oder den Upload derer Hashes zu *Virustotal* (siehe 2.5.1). Ebenso kann es alle von einem Prozess geladenen *DLLs* anzeigen.

2.4.7. Autoruns

Autoruns ist ein weiteres Tool aus der *Sysinternal-Suite*. Es listet *Autostart*-Einträge des Systems aus verschiedensten Quellen auf und erlaubt so zu prüfen, ob ein unerwünschtes Programm bei Systemstart ausgeführt wird. Es kann ebenfalls dazu verwendet werden, die unerwünschten Einträge zu deaktivieren.

2.4.8. IDA Pro Free

Als Disassembler wurde *IDA PRO Free* (Version 5.0) verwendet. Diese Version reicht für grundlegende Analysen, jedoch sind die analysierbaren Anwendungen auf 32-bit-Anwendungen begrenzt. Da Malware jedoch so konzipiert ist, dass ein möglichst breites Spektrum an Geräten angegriffen werden kann, fällt diese Einschränkung zumeist nicht ins Gewicht. Um mit *IDA PRO Free* 64-bit Malware zu analysieren, ist eine kostenpflichtige Version notwendig. Als eine kostengünstigere Alternative zu *IDA PRO* kann *Hopper*⁴ in Betracht gezogen werden. *Hopper* gibt es ebenfalls in einer kostenfreien Version. Die Stärke liegt jedoch in der kostenpflichtigen Lizenz, welche ähnliche Features bietet wie *IDA PRO*, jedoch wesentlich billiger und somit auch für Privatpersonen erschwinglich ist. Zudem bietet *Hopper* den Vorteil, dass es vorgefertigte Versionen für *OS X* und *Linux* gibt. *IDA PRO (Free)* liegt nur für *Windows* vor, ist jedoch über *Wine* relativ einfach auf *Linux* installierbar.

2.5. Webseiten

Neben Tools auf dem Rechner können ebenfalls Webseiten bei der Analyse von Malware helfen. Folgend sind zwei dieser Seiten aufgeführt.

2.5.1. Virustotal

Die Webseite *Virustotal* ermöglicht die Analyse von Dateien oder URLs, jeweils im Original oder als Hash. Nach dem Upload wird die Datei/URL von einer Vielzahl von bekannten Virens Scanner gescannt und die Ergebnisse zurück gegeben. Falls die Datei/URL vor nicht allzu langer Zeit bereits gescannt wurde, wird dieses Ergebnis angezeigt. Unternehmen können sich ebenso weitere Funktionen kaufen, um breitere Analysen auf der Datenbasis von *Virustotal* durchzuführen.

2.5.2. Malwr

Malwr basiert auf der *Cuckoo-Sandbox* und stellt eine automatisierte Lösung zur dynamischen Analyse von Malware dar. Dabei werden unter anderen Screenshots erstellt, Netzwerkverbindungen aufgezeichnet oder Änderungen in der Registry festgehalten.

⁴<http://www.hopperapp.com/>

2.5.3. Immunity Debugger

Der *Immunity Debugger*⁵ ist ein kostenloser Debugger für *Windows*. Er zeichnet sich vor allem durch den hohen Funktionsumfang sowie um die Möglichkeit, den Debugger durch *Python-Skripte* zu erweitern, aus.

3. Infektionsweg

In dieser Arbeit wird eine Malware, die über E-Mail verteilt wurde, analysiert. Die Mail ist im Anhang, Abbildung 13, dargestellt. Das Design ist am Speditionsunternehmen *UPS* orientiert⁶. Auch der Absender *UPS express [mailto:vorname_nachname@smtp.acenet.hu]* sowie der Titel *UPS - Zustellbenachrichtigung, Kontrollnummer 635M6265282635* sind passend gewählt. Öffnet man den Link in der E-Mail wird eine Zip-Datei Namens *ups_webtracking_1S63A0003659818362.zip* heruntergeladen, welche die hier analysierte Malware enthält.

4. Statische Analyse

Die statische Analyse beschäftigt sich mit der Untersuchung der Malware, ohne diese auszuführen. Dies geschieht meist über Tools, welche die im Binary enthaltenen Informationen auslesen. Im folgenden werden verschiedene Methoden aufgezeigt, um die Malware statisch zu analysieren, ohne das Gastsystem einem Risiko zu unterziehen.

4.1. Identifikation des Samples

Zu aller erst wird das Zip-File aus der E-Mail gesichert und ein Hash erstellt, um das Sample in Zukunft auch unter einem anderen Namen identifizieren zu können. Der Hash wurde hier über die Software *winMD5Free*⁷ in der Version 1.20 genutzt. Alternativ hätte man unter Linux das Tool *md5sum* nutzen können

```
ups_webtracking_1S63A0003659818362.zip
--> 81397589ad7bf0a7faecf977644b1486
```

Die Datei wird im folgenden Text kurz *8139* genannt. Um zu verifizieren, dass es sich bei der Datei *8139* wirklich um ein Archiv handelt, wendet man in einem *Linux*-System das *File*-Kommando an. Wie in der Abbildung 3 zu sehen ist, bestätigt das *File*-Kommando den Verdacht, dass es sich hier um ein Archiv handelt. Das *File*-Kommando identifiziert Dateien anhand derer Header und gibt somit meist einen verlässlichen ersten Eindruck, von welchem Typ die Datei ist. In anderen Gebieten, wie zum Beispiel der Steganographie, sollte man sich jedoch nicht uneingeschränkt darauf verlassen, sondern die Datei manuell untersuchen. Da nun klar ist, dass es sich bei der Datei *8139* wirklich um ein Archiv handelt, kann dieses nun entpackt werden. Hier wurde *7Zip*⁸ unter *Windows* verwendet. Alternativ hätte man hier das *Unzip*-Kommando⁹ unter *Linux* nutzen können.

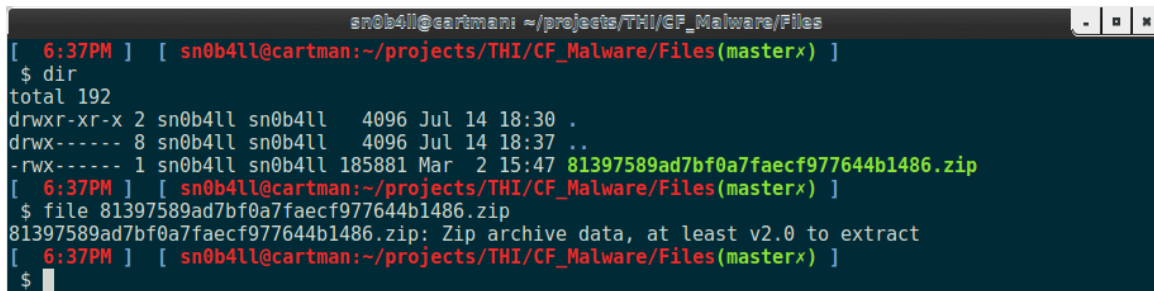
⁵<http://debugger.immunityinc.com/>

⁶<http://www.ups.com/de>

⁷<http://winmd5.com/>

⁸<http://www.7-zip.de/>

⁹<http://www.info-zip.org/mans/unzip.html>



```

sn0b4ll@cartman: ~/projects/THI/CF_Malware/Files
[ 6:37PM ] [ sn0b4ll@cartman:~/projects/THI/CF_Malware/Files(masterx) ]
$ dir
total 192
drwxr-xr-x 2 sn0b4ll sn0b4ll 4096 Jul 14 18:30 .
drwx----- 8 sn0b4ll sn0b4ll 4096 Jul 14 18:37 ..
-rwx----- 1 sn0b4ll sn0b4ll 185881 Mar 2 15:47 81397589ad7bf0a7faecf977644b1486.zip
[ 6:37PM ] [ sn0b4ll@cartman:~/projects/THI/CF_Malware/Files(masterx) ]
$ file 81397589ad7bf0a7faecf977644b1486.zip
81397589ad7bf0a7faecf977644b1486.zip: Zip archive data, at least v2.0 to extract
[ 6:37PM ] [ sn0b4ll@cartman:~/projects/THI/CF_Malware/Files(masterx) ]
$

```

Abbildung 1: Anwendung des File-Kommandos auf die Datei *8139*

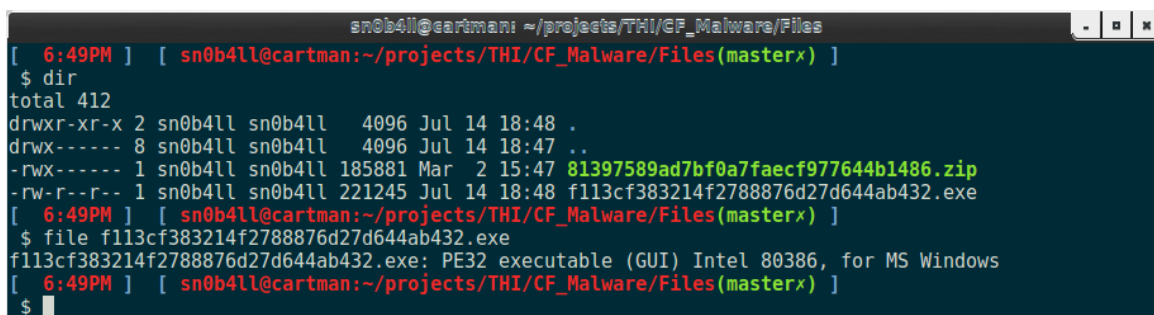
Als Ergebnis bekommt man eine Datei mit einem zur Spam-Mail passenden Namen. Von der Datei wird wiederum der Hash (MD5) entnommen.

```

ups_webtracking_1S63[...]62_0003947_de_2015_02_tracknum_09234728.exe
--> f113cf383214f2788876d27d644ab432

```

Die Datei wird im Folgenden unter dem Namen *f113* geführt. Aus der Endung *.exe* lässt sich auf eine ausführbare Windows-Datei schließen. Die Datei *f113* wird nun ebenfalls mit dem *File*-Kommando untersucht, welches den Verdacht bestätigt (Abbildung 2).



```

sn0b4ll@cartman: ~/projects/THI/CF_Malware/Files
[ 6:49PM ] [ sn0b4ll@cartman:~/projects/THI/CF_Malware/Files(masterx) ]
$ dir
total 412
drwxr-xr-x 2 sn0b4ll sn0b4ll 4096 Jul 14 18:48 .
drwx----- 8 sn0b4ll sn0b4ll 4096 Jul 14 18:47 ..
-rwx----- 1 sn0b4ll sn0b4ll 185881 Mar 2 15:47 81397589ad7bf0a7faecf977644b1486.zip
-rw-r--r-- 1 sn0b4ll sn0b4ll 221245 Jul 14 18:48 f113cf383214f2788876d27d644ab432.exe
[ 6:49PM ] [ sn0b4ll@cartman:~/projects/THI/CF_Malware/Files(masterx) ]
$ file f113cf383214f2788876d27d644ab432.exe
f113cf383214f2788876d27d644ab432.exe: PE32 executable (GUI) Intel 80386, for MS Windows
[ 6:49PM ] [ sn0b4ll@cartman:~/projects/THI/CF_Malware/Files(masterx) ]
$

```

Abbildung 2: Anwendung des File-Kommandos auf die Datei *f113*

4.2. Virustotal

Im Anschluss kann das Sample *f113* auf *Virustotal* (siehe Sektion 2.5.1) hochgeladen werden. Das Ergebnis ist, dass bereits 44 von 56 Virensclannern das Sample erkennen. Als Malware-Familie (nach *NOD32*) wird *Win32/Emotet.AD* angegeben. Ebenso werden alternative Dateinamen wie

- Telekom_Rechnung_2015.02_de.04349_AIEO_POP_MAIL_W5.5[...]H.exe
- dhl_paket_de_003407293054131348371_02.2015_HD_38300_J[...]MAIL.exe

angegeben. Daraus lässt sich schließen, dass die Malware bereits in anderen Spam-Angriffen Anwendung fand. Es werden ebenfalls weiterführende Informationen zum Sample geliefert, welche jedoch im Folgenden über lokale Tools erarbeitet werden. Dies hat den Hintergrund, dass, falls man Opfer eines gezielten Angriffs ist, das Sample nicht umgehend bei Virustotal

veröffentlicht werden sollte. Ein Angreifer könnte regelmäßig prüfen, ob das Sample Virustotal bereits bekannt ist. Ist das Sample online, weiß der Angreifer, dass man einen ersten Verdachtsmoment hat und beginnt eventuell damit, seine Spuren zu verwischen.

4.3. Resource Hacker

Nun findet das *Resource Hacker* (siehe Abschnitt 2.4.1) Anwendung. Hierzu öffnet man die Anwendung und lädt anschließend über *File* → *Open* die Datei *f113*. In den jeweiligen

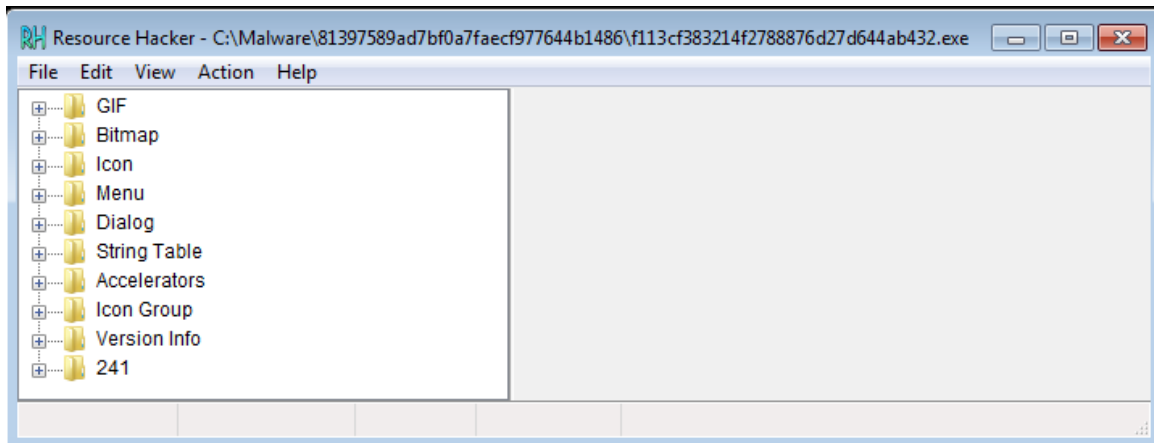


Abbildung 3: Anwendung des Tools *Resource Hacker* auf die Datei *f113*

Kategorien lassen sich verschiedene Informationen darstellen. Insbesondere drei Informationen fallen bei der Analyse des Files *f113* auf.

- 1. Version Info:** Im Feld *Version Info* wird die Versions-Information des bekannten First-Person-Shooters *Half-Life* von Valve¹⁰ verwendet. Der volle Text ist im Anhang unter A.1 zu finden. Diese Maßnahme wird vermutlich verwendet, um die Malware zu tarnen. Durch die kleinste Veränderung von Inhalten wird der Hash der Datei verändert, wodurch primitive Vergleiche das Sample nicht mehr erkennen würden. Zudem können Anti-Viren-Hersteller schlecht den Text als Pattern hinterlegen, da sonst in Zukunft die ausführbare Datei des Spiels *Half-Life* als Malware erkannt werden könnte. Dies wäre ein sogenanntes *False-Positive*, also eine Datei, welche keine Malware ist aber als solche erkannt wird. Im Allgemeinen versuchen Anti-Viren-Hersteller dies zu vermeiden, da sonst die Akzeptanz der Produkte sinkt.
- 2. Icon:** Im Feld *Icon* kann das in der Abbildung 4 gezeigte Icon gefunden werden. Es zeigt das Logo des *Akrobat Readers*¹¹. Es wurde vermutlich gewählt, um den Nutzer den Eindruck zu vermitteln, es würde sich um eine *PDF*-Datei handeln, welche zumeist mit dem *Akrobat Reader* geöffnet werden. Durch die weite Bekanntheit des *Akrobat Readers* erreichen die Malware-Ersteller so eine höhere Quote der Fälle, in welchen die Malware geöffnet wird.

¹⁰<http://store.steampowered.com/app/70>

¹¹<https://get.adobe.com/de/reader/otherversions/>

Abbildung 4: Icon der Datei *f113*

3. Menü-Bestandteile: In den Feldern *Bitmap*, *Menu* und *Dialog* können Bestandteile eines Menüs gefunden werden. Die Abbildung 5 zeigt das unter *Bitmap* zu findende Bild, welches eine herkömmliche Werkzeugleiste eines Programms zeigt. Unter dem Punkt *Menu* kann zudem ein volles Menü mit asiatischen Schriftzeichen gefunden werden. *Resource Hacker* erlaubt es hier, sich das Menü, wie es im Programm eingebettet wäre, zu inspizieren. Es zeigt sich ein normales Menü, aus dem man, auch ohne die Schriftzeichen zu verstehen, normale Felder wie Öffnen, Speichern oder Schließen ableiten kann (siehe Abbildung 6). Die Beweggründe, warum das Menü enthalten ist, können unterschiedlich sein. Es könnte direkt vom Programm stammen, sodass man zum Beispiel durch eine bestimmte Eingabe eine Menü öffnen kann. Alternativ kann es auch nur ein weiteres Element der Malware sein, um das Sample schwer von Pattern erfassbar zu machen. Eine andere Alternative wäre, dass die Malware-Ersteller hier Analysten zu falschen Schlussfolgerungen bringen wollen, zum Beispiel, dass die Malware aus dem asiatischen Raum stammt. Daher sollte mit dieser Information sehr behutsam umgegangen werden und die Thesen später bei einem tieferen Wissen nochmals geprüft werden. Zudem ist unter dem Feld *Dialog* ein kleiner Dialog zu finden, welcher in der Abbildung 7 dargestellt ist.

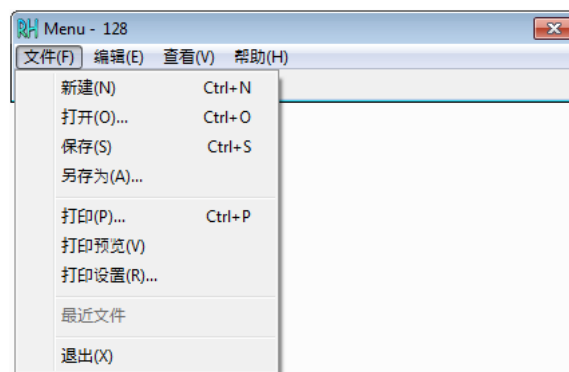
Abbildung 5: Enthaltene Bitmap der Datei *f113*Abbildung 6: Enthaltene Menü der Datei *f113*

Abbildung 7: Enthaltener Dialog der Datei *f113*

4.4. Dependency Walker

Als nächstes wird die Datei *f113* mit der Software *Dependency Walker* (siehe Beschreibung 2.4.2) untersucht. Dazu öffnet man den *Dependency Walker* und öffnet über *File* → *Open* die Datei *f113*. Daraufhin zeigt *Dependency Walker* alle verwendeten *DLLs*, sowie deren genutzte Funktionen an. Das Ergebnis ist im Anhang in der Abbildung 14 zu sehen. Aus den aufgerufenen Funktionen der *Kernel32.dll* kann man auf die Funktionsweise der Malware schließen. Die interessanten Funktionen sind im Folgenden aufgeführt:

CompareStringA: Es wird ein String-Vergleich angestellt. Dies könnte in vielen Szenarien zum Einsatz kommen. Möglichkeiten wären hier ein Vergleich um zu erkennen, ob die Malware in einer VM ausgeführt wird oder ein einfacher Vergleich, in welchem Ordner sich die aktuell ausgeführte Datei befindet.

CreateFileW: Es wird eine Datei auf das Dateisystem geschrieben. Dies könnte zum Beispiel eine Datei sein, welche beim Systemstart ausgeführt wird.

FindNextFileA: Es wird nach einer Datei gesucht. Dies kann wiederum der Erkennung einer VM dienen, falls zum Beispiel nach einer bestimmten *DLL* gesucht wird, welche nur von *VMWare* (oder *VirtualBox*) verwendet wird.

GetModuleFileNameW: Es wird entweder der Pfad zu einem Modul oder, falls kein Parameter übergeben wurde, die Pfad zur ausgeführten Anwendung zurückgegeben. Dies könnte zusammen mit *CompareStringA* dazu genutzt werden, die aktuelle Position der Datei *f113* abzufragen und ausgehend davon verschiedene Aktionen auszuführen.

GetModuleHandleW: Über diese Funktion kann auf eine bereits geladene *DLL* zugegriffen werden. Gelingt es der Malware eine *DLL* in den Kontext zu laden, könnte diese damit benutzt werden.

GetStartupInfoW: Es wird der *StartupInfo*-Vektor abgefragt. Dieser umfasst unter anderem den Rechnernamen, den Titel der Anwendung sowie andere Elemente des Environments.¹²

HeapSize und VirtualAlloc: *HeapSize* gibt die aktuelle Größe des *Heaps* an. *VirtualAlloc* reserviert Speicherbereiche. Das Besondere an *VirtualAlloc* ist, dass für die reservierten

¹²[https://msdn.microsoft.com/en-us/library/windows/desktop/ms686331\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms686331(v=vs.85).aspx)

Speicherbereiche *DEP* deaktiviert werden kann.¹³ Dies könnte ein Hinweis dafür sein, dass die Malware versucht (eventuell dynamisch nachgeladenen) *Shellcode* auszuführen. Die *HeapSize* kann dazu eine wichtige Information sein (wird jedoch nicht unbedingt benötigt).

Ob durch die Malware alle festgestellten Funktionen ausgeführt werden, lässt sich in der statischen Analyse nur sehr schwer feststellen. Zudem kann der Ersteller der Malware zu jeder Zeit nicht wirklich benötigte Funktionen einschleusen, um die Analyse zu erschweren.

4.5. PEView

Als nächstes wird das Tool *PEView* (siehe Beschreibung 2.4.3) angewendet. Dazu wird *PEView* gestartet und die Datei *f113* ausgewählt. Daraufhin werden im *PEView*-Fenster die verschiedenen Sektionen der ausführbaren Datei angezeigt, wie zu sehen in Abbildung 8. Auffällig waren dabei:

SECTION .data: Enthält den Text "Rocal AppWizard-Generated Application". Dieser Text wurde bereits in vielen anderen Malware-Samples gefunden¹⁴ und wird auch später noch in der Registry auftauchen (siehe Abschnitt 5.2.2).

SECTION .rsrc: Enthält den Text "PADDINGXPADDINGXPADDINGX" vielfach. Es lassen sich 2 Funktionen ableiten. Entweder wurde das Padding genutzt, um die ausführbare Datei auf eine korrekte (vom PE/COFF-Format geforderte) Länge zu strecken oder als Padding bei einem möglichen Exploit (siehe *VirtualAlloc* in Sektion 4.4).

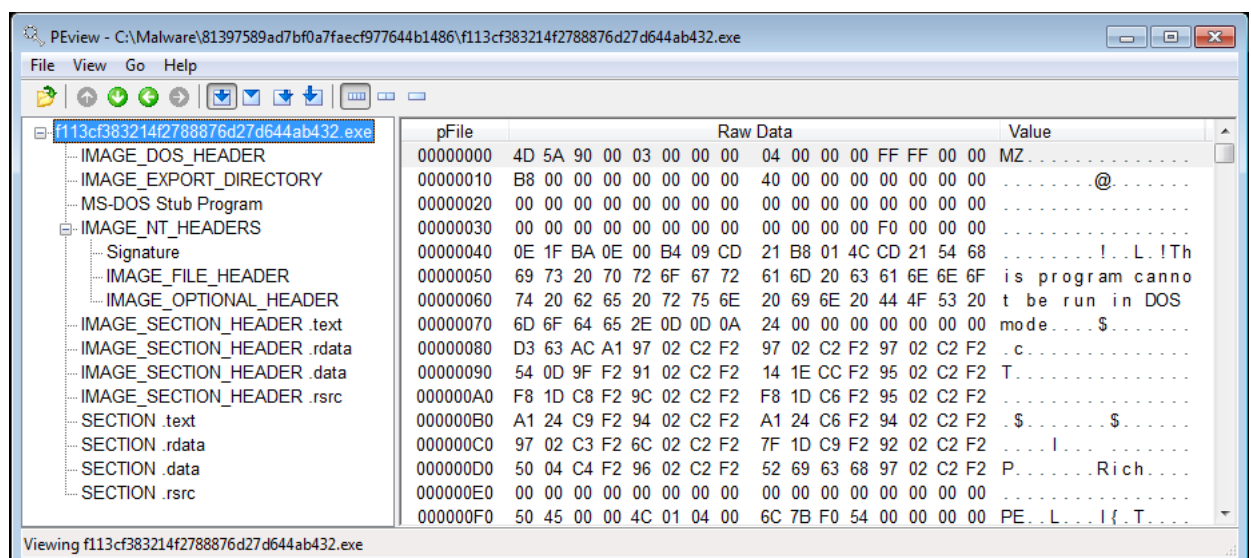


Abbildung 8: Aufruf der Datei *f113* mit *PEView*

Leider liefert *PEView* hier keine weiteren, neuen Informationen zur Datei *f113*.

¹³<http://0xdabbad00.com/2012/12/07/dep-data-execution-prevention-explanation/>

¹⁴<https://www.google.de/search?q=Rocal+AppWizard>

4.6. IDA Pro Free

Als letzten Schritt der statischen Analyse wird die Datei *f113* mit dem Disassembler *IDA PRO Free* (siehe Beschreibung 2.4.8) untersucht. Dazu öffnen wir die Datei mit dem Programm. Wirft man einen Blick auf die enthaltenen Funktionen, so erkennt man viele, welche für graphische Oberflächen nützlich sind. Beispiele sind im Anhang in der Abbildung 15 aufgezeigt. Auffällig ist, dass viele der Funktionen anscheinend nicht aufgerufen werden oder nur im *.rdata*-Segment verankert sind. Da diese Funktionen im Disassembler nicht genutzt werden, heißt jedoch nicht, dass das Programm in der Ausführung diese nicht dynamisch zur Laufzeit nutzt.

Die Analyse, beginnend mit der *start*-Funktion, welche durch den Eintrittspunkt in das Programm ermittelt wird, ergibt folgendes:

Funktion *start*: Es wird zu Anfang die Unterfunktion *sub_402718* aufgerufen, welche im weiteren Verlauf genauer analysiert wird. Nach dem Aufruf würde das Programm in Unterfunktion *loc_402C05* springen, in welcher ein neuer Thread erstellt wird. Danach endet das Programm.

Funktion *sub_402718*: Die Funktion *sub_402718* ist wesentlich komplexer (siehe Abbildung 9). Aktionen in der Klasse sind:

1. Es wird der Applikationstyp über *set_app_type*¹⁵ gesetzt.
2. Anschließend werden einige Variablen und Vektoren initialisiert. Im Anschluss wird ein Sprung genommen, welcher dem Autor auf den ersten Blick nicht ersichtlich ist

```
call    _initterm
add     esp, 24h
mov     eax, ds:_wcmdln
mov     esi, [eax]
cmp     esi, ebx
jnz     short contWithSub
```

Fällt der Vergleich negativ aus, springt das Programm über eine kurze Unteroutine zurück in die *start*-Funktion. Ist der Vergleich positiv, fährt die Funktion *sub_402718* fort und wird, wie sich später zeigt, nicht mehr in die *start*-Funktionen zurückkehren.

3. Im Anschluss fährt das Programm über verschiedene Sub-Funktionen fort, welche vermutlich eine Schleife beinhalten und weitere Variablen initialisiert (in der Abbildung 9 knapp nach der Hälfte zu sehen).
4. Daraufhin wird die *GetStartupInfoW* geladen und verglichen. Der Vergleich bestimmt jedoch nur, ob *0Ah* oder der Wert an der Adresse *ebp+StartupInfo.wShowWindow* als Parameter für die nächste Funktion genutzt wird. Mit welchem Wert die *StartupInfo* verglichen wird, ist leider nicht ersichtlich.
5. Im weiteren Verlauf wird die Funktion *GetModuleHandleW* aufgerufen, welche den *Handle* auf ein Modul lädt. Welches dies genau ist, ist leider in *IDA* nicht ersichtlich.

¹⁵<https://msdn.microsoft.com/en-us/library/ff770596.aspx>

6. Im Anschluss wird die (selbst-benannte) Funktion *openWindow* aufgerufen, welche intern die Funktion *AfxWinMain()* aufruft. Dadurch wird sehr wahrscheinlich ein Fenster erstellt. Die Funktion kehrt anschließend in die Funktion *sub_402718* zurück.
7. Dort wird als letztes die Funktion *exit* aufgerufen, welche das Programm beendet.

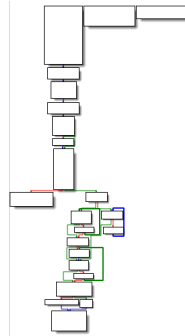


Abbildung 9: Funktion *sub_402718* der Datei *f113*, Ansicht in IDA Graphview

Anmerkung: Die statische Disassemblierung gibt eine grobe Einsicht in das Programm, kann aber niemals allen Sprüngen folgen. Es werden von Malware-Entwicklern oft Techniken genutzt, welche die statische Disassemblierung wesentlich erschweren (dynamisches Laden von Programmteilen, Verschlüsselung kritischer Abschnitte oder Ähnliches). Auch hier wurde sehr sicher nicht der volle Funktionsumfang erfasst, da der Thread nur mit wesentlich erhöhtem Aufwand nachverfolgt werden kann.

4.7. Fazit zur statischen Analyse

Die statische Analyse gab folgende Informationen

- Die Malware versucht das Opfer glauben zu lassen, sie wäre ein PDF. (siehe 4.3)
- Die Malware benutzt die Daten eines Videospiels als Versions-Informationen. (siehe 4.3)
- Die Malware beinhaltet Menü-Elemente. (siehe 4.3)
- Die Malware stellt Vergleiche an und erstellt eine oder mehrere Dateien. Ebenso führt sie eventuell Shellcode aus. (siehe 4.4)

5. Dynamische Analyse

Die dynamische Analyse gewinnt, im Gegensatz zur statischen Analyse, über die Ausführung der Malware weitere Informationen. So können Verhaltensmuster, welche in der statischen Analyse nur grob vorherzusehen sind, beobachtet und weiter analysiert werden. Im Folgenden werden verschiedene Möglichkeiten und Tools zur dynamischen Analyse vorgestellt.

5.1. Malwr

Als erster Punkt der statischen Analyse wird die Datei *f113* auf der Seite *Malwr* (siehe Abschnitt 2.5.2) analysiert.

Dabei weist *Malwr* auf folgende Eigenheiten hin:

- File has been identified by at least one AntiVirus on VirusTotal as malicious
- The binary likely contains encrypted or compressed data.
- Tries to unhook Windows functions monitored by Cuckoo
- Executed a process and injected code into it, probably while unpacking

Demnach enthält die Datei verschlüsselte Daten, versucht aktiv Sandboxen zu vermeiden und lädt „on-the-fly“-Code nach. Dies erklärt, warum die Analyse bisher und im weiteren Verlauf nicht vollständig durchlaufen werden kann. Dem vollständigen Report¹⁶ können weitere Daten entnommen werden. Aus den gleichen Gründen wie bei *Virustotal*, siehe 4.2, wird nun jedoch ein Großteil der Informationen lokal gesammelt.

5.2. Live-Analyse

Als Nächstes folgt die lokale Live-Analyse. Dazu sollte zuallererst ein Snapshot der Virtuellen Maschine erstellt werden, damit man die Maschine nach der Ausführung und Analyse wieder auf einen nicht-kompromittierten Zustand zurücksetzen kann. Im Anschluss werden vor der Analyse folgende Programme gestartet:

- RegShot
- Process Monitor
- Wireshark
- Process Explorer

Nachdem alle Programme gestartet sind und aufzeichnen, kann die Malware (Datei *f113*) ausgeführt werden. Im Folgenden sind die Ergebnisse aufgezeigt.

5.2.1. Grundlegende Beobachtungen

Nach der Ausführung des Programms öffnet sich kein Fenster. Dafür wird die Datei *f113* im ursprünglichen Ordner gelöscht. Es kann keine weitere Aktion beobachtet werden.

5.2.2. RegShot

RegShot liefert folgendes Log (gekürzt):

¹⁶<https://malwr.com/analysis/MDIwOTgyODFmOTFiNDU1NDk3ZWFiMzc1NzAwYWI2MGU/>

```

Keys added:9
HKU\S-1-5-21-12[...]\Software\Microsoft\Multimedia\Audio
HKU\S-1-5-21-12[...]\Software\Microsoft\Multimedia\Audio\Box
HKU\S-1-5-21-12[...]\Software\Microsoft\Multimedia\Audio\Box\
    ba8c80406
HKU\S-1-5-21-12[...]\Software\Microsoft\Multimedia\Audio\Box\
    ba8c80407
HKU\S-1-5-21-12[...]\Software\Microsoft\Multimedia\Audio\Box\
    ba8c80408
HKU\S-1-5-21-12[...]\Software\Rocal AppWizard-Generated
    Applications
HKU\S-1-5-21-12[...]\Software\Rocal AppWizard-Generated
    Applications\BG
HKU\S-1-5-21-12[...]\Software\Rocal AppWizard-Generated
    Applications\BG\Recent File List
HKU\S-1-5-21-12[...]\Software\Rocal AppWizard-Generated
    Applications\BG\Settings

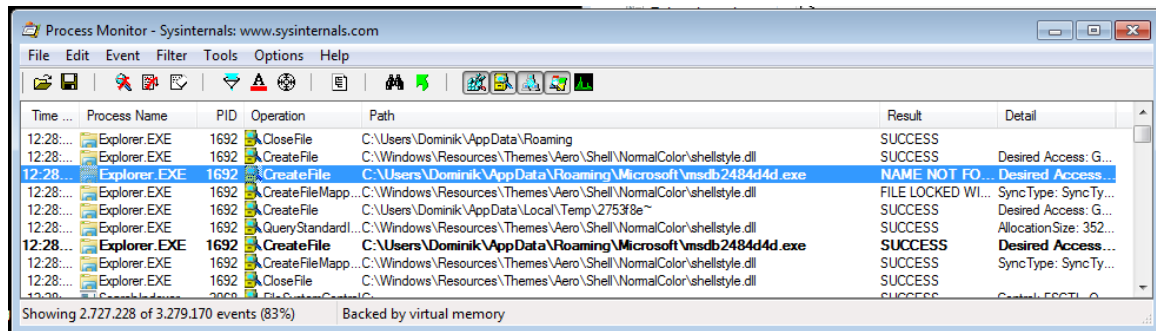
Values added:3
HKU\S-1-5-21-12[...]\Software\Microsoft\Windows\CurrentVersion\
    Explorer\UserAssist\{CE[...]EA}\Count\P:\Znyjner\81[...]86\s1
    [...]32.rkr:
        00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 80 BF
        00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF
        00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF FF FF FF FF
        00 88 32 0C E9 BE D0 01 00 00 00 00
HKU\S-1-5-21-12[...]\Software\Microsoft\Windows\CurrentVersion\
    Internet Settings\GlobalUserOffline: 0x00000000
HKU\S-1-5-21-12[...]\Software\Microsoft\Windows\CurrentVersion\Run\
    msdb2484d4d.exe: "C:\Users\Dominik\AppData\Roaming\Microsoft\
    msdb2484d4d.exe"

```

Besonders interessant ist dabei der letzte Eintrag, welcher verrät, dass unter *C:\Users\Dominik\AppData\Roaming\Microsoft\msdb2484d4d.exe* ein neues Programm angelegt wurde, welches beim nächsten Start ausgeführt wird. Prüft man den Hash der Datei, entspricht dieser der Datei *f113, f113cf383214f2788876d27d644ab432*. Daher wird die Datei im Folgenden *msdb* genannt. Diese wird unter dem oben genannten Pfad gehalten, um das Verhalten der Malware nicht negativ zu beeinflussen. Ebenfalls fallen die Einträge, welche „Rocal AppWizard-Generated Applications“ enthalten, auf. Dieser String wurde bereits in der statischen Analyse mit Hilfe von *PEView* unter Abschnitt 4.5 gefunden.

5.2.3. Process Monitor

Aus dem *Process Monitor*-Log konnten keine weiteren Kenntnisse gewonnen werden. Das Log ist (gefiltert nach der Datei *f113*) unter *f113cf383214f2788876d27d644ab432.CSV* an die Arbeit angehängt. Es konnte jedoch bestätigt werden, dass die Datei *msdb* geschrieben und zum Autostart hinzugefügt wurde (siehe Abbildung 10). Die Datei *msdb* scheint nicht direkt aufgerufen worden zu sein. Zumindest zeigt der Filter keinen Treffer für *msdb*.

Abbildung 10: Process Monitor nach Ausführung von *f113* zeigt die Erstellung von *msdb*

5.2.4. Wireshark

Es konnte kein von der Malware in diesem Stadium ausgelöster Netzwerkverkehr festgestellt werden.

5.2.5. Process Explorer

Im *Process Explorer* können Post-Mortem keine Anomalien festgestellt werden. Über das Aufrufen von *Optionen* → *Virus Total* → *Check Virustotal* können die Hashes aller aktuell laufenden Prozesse gegen die Onlinedatenbank geprüft werden. Dabei wurden alle Prozesse negativ getestet (siehe Abbildung 16). Daraus lässt sich vermuten, dass die Malware zumindest nach der ersten direkten Ausführung nicht mehr läuft. Nachdem die Datei *msdb* jedoch bei einem Neustart des Rechners ausgeführt wird, könnten sich nach einem Neustart neue Funktionen zeigen.

5.2.6. Autoruns

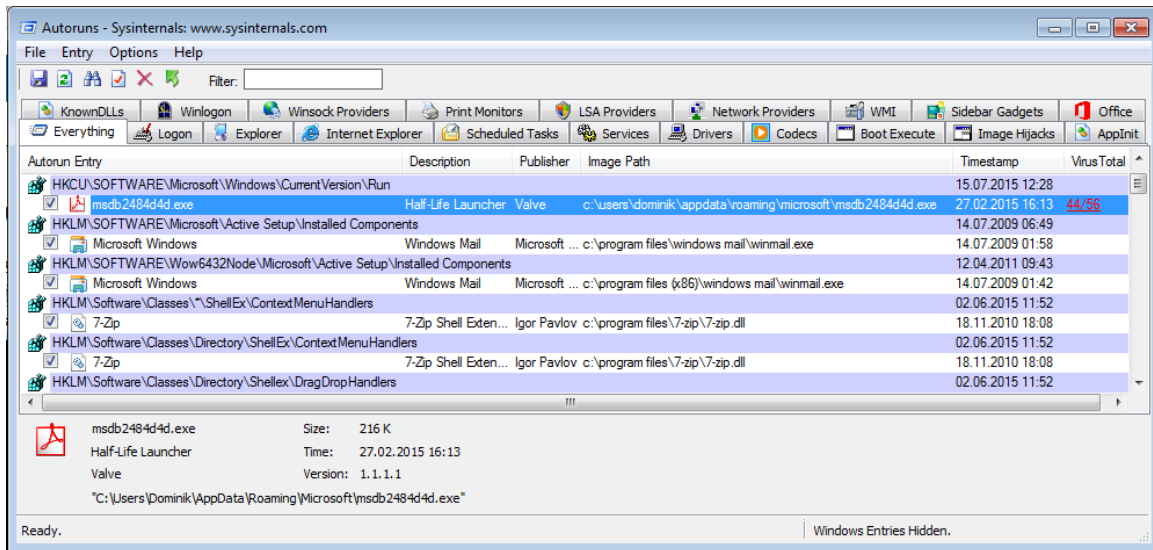
Über das Tool *Autoruns* kann die Erkenntnis aus 5.2.2 verifiziert werden. Wie unter der Abbildung 11 zu sehen ist, wurde ein Autostart-Eintrag für die Datei *msdb* erstellt.

5.3. Immunity Debugger

Nach der Analyse wird die Maschine auf den Zustand vor der Live-Analyse zurückgesetzt. Anschließend wird die Datei *f113* mit dem *Immunity Debugger* (siehe Beschreibung 2.5.3) untersucht. Dazu öffnet man das Tool und anschließend die Datei *f113*. Das Fenster ist im Anhang in Abbildung 17 dargestellt. Anschließend wird über die Tasten *F7* (*Step Into*) und *F8* (*Step Over*) die Ausführung gesteuert. Dabei sind vor allem System-Calls interessant, da diese meist ohne großen Aufwand auf Aktivitäten der Malware schließen lassen.

Schon zu Beginn lassen sich Parallelen zwischen der Disassemblierung von *IDA* und der Ausführung im Debugger herstellen. Dies ist im Anhang in Abbildung 18 verdeutlicht.

Später in der Laufzeit kommt man zu der in der statischen Analyse beschriebenen Entscheidung, ob die Unterfunktion *sub_402718* oder die Start-Funktion weiter ausgeführt werden

Abbildung 11: Autoruns nach Ausführung von *f113*

soll. Die Situation kurz vor dem Vergleich ist im Anhang in der Abbildung 19 dargestellt. Da *EBX* und *ESI* nicht gleich sind, wird die *Zero*-Flag nicht gesetzt und der Jump (*jnz*) wird normalerweise ausgeführt. Damit würde das Programm in Unterfunktion fortfahren. Das folgende Verhalten gleicht dem bisher beschriebene Verhalten. Um zu testen, was passiert, wenn der Vergleich anders ausgeht, wird der Sprung *JNZ* auf *JE* manipuliert (welcher das Gegenstück zu *JNZ* ist). Abbildung 12 zeigt den Assemblercode nach der Anpassung.

```

004027EC . 8830 MOV ESI,DWORD PTR DS:[EAX]
004027EE . 3BF3 CMP ESI,EBX
004027F0 74 13 JE SHORT f113cf38.00402805
004027F2 . 834D FC FF OR DWORD PTR SS:[EBP-4],FF

```

Abbildung 12: Veränderung *JNZ* zu *JZ* während der Ausführung von *f113*

Leider führt diese weitere Ausführung auf eine *Access-Violation*. Da Programme gezielt gegen Debugging geschützt werden können und eine solche Methode hier vermutlich zum Einsatz kommt, wurde die weitere Ausführung aufgegeben.

5.4. Weiter Analyse

Um das weitere Verhalten der Datei *msdb* zu analysieren, wurden *Process Monitor* und *Wireshark* herangezogen. Zur Analyse der Datei *msdb* bei einem Neustart wird im *Process Monitor* die Option \Rightarrow *Enable Boot Logging* aktiviert. Nach einem Neustart kann man nun die während der Bootzeit ausgeführten Aktionen analysieren.

Wireshark zeigt wiederum keinen auffälligen Netzwerkverkehr. Das Log des *Process Monitor* (welches nach dem Öffnen der Software nach dem Neustart angeboten wird) zeigt zwar, dass

die Datei *msdb* geladen wurde, zeigt aber keine weiteren Aktionen. Im *Prozess Explorer* kann die Datei *msdb* ebenfalls nicht als laufendes Programm gefunden werden.

5.5. Fazit zur dynamischen Analyse

Die dynamische Analyse konnte einige grundlegende Verhalten der Malware in Erfahrung bringen:

- Die Ursprungs-Datei *f113* löscht sich nach der Ausführung selbst.
- Die Datei *msdb* wird nach der Ausführung von Datei *f113* in den Ordner `C:\Users\Dominik\AppData\Roaming\Microsoft` kopiert.
- Für die Datei *msdb* wird ein Autostart-Eintrag unter Software `\Microsoft\Windows\CurrentVersion\Run` angelegt

Leider konnten, vermutlich aufgrund der Eigenschaft, dass die Malware erkennt, dass sie analysiert wird, nicht alle Funktionalitäten in Erfahrung gebracht werden.

6. Fazit

Durch die statische Analyse ließen sich verschiedene mögliche Funktionen der Malware erschließen (siehe Sektion 4.7). Daraus lassen sich für Nutzer Sicherheitshinweise ableiten. So sollten diese immer zuerst die Dateiendung überprüfen, und sich nicht von Icons täuschen lassen.

Ebenso konnte über die dynamische Analyse zumindest ein Teil der Funktionalität aufgedeckt werden (siehe Abschnitt 5.5). Aus den Erkenntnissen lassen sich *Indicator of Compromise* ableiten. So könnte man zum Beispiel bei dem Verdacht, dass ein Rechner mit der hier analysierten Malware infiziert ist, auf die Datei *msdb* prüfen und die Autorun-Einträge untersuchen.

Dass keine weiteren Aktivitäten der Malware verzeichnet werden konnten, kann sich auf mehrere Gründe zurückführen lassen. So könnte die Malware nur für eine spezielle Windows-Konfiguration entwickelt worden sein, welche hier nicht geboten wurde. Auch wäre es möglich, dass die Malware sich erst zu einem bestimmten Zeitpunkt aktiviert und bis dahin schläft. Am wahrscheinlichsten ist jedoch, dass die Malware die virtuelle Umgebung erkannt hat und daher nicht seine eigentliche Funktionalität entfaltet. Die Annahme, dass die Malware die VM erkannt hat, wird durch die Analyse von Alexey Shulmin auf [securelist.com](https://securelist.com/analysis/publications/69560/the-bank-ing-trojan-emetet-detailed-analysis/)¹⁷ gestützt.

¹⁷<https://securelist.com/analysis/publications/69560/the-bank-ing-trojan-emetet-detailed-analysis/>

A. Appendix

A.1. Version Info

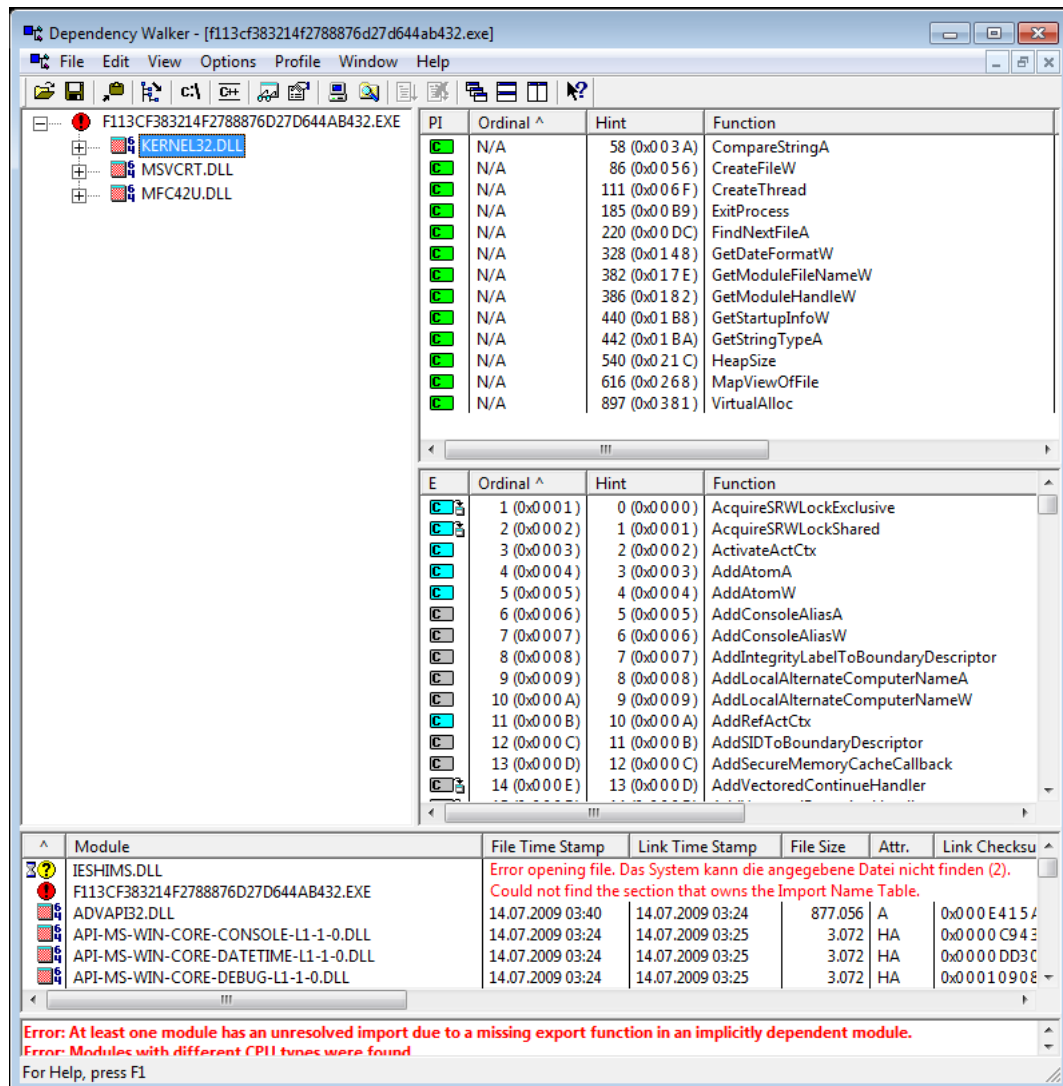
```
1 VERSIONINFO
FILEVERSION 1,1,1,1
PRODUCTVERSION 1,1,1,1
FILEOS 0x4
FILETYPE 0x1
{
BLOCK "StringFileInfo"
{
    BLOCK "040904b0"
    {
        VALUE "CompanyName", "Valve"
        VALUE "FileDescription", "Half-Life Launcher"
        VALUE "FileVersion", "1, 1, 1, 1"
        VALUE "InternalName", "Half-Life Launcher"
        VALUE "LegalCopyright", "Copyright (c) 1996-2003"
        VALUE "LegalTrademarks", ""
        VALUE "OriginalFilename", "hl.exe"
        VALUE "ProductName", "Half-Life Launcher"
        VALUE "ProductVersion", "1, 1, 1, 1"
    }
}

BLOCK "VarFileInfo"
{
    VALUE "Translation", 0x0409 0x04B0
}
}
```

A.2. Abbildungen



Abbildung 13: Gefälschte UPS-Mail

Abbildung 14: Aufruf der Datei *f113* mit dem Dependency Walker


















































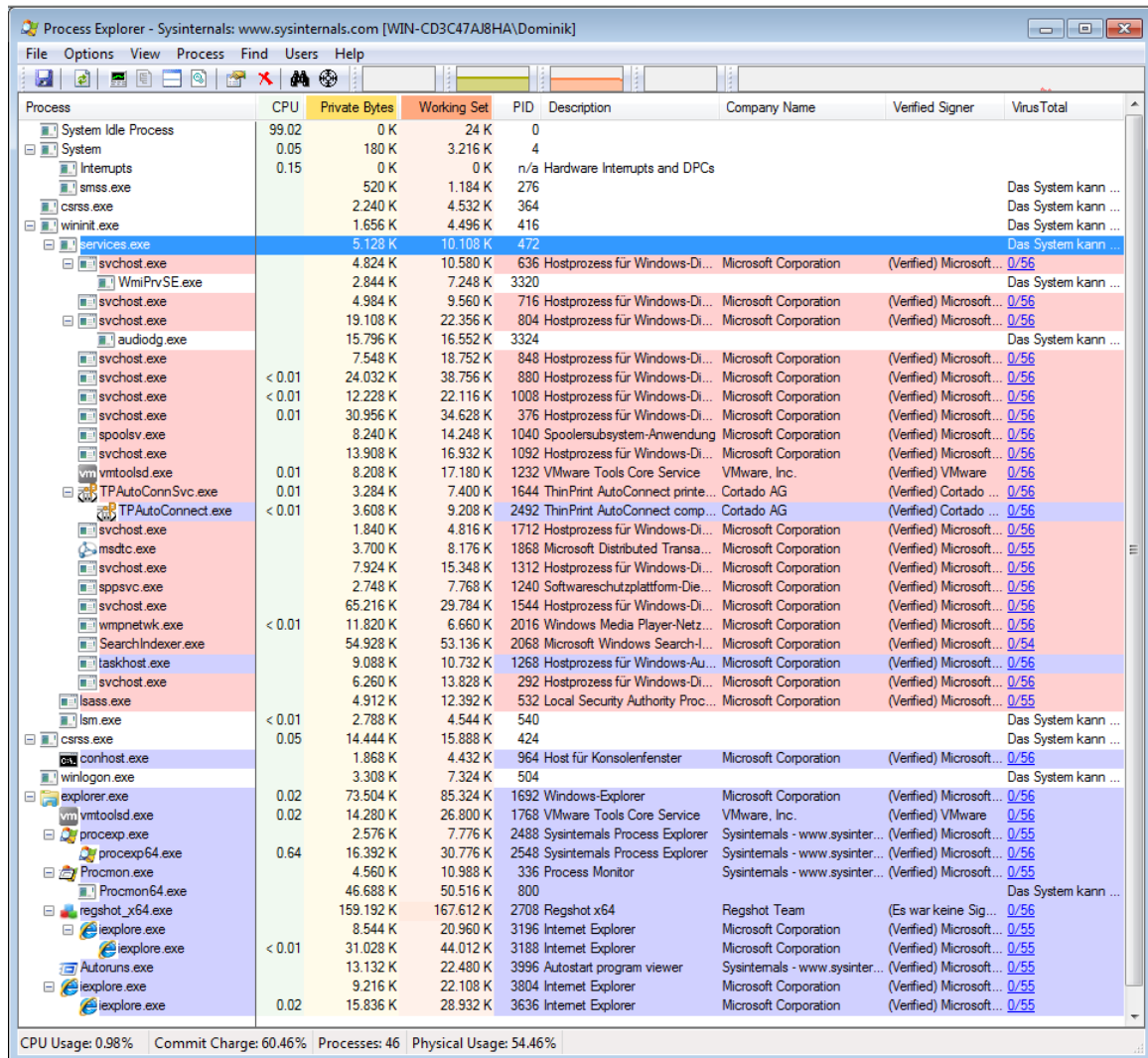
Function Name	Segment	Start	Length	Offset	Segment	R	X	C	D	W	...
 CFrameWnd::GetActiveDocument(void)	.text	0000000000402688	00000006			R	T
 CFrameWnd::GetActiveFrame(void)	.text	0000000000402682	00000006			R	T
 CFrameWnd::GetDefaultAccelerator(void)	.text	0000000000402646	00000006			R	T
 CFrameWnd::GetMessageBar(void)	.text	0000000000402664	00000006			R	T
 CFrameWnd::GetMessageString(uint, CString &)	.text	000000000040267C	00000006	00000000	00000000	R	T
 CFrameWnd::IsFrameWnd(void)	.text	0000000000402694	00000006			R	T
 CFrameWnd::LoadFrame(uint,ulong,CWnd *,CCreateCo...	.text	000000000040268E	00000006	00000000	00000000	R	T
 CFrameWnd::NegotiateBorderSpace(uint,tagRECT *)	.text	000000000040265E	00000006	00000000	00000000	R	T
 CFrameWnd::OnCmdMsg(uint,int,void *,AFX_CMDHAN...	.text	00000000004026B8	00000006	00000000	00000000	R	T
 CFrameWnd::OnCommand(uint,long)	.text	00000000004026B2	00000006	00000000	00000000	R	T
 CFrameWnd::OnCreate(tagCREATESTRUCTW *)	.text	0000000000402706	00000006	00000000	00000000	R	T
 CFrameWnd::OnCreateClient(tagCREATESTRUCTW *,CC...	.text	0000000000402658	00000006	00000000	00000000	R	T
 CFrameWnd::OnSetPreviewMode(int,CPrintPreviewState...	.text	000000000040266A	00000006	00000000	00000000	R	T
 CFrameWnd::OnUpdateFrameMenu(HMENU_*)	.text	000000000040264C	00000006	00000000	00000000	R	T
 CFrameWnd::OnUpdateFrameTitle(int)	.text	0000000000402652	00000006	00000000	00000000	R	T
 CFrameWnd::PostNcDestroy(void)	.text	000000000040269A	00000006			R	T
 CFrameWnd::PreCreateWindow(tagCREATESTRUCTW &)	.text	000000000040270C	00000006	00000000	00000000	R	T
 CFrameWnd::PreTranslateMessage(tagMSG *)	.text	00000000004026A0	00000006	00000000	00000000	R	T
 CFrameWnd::RecalcLayout(int)	.text	0000000000402676	00000006	00000000	00000000	R	T
 CFrameWnd::~CFrameWnd(void)	.text	00000000004026C4	00000006			R	T
 CGdiObject::Attach(void *)	.text	000000000040260A	00000006	00000000	00000000	R	T
 CGdiObject::DeleteObject(void)	.text	0000000000402604	00000006			R	T
 CGdiObject::GetRuntimeClass(void)	.text	00000000004025F8	00000006			R	T
 CObject::GetRuntimeClass(void)	.text	00000000004025FE	00000006			R	T
 CPen::GetRuntimeClass(void)	.text	00000000004025F2	00000006			R	T
 CSingleDocTemplate::CSingleDocTemplate(uint,CRuntim...	.text	000000000040239A	00000006	00000000	00000000	R	T
 CStatusBar::CStatusBar(void)	.text	00000000004026D0	00000006			R	T
 CStatusBar::Create(CWnd *,ulong,uint)	.text	00000000004026FA	00000006	00000000	00000000	R	T
 CStatusBar::SetIndicators(uint const *,int)	.text	00000000004026F4	00000006	00000000	00000000	R	T
 CStatusBar::~CStatusBar(void)	.text	000000000040268E	00000006			R	T
 CToolBar::CToolBar(void)	.text	00000000004026CA	00000006			R	T
 CToolBar::CreateEx(CWnd *,ulong,ulong,CRect,uint)	.text	0000000000402700	00000006	00000000	00000000	R	T
 CToolBar::LoadToolBar(ushort const *)	.text	0000000000402712	00000006	00000000	00000000	R	T
 CToolBar::~CToolBar(void)	.text	00000000004026DC	00000006			R	T
 CView::CView(void)	.text	00000000004025C2	00000006			R	T
 CView::CalcWindowRect(tagRECT *,uint)	.text	00000000004025B6	00000006	00000000	00000000	R	T
 CView::DoPreparePrinting(CPrintInfo *)	.text	00000000004025D4	00000006	00000000	00000000	R	T
 CView::GetScrollBarCtrl(int)	.text	00000000004025B0	00000006	00000000	00000000	R	T
 CView::IsSelected(CObject const *)	.text	0000000000402592	00000006	00000000	00000000	R	T
 CView::OnActivateFrame(uint,CFrameWnd *)	.text	000000000040254A	00000006	00000000	00000000	R	T
 CView::OnActivateView(int,CView *,CView *)	.text	0000000000402550	00000006	00000000	00000000	R	T
 CView::OnCmdMsg(uint,int,void *,AFX_CMDHANDLERI...	.text	00000000004025BC	00000006	00000000	00000000	R	T
 CView::OnCreate(tagCREATESTRUCTW *)	.text	00000000004025EC	00000006	00000000	00000000	R	T
 CView::OnDragEnter(COLEDataObject *,ulong,CPoint)	.text	0000000000402580	00000006	00000000	00000000	R	T
 CView::OnDragLeave(void)	.text	0000000000402574	00000006			R	T
 CView::OnDragOver(COLEDataObject *,ulong,CPoint)	.text	000000000040257A	00000006	00000000	00000000	R	T
 CView::OnDragScroll(ulong,CPoint)	.text	0000000000402562	00000006	00000000	00000000	R	T
 CView::OnDrop(COLEDataObject *,ulong,CPoint)	.text	000000000040256E	00000006	00000000	00000000	R	T
 CView::OnDropEx(COLEDataObject *,ulong,ulong,CPoint)	.text	0000000000402568	00000006	00000000	00000000	R	T

Abbildung 15: Funktionsaufrufe der Datei *f113*, Ansicht in IDA

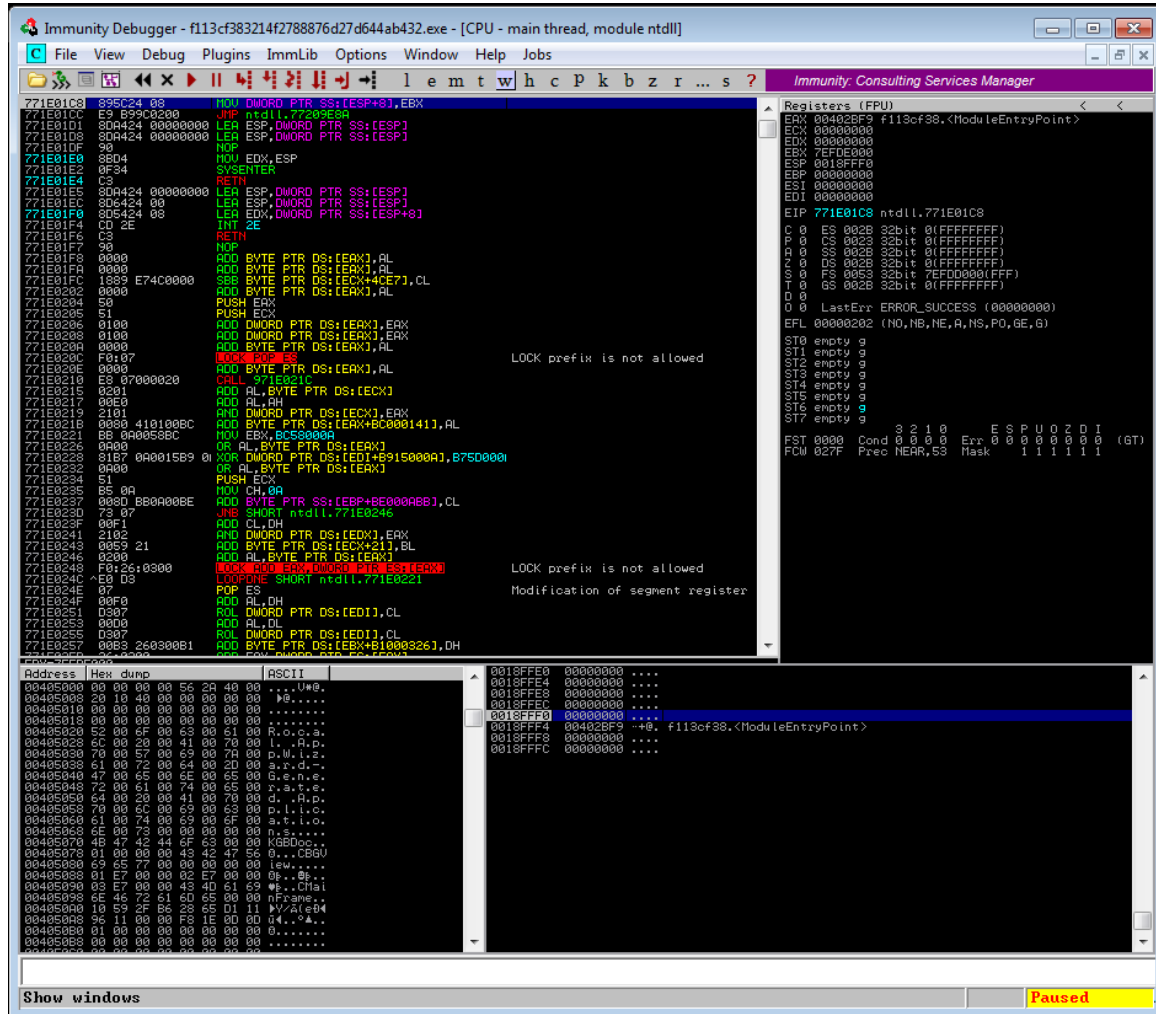


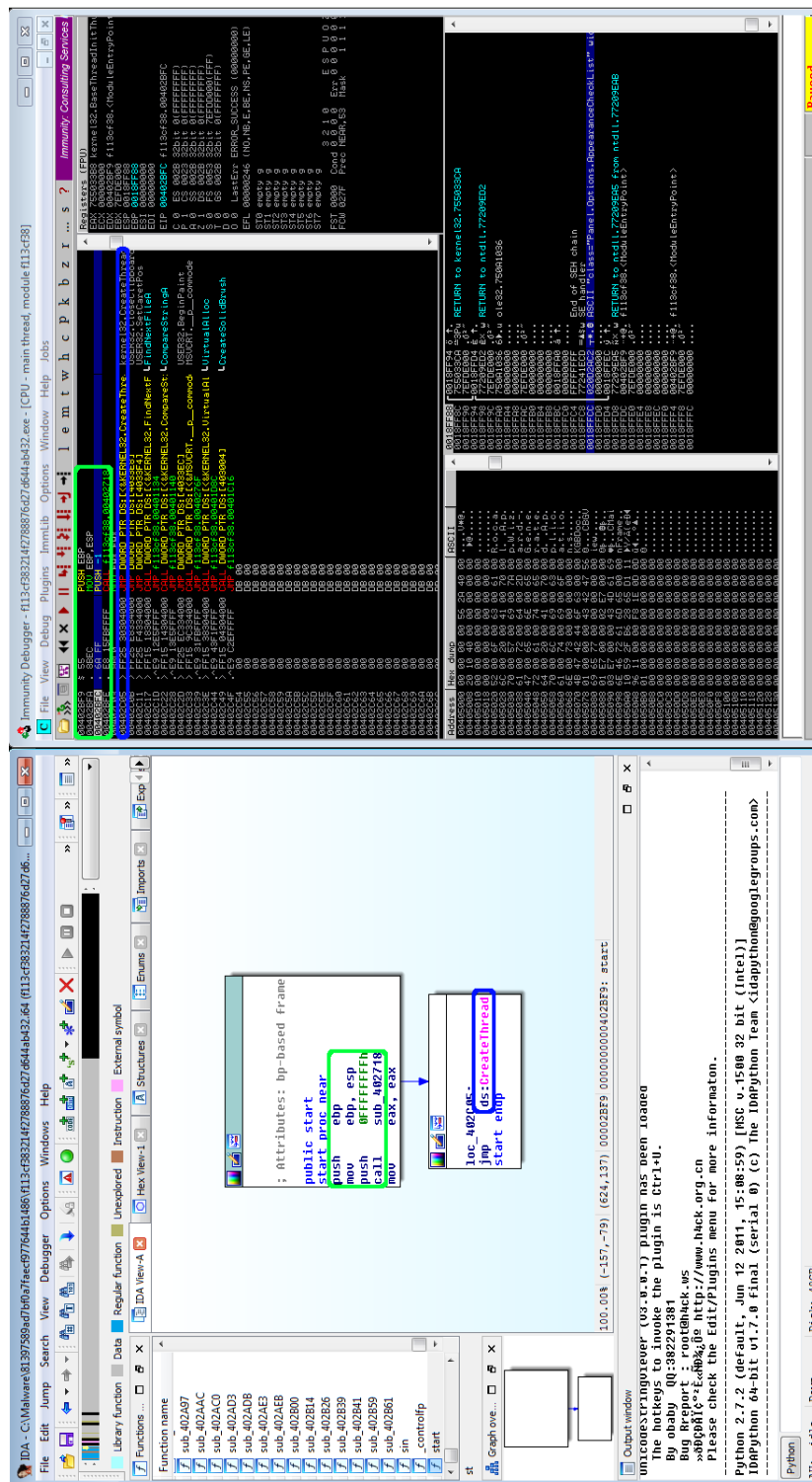
Process Explorer - Sysinternals: www.sysinternals.com [WIN-CD3C47AJ8HA\Dominik]

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Verified Signer	Virus Total
System Idle Process	99.02	0 K	24 K	0				
System	0.05	180 K	3.216 K	4				
Interrupts	0.15	0 K	0 K	n/a	Hardware Interrupts and DPCs			
smss.exe		520 K	1.184 K	276				Das System kann ...
csrss.exe		2.240 K	4.532 K	364				Das System kann ...
wininit.exe		1.656 K	4.496 K	416				Das System kann ...
services.exe		5.128 K	10.108 K	472				Das System kann ...
svchost.exe		4.824 K	10.580 K	636	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
WmiPrvSE.exe		2.844 K	7.248 K	3320				Das System kann ...
svchost.exe		4.984 K	9.560 K	716	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
svchost.exe		19.108 K	22.356 K	804	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
audiodg.exe		15.796 K	16.552 K	3324				Das System kann ...
svchost.exe		7.548 K	18.752 K	848	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
svchost.exe	< 0.01	24.032 K	38.756 K	880	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
svchost.exe	< 0.01	12.228 K	22.116 K	1008	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
svchost.exe	0.01	30.956 K	34.628 K	376	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
spoolsv.exe		8.240 K	14.248 K	1040	Spoolersubsystem-Anwendung	Microsoft Corporation	(Verified) Microsoft...	0/56
svchost.exe		13.908 K	16.932 K	1092	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
vmtoolsd.exe	0.01	8.208 K	17.180 K	1232	VMware Tools Core Service	VMware, Inc.	(Verified) VMware	0/56
TPAutoConnSvc.exe	0.01	3.284 K	7.400 K	1644	ThinPrint AutoConnect printe...	Cortado AG	(Verified) Cortado	0/56
TPAutoConnect.exe	< 0.01	3.608 K	9.208 K	2492	ThinPrint AutoConnect comp...	Cortado AG	(Verified) Cortado	0/56
svchost.exe		1.840 K	4.816 K	1712	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
msdtc.exe		3.700 K	8.176 K	1868	Microsoft Distributed Transa...	Microsoft Corporation	(Verified) Microsoft...	0/55
svchost.exe		7.924 K	15.348 K	1312	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
spsvc.exe		2.748 K	7.768 K	1240	Softwareschutzplattform-Die...	Microsoft Corporation	(Verified) Microsoft...	0/56
svchost.exe		65.216 K	29.784 K	1544	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
wmpnetwk.exe	< 0.01	11.820 K	6.660 K	2016	Windows Media Player-Netz...	Microsoft Corporation	(Verified) Microsoft...	0/56
SearchIndexer.exe		54.928 K	53.136 K	2068	Microsoft Windows Search-I...	Microsoft Corporation	(Verified) Microsoft...	0/54
taskhost.exe		9.088 K	10.732 K	1268	Hostprozess für Windows-Au...	Microsoft Corporation	(Verified) Microsoft...	0/56
svchost.exe		6.260 K	13.828 K	292	Hostprozess für Windows-Di...	Microsoft Corporation	(Verified) Microsoft...	0/56
lsass.exe		4.912 K	12.392 K	532	Local Security Authority Proc...	Microsoft Corporation	(Verified) Microsoft...	0/55
lsn.exe	< 0.01	2.788 K	4.544 K	540				Das System kann ...
csrss.exe	0.05	14.444 K	15.888 K	424				Das System kann ...
conhost.exe		1.868 K	4.432 K	964	Host für Konsolenfenster	Microsoft Corporation	(Verified) Microsoft...	0/56
winlogon.exe		3.308 K	7.324 K	504				Das System kann ...
explorer.exe	0.02	73.504 K	85.324 K	1692	Windows-Explorer	Microsoft Corporation	(Verified) Microsoft...	0/56
vmtoolsd.exe	0.02	14.280 K	26.800 K	1768	VMware Tools Core Service	VMware, Inc.	(Verified) VMware	0/56
procexp64.exe		2.576 K	7.776 K	2488	Sysinternals Process Explorer	Sysinternals - www.sysinter...	(Verified) Microsoft...	0/55
Procmon.exe	0.64	16.392 K	30.776 K	2548	Sysinternals Process Explorer	Sysinternals - www.sysinter...	(Verified) Microsoft...	0/56
Procmon64.exe		4.560 K	10.988 K	336	Process Monitor	Sysinternals - www.sysinter...	(Verified) Microsoft...	0/55
regshot_x64.exe		46.688 K	50.516 K	800				Das System kann ...
regshot_x64.exe		159.192 K	167.612 K	2708	Regshot x64	Regshot Team	(Es war keine Sig...	0/56
ieexplore.exe		8.544 K	20.960 K	3196	Internet Explorer	Microsoft Corporation	(Verified) Microsoft...	0/55
ieexplore.exe	< 0.01	31.028 K	44.012 K	3188	Internet Explorer	Microsoft Corporation	(Verified) Microsoft...	0/55
Autoruns.exe		13.132 K	22.480 K	3996	Autostart program viewer	Sysinternals - www.sysinter...	(Verified) Microsoft...	0/55
ieexplore.exe		9.216 K	22.108 K	3804	Internet Explorer	Microsoft Corporation	(Verified) Microsoft...	0/55
ieexplore.exe	0.02	15.836 K	28.932 K	3636	Internet Explorer	Microsoft Corporation	(Verified) Microsoft...	0/55

CPU Usage: 0.98% Commit Charge: 60.46% Processes: 46 Physical Usage: 54.46%

Abbildung 16: Process Explorer nach Ausführung von *f113* mit Abgleich der Prozesse mit Virustotal

Abbildung 17: Hauptfenster des *Immunity Debugger* nach Ausführung von *f113*

Abbildung 18: Zusammenhang von *IDA* und *Immunity Debuggers* bei *f113*

