

KRESTEN JACOBSEN

PACKET INJECTION M/ SCAPY

Hvad: Python modul til at manipulere og arbejde med pakker.

Send, modtag og indfang pakker.

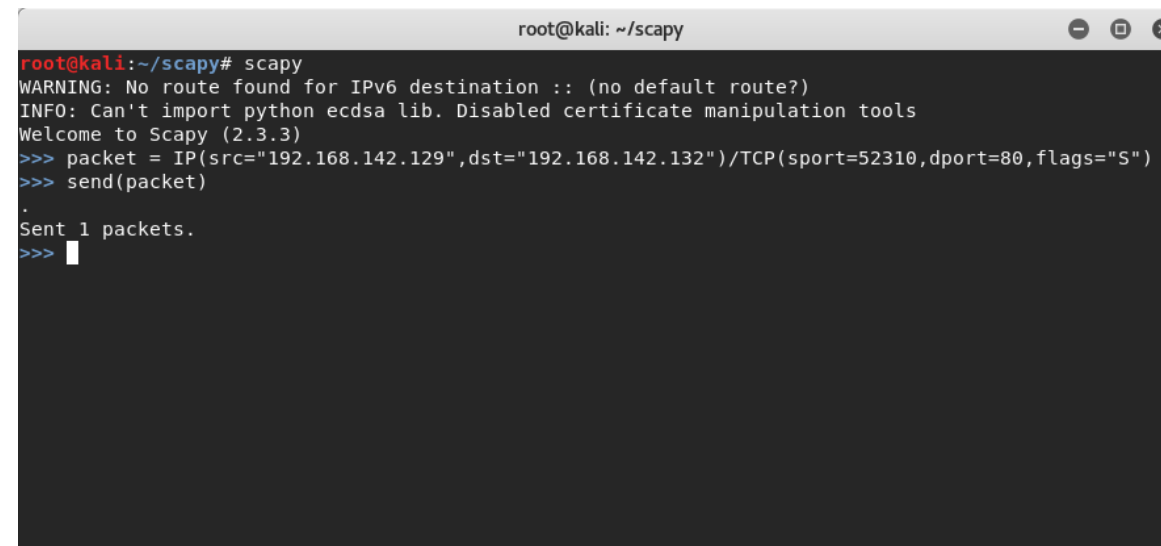
Hvorfor: Primært test formål (men kan principielt også bruges offensivt).

Eks. 1: Test af netværk over en bred kam (å la wireshark).

Eks. 2: Test af forsvarsmekanismer (eks. IDS / IPS regler).

EKSEMPEL 1 – PACKET INJECTION MED SCAPY

- ▶ Enkelt syn-pakke bygget og sendt:



```
root@kali: ~/scapy
root@kali:~/scapy# scapy
WARNING: No route found for IPv6 destination :: (no default route?)
INFO: Can't import python ecdsa lib. Disabled certificate manipulation tools
Welcome to Scapy (2.3.3)
>>> packet = IP(src="192.168.142.129",dst="192.168.142.132")/TCP(sport=52310,dport=80,flags="S")
>>> send(packet)
.
Sent 1 packets.
>>> 
```

1) Start scapy

2) Byg pakke:

IP-niveau: afsender-ip, destination-ip / TCP-niveau: afsender port, destination-port, syn-flag.

3) Send pakke

EKSEMPEL 1 (FORTSAT) – PACKET INJECTION MED SCAPY

Wireshark capture showing a packet injection example. The packet list shows three packets: a SYN packet (No. 11), a SYN/ACK packet (No. 12), and a RST packet (No. 15). The packet details for packet 11 are expanded, showing the TCP header with flags set to SYN. The packet bytes show the raw data of the SYN packet.

No.	Time	Source	Destination	Protocol	Length	Info
11	1.669293623	192.168.142.129	192.168.142.132	TCP	54	52310 → 80 [SYN] Seq=0 Win=8192 Len=0
12	1.669817503	192.168.142.132	192.168.142.129	TCP	60	80 → 52310 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460
15	1.670027211	192.168.142.129	192.168.142.132	TCP	54	52310 → 80 [RST] Seq=1 Win=0 Len=0

Frame 11: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0

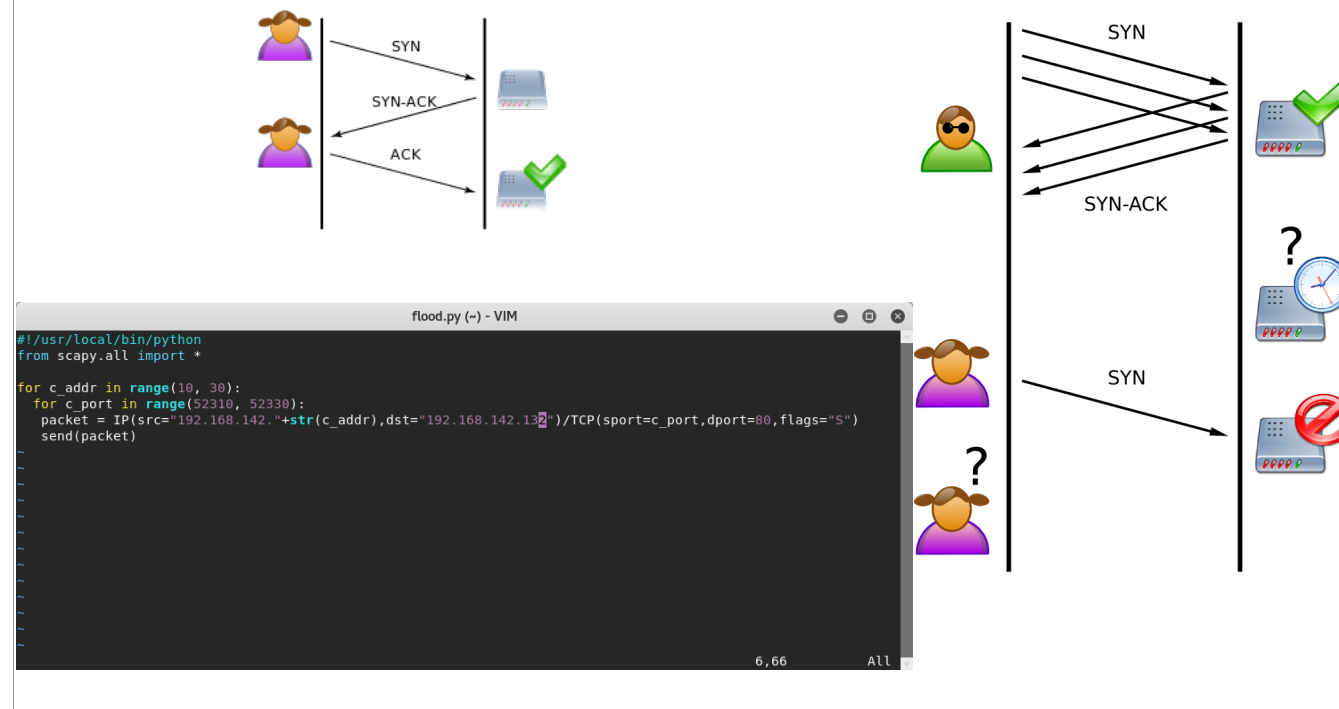
- Ethernet II, Src: Vmware_ba:b8:5a (00:0c:29:ba:b8:5a), Dst: Vmware_63:7f:77 (00:0c:29:63:7f:77)
- Internet Protocol Version 4, Src: 192.168.142.129, Dst: 192.168.142.132
- Transmission Control Protocol, Src Port: 52310, Dst Port: 80, Seq: 0, Len: 0
 - Source Port: 52310
 - Destination Port: 80
 - [Stream index: 0]
 - [TCP Segment Len: 0]
 - Sequence number: 0 (relative sequence number)
 - Acknowledgment number: 0
 - Window size value: 8192
 - [Calculated window size: 8192]
 - Checksum: 0x24e5 [unverified]
 - [Checksum Status: Unverified]
 - Urgent pointer: 0

0000 00 0c 29 63 7f 77 00 0c 29 ba b8 5a 08 00 45 00 ..)c.w..)..Z..E.
0010 00 28 00 01 00 00 40 06 dc 78 c0 a8 8e 81 c0 a8 .(....@. .x.....
0020 8e 84 cc 56 00 50 00 00 00 00 00 00 00 50 02 ...V.P..P.
0030 20 00 24 e5 00 00 ..\$....

Wireshark-capture af pakken fra foregående slide. Normal trafik burde skabe et “syn, syn/ack, ack”-flow.

Grunden til at, der også bliver sendt en "reset" pakke til sidst er at ip'en _ikke_ er spoofet i dette tilfælde og at den oprindelige afsender-maskine ikke forstår, hvorfor den pludselig får en syn/ack-pakke; der er jo ingen applikation, der står og venter på et syn/ack-svar. Dette kan undgås ved at sætte en regel i den lokale firewall, som blokerer de udgående RST pakker.

EKSEMPEL 2 SYN-FLOOD MED SCAPY



Øverst til venstre: Normalt syn, syn-ack, ack flow.

Nederst til venstre: simpelt script til syn-flooding.

Højre: Syn-flood flow (blokering af Alice' legitime syn-request).

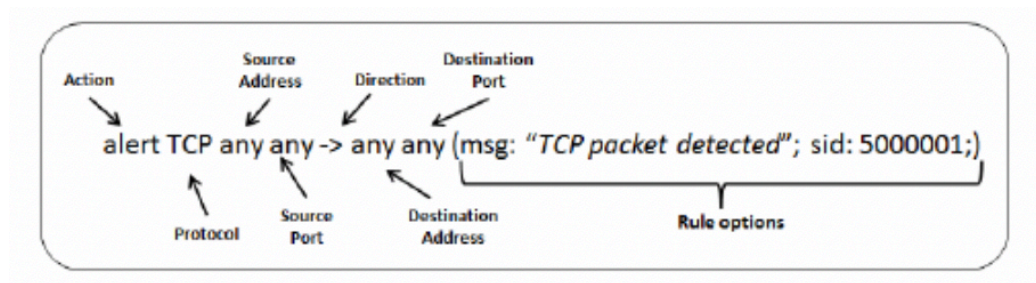
Forbedringsforslag: randomisering af afsendende MAC-adresse indsæt (Ether(src="ab:cd:ef:ab:cd:ef", dst="ff:ff:ff:ff:ff:ff"))

RELATERET EMNE: IDS / IPS

- IDS / IPS'er kan bruges til at opdage og forhindre eks. syn-flooding.

- Eks. på snort-regel :

```
alert tcp any any -> 192.168.65.132 any (msg:"TCP SYN flood attack  
detected"; flags:S; threshold: type threshold, track by_dst, count 20,  
seconds 60; classtype=denial-of-service; priority:5; sid:5000001; rev:1;)
```



Snort-reglen sættes i `/etc/nsm/rules/local.rules`

Classtype overstreget, da jeg simpelthen ikke kunne få det til at virke med den sat og den derfor er pillet ud i reglen på næste side...

RELATERET EMNE: IDS / IPS

The screenshot shows the SGUIL-0.9.0 interface. At the top, there's a status bar with 'SGUIL-0.9.0 - Connected To localhost', 'ServerName: localhost', 'UserName: kresten', 'UserID: 2', and '2017-12-14 12:55:00 GMT'. Below this is a tabbed interface with 'RealTime Events' and 'Escalated Events'. The 'RealTime Events' tab is active, showing a table of events.

T	CNT	Sensor	Aler...	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event...
ST	3	securi...	3.1	2017-12-14 11:33:18	192.168.142.129	68	192.168.142.254	67	17	ET P...
ST	1	securi...	3.4	2017-12-14 12:48:05	192.168.232.10	52329	192.168.232.138	80	6	TCP S...
ST	1	securi...	3.5	2017-12-14 12:48:07	192.168.232.11	52329	192.168.232.138	80	6	TCP S...
ST	1	securi...	3.6	2017-12-14 12:48:08	192.168.232.134	68	192.168.232.254	67	17	ET P...
ST	1	securi...	3.7	2017-12-14 12:48:08	192.168.232.12	52329	192.168.232.138	80	6	TCP S...

Below the table, there's a 'Show Packet Data' and 'Show Rule' section. A red circle highlights the rule configuration text:

alert tcp any any -> 192.168.232.138 any (msg:"TCP SYN flood attack detected"; flags:S; threshold: type threshold, track by_dst, count 20, seconds 60; priority:5; sid:5000001; rev:2;)

The rule is located in /var/lib/sguidd/server_data/securityonion/rules/security-onion-eth0-1/local.rules: Line 2.

Below the rule, there's a packet details section showing IP, TCP, and DATA layers. The IP layer shows Source IP: 192.168.232.10 and Destination IP: 192.168.232.138. The TCP layer shows Source Port: 52329 and Destination Port: 80. The DATA layer is empty.

Screenshot fra SGUIL af capture fra foregående snort-regel.