why packet filtering

some reasons to *filter* packets:

remove (malicious?) packets with false source addresses

disallow external access to servers not on allowlist

block traffic from known malicious sources

only permit services on allowlist out of private network

packet filters = firewalls

typical name for packet filter: “firewall”

especially filtering focused on security

firewall design decisions

where to filter?

- typically: at "edges" of network
- in router, or separate box
- can also do elsewhere

how much to track when filtering?

- "stateless" or "stateful"

firewall design decisions

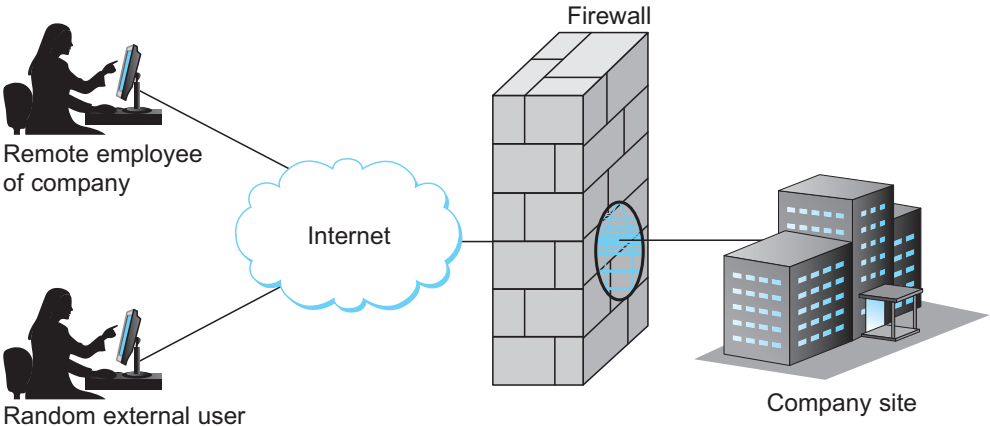
where to filter?

- typically: at "edges" of network
- in router, or separate box
- can also do elsewhere

how much to track when filtering?

- "stateless" or "stateful"

the corporate firewall



zones of trust

most common kind of firewall policy, three zones:

internal network

- can send most things
- can't receive unsolicited traffic from external network
- where typical laptops/desktops live

DMZ ('demilitrized zone')

- not 'protected' from external networks
- access to internal network similar to being within it
- where externally accessible services live

external network

- can only unsolicited to DMZ
- where the rest of the Internet lives

firewall rules

simple idea: look at packet's fields

table indicating whether to reject

similar idea to routing, but...

table matches fields other than destination

table acitons are 'drop', 'reject', etc.

“stateless” firewall

this design: table is set once by sysadmin

doesn’t change based on connections, etc.

might change based on manual reconfiguration, etc.

called “stateless”

firewall doesn’t track ‘state’ based on current activity

exercise: what info do we need

what fields do we need to match these

(and/or can we not handle without something more stateful?)

DNS queries to recursive DNS server from external servers

packets with source address that does not match network

packets from known malicious networks

connecting from outside to TCP servers not on allowlist

connecting from inside using services (HTTP, etc.) not on allowlist

Linux firewall: nftables

Linux's nftables — one Linux kernel packet filter interface

- command-line tools + system call API implemented in kernel
- will show syntax for `nft` tool
- also some other kernel APIs: eBPF 'programs', iptables

different "hooks" where filtering can happen, list for IP:

- prerouting
- input (if received locally), forward (otherwise)
- output (if sent from local program)
- postrouting

each hook has multiple "chains" of "rules"

```
# for use with 'nft' on Linux; probably not the best way to write these rules:
table inet filter {
    chain EXTERNAL-INPUT {
        ip6 saddr {3fff:14::/32, 3fff:30::/32} drop
        ip saddr 198.51.100.0/24 drop
        ip6 daddr 3fff:14:1:15 tcp dport {80, 443} accept
        ip6 daddr 3fff:14:1::/48 tcp dport 22 accept
        drop
    }
    chain INTERNAL-INPUT {
        ip6 saddr != {3fff:14::/32, 3fff:30::/32} drop
        ip saddr != 198.51.100.0/24 drop
        udp dport 137 drop
        accept
    }
    chain INPUT {
        type filter hook input priority 0
        iifname lo accept
        iifname ethExt jump EXTERNAL-INPUT
        iifname ethInt jump INTERNAL-INPUT
        reject
    }
}
```

limits of packet matching?

so far: showing stateless filtering

can stateless filtering to do a lot...:

match more header fields

check if packet contents match pattern

but some fundamental limits of design

filtering connections? (1)

let's say we're writing rule on router between 3fff::/16 network and internet

want to rule allow...

TCP connections from 3fff::2 to outside machines

and drop all other traffic

can't really do this with stateless rules

but can come close

exercise (1)

goal: allow 3fff::2 outside TCP connections only

how is this going to break?

assume we run this only packets about to be forwarded in router

```
ip6 saddr 3fff::2 protocol tcp accept
drop
```

exercise

goal: allow 3fff::2 outside TCP connections only

assume `ethInt` = inside; `ethExt` = outside

is this good enough?

assume we run this only packets about to be forwarded in router

```
iifname ethInt protocol tcp accept
iifname ethExt protocol \
    tcp flags & (fin|syn|rst|ack) == syn drop
iifname ethExt ip6 daddr 3fff::2 protocol tcp accept
iifname ethExt ip6 daddr 3fff::2 \
    icmpv6 { destination-unreachavble, time-exceeded } accept
drop
```

layering violation

previously router only handled IP (network layer)

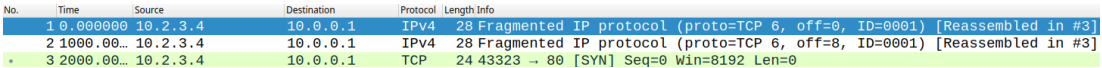
but looking at UDP/TCP fields

router typically doesn't do defragmentation

might interpret TCP/UDP fields different than end-host

fragmented SYN

```
tcp flags & (fin|syn|rst|ack) == syn drop
```



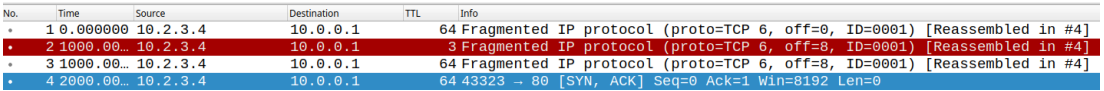
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.2.3.4	10.0.0.1	IPv4	28	Fragmented IP protocol (proto=TCP 6, off=0, ID=0001) [Reassembled in #3]
2	1000.00…	10.2.3.4	10.0.0.1	IPv4	28	Fragmented IP protocol (proto=TCP 6, off=8, ID=0001) [Reassembled in #3]
3	2000.00…	10.2.3.4	10.0.0.1	TCP	24	43323 → 80 [SYN] Seq=0 Win=8192 Len=0

tcp flags are not first fragment

need to reassemble to accurately filter

…or maybe filter out very short fragments

ambiguous fragmentation



No.	Time	Source	Destination	TTL	Info
1	0.000000	10.2.3.4	10.0.0.1	64	Fragmented IP protocol (proto=TCP 6, off=0, ID=0001) [Reassembled in #4]
2	1000.00…	10.2.3.4	10.0.0.1	3	Fragmented IP protocol (proto=TCP 6, off=8, ID=0001) [Reassembled in #4]
3	1000.00…	10.2.3.4	10.0.0.1	64	Fragmented IP protocol (proto=TCP 6, off=8, ID=0001) [Reassembled in #4]
4	2000.00…	10.2.3.4	10.0.0.1	64	43323 → 80 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0

can have two different versions of a fragment

using different TTLs to choose which one makes it

one idea: defragment to choose which version

and drop unknown fragments

other interpretation issues

wrote:

```
tcp flags & (fin|syn|rst|ack) == syn drop
```

versus: `tcp flags == syn drop`

"mishandles":

- SYN | ECE ?
- SYN | URG
- SYN | first reserved flag bit?
- ...

some of these combinations not defined by TCP standard

- don't really know if they open connection
- don't really know if they might be used for other purpose

probably related to why ECN often filtered

higher-level filtering?

let's say we want to disallow

`GET /malicious HTTP/1.1`, etc.

one idea: check for `GET /malicious`

huge number of issues:

- what if split across multiple TCP packets?
- what if uploading file containing `GET /malicious`?
- what about `GET  /malicious`, `GET /%6dalicious`?
- ...

web application firewalls

reverse HTTP proxy for firewalling

similar rules, but on HTTP requests/responses, not packets

example: Apache mod_security

follows: some rules from OWASP coreruleset project

example mod_security rule (wrapped)

```
SecRule RESPONSE_BODY "@rx <title>r57 Shell Version [0-9.]+
                        </title>|<title>r57 shell</title>" \
    "id:955110,\
    phase:4,\
    block,\
    capture,\
    t:none,\
    msg:'r57 web shell',\
    logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}',\
    ...
    severity:'CRITICAL',\
    setvar:'tx.outbound_anomaly_score_pl1=+%{tx.critical_anomaly_score}'"
```

example mod_security rule (wrapped)

```
SecRule REQUEST_URI|ARGS|REQUEST_HEADERS|!REQUEST_HEADERS:Referer|FILES|XML:\
      /* "@rx (?:(?:^|[\x5c/;])\.{2,3}[\x5c/;]|[\x5c/;]\.{2,3}(?:[\x5c/;]|$))" \
    "id:930110,\
    phase:2,\
    block,\
    capture,\
    t:none,t:utf8toUnicode,t:urlDecodeUni,t:removeNulls,t:cmdLine,\
    msg:'Path Traversal Attack (/../) or (/.../)',\
    logdata:'Matched Data: %{TX.0} found within %{MATCHED_VAR_NAME}: %{MATCHED_VAR
    ....
    ver:'OWASP_CRS/4.8.0',\
    severity:'CRITICAL',\
    multiMatch,\
    setvar:'tx.inbound_anomaly_score_pl1=+%{tx.critical_anomaly_score}',\
    setvar:'tx.lfi_score=+%{tx.critical_anomaly_score}'"
```

request smuggling

```
POST /foo HTTP/1.1
Content-Length: 5
Content-Length: 41

XXXXXBADMETHOD HTTP/1.1
Header-for-bad: GET /malicious HTTP/1.1
...
```

general ambiguity problem

when protocol ambiguous, filtering rules ineffective

usually lots of 'corner cases' where this can happen

- multiple content-length heaers
- unused flag bits being set to 1
- multiple versions of TCP segment or fragment
- ...

common defense: "normalization"

- remove things you don't understand
- make everything fit simple profile
- problem: breaking any fancy HTTP/TCP/etc. features

stateful firewall

common policy: allow outgoing connections only

prior approach:

- drop incoming non-TCP, or TCP SYN

problems:

- disallowing UDP-based protocols (example: DNS over UDP, HTTP/3)
- disallowing normal ICMP (example: ICMP ping replies)
- allowing unsolicited TCP packets

keeping state

outgoing connections only?

really want to track list of *connections*

most common form of *stateful firewall*

connection tracking?

simple idea for TCP:

see SYN: add (TCP, source IP+port, dest IP+port) to table

see FIN, FIN+ACK, ACK: remove row from table after timeout

(plus other timeouts, error cases, etc.)

mirrors TCP state machine

Linux conntrack

in-kernel table of active “connections”

includes notion of connections for UDP, ICMP, etc.

- heuristic guesses since protocol has no connect/close operation

maintains table of (proto, source host+port, dest host+port)

packets marked with connection state

- new, established, related, invalid

Linux conntrack

in-kernel table of active “connections”

includes notion of connections for UDP, ICMP, etc.

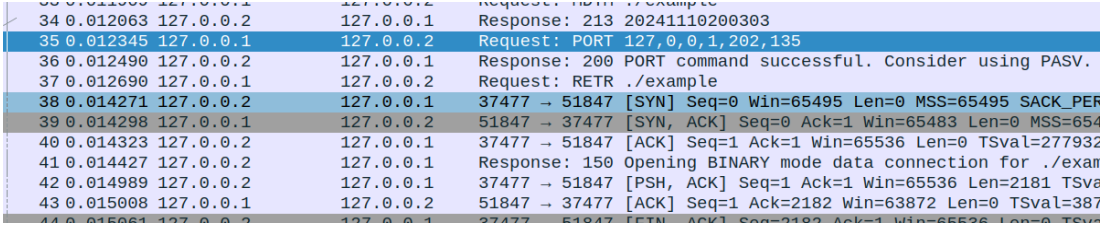
- heuristic guesses since protocol has no connect/close operation

maintains table of (proto, source host+port, dest host+port)

packets marked with connection state

- new, established, related, invalid

related connection example



34	0.012063	127.0.0.2	127.0.0.1	Response: 213 20241110200303
35	0.012345	127.0.0.1	127.0.0.2	Request: PORT 127,0,0,1,202,135
36	0.012490	127.0.0.2	127.0.0.1	Response: 200 PORT command successful. Consider using PASV.
37	0.012690	127.0.0.1	127.0.0.2	Request: RETR ./example
38	0.014271	127.0.0.2	127.0.0.1	37477 → 51847 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PER
39	0.014298	127.0.0.1	127.0.0.2	51847 → 37477 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=654
40	0.014323	127.0.0.2	127.0.0.1	37477 → 51847 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=277932
41	0.014427	127.0.0.2	127.0.0.1	Response: 150 Opening BINARY mode data connection for ./exam
42	0.014989	127.0.0.2	127.0.0.1	37477 → 51847 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=2181 TSva
43	0.015008	127.0.0.1	127.0.0.2	51847 → 37477 [ACK] Seq=1 Ack=2182 Win=63872 Len=0 TSval=387

FTP — uses separate control + data TCP connections

PORT command: server creates data connection to specified address

$202 \times 256 + 135 = 51847$

problem for firewalls: looks like ‘fresh’ TCP connection

FTP: server connect to client?

PORT command: server creates data connection to specified address

PASV command: server gets address for client to conect

- most FTP clients default to this mode

why both: in theory, allows direct server-to-server transfers

- one client uses PASV on one server, PORT on the other

other related connections Linux supports

usually: separate control and data connections

esp. when data sent with UDP or from different machine

direct-client-to-client file transfers in Internet Relay Chat

SIP, H.323 (video/audio call/conferencing protocols)

PPTP (VPN protocol)

nft with conntrack

```
table inet filter {
    chain input {
        type filter hook input priority 0

        ct state established accept
        ct state related accept
        ct state invalid drop
        iifname ethInt ct state new accept
        drop
    }
}
```

connection state timeouts

problem: TCP connection with no activity for 30 minutes

should it stay in table?

how about after 8 hours?

Linux conntrack, configurable timeouts:

- default: 5 days if in ESTABLISHED TCP state machine state
  lower for other connection TCP states (e.g. middle of handshake)

could disagree with end-host timeouts!

- mysterious connection dropping

TCP keep-alive

for TCP: SO_KEEPALIVE socket option

enables periodic “keep-alive” messages

- periodically send empty probe packets, resend last byte of data
- threshold for number of probes after which to consider connection lost

also many protocols have periodic ‘ping/pong’ messages

state size

problem with stateful firewalls:

how big can state table get?

what to do if state table is too big?

no great answers

SYN floods

easy to create *tons* of connections

attacker → server some port: TCP flags=SYN, ...
attacker → server some port: TCP flags=SYN, ...
...

can forge source IP address to make this harder to evade

problem for stateful firewalls and servers themselves

aside: SYN cookies

common mitigation for *end-hosts*: SYN cookies

don't store info about connections in SYN state

encode info MAC of conn.info in SYN+ACK's sequence number

MAC = message authentication code $\approx$ keyed hash

check MAC when initial ACK received

Linux synproxy

Linux tool for stateful firewall SYN floods:

'synproxy': firewall uses SYN cookies itself

replies instead of sending ACK directly to machine

requires knowledge of what SYN+ACK should look like

means firewall will break more TCP features?

denial-of-service

denial of service attack:

- overload network with too much traffic

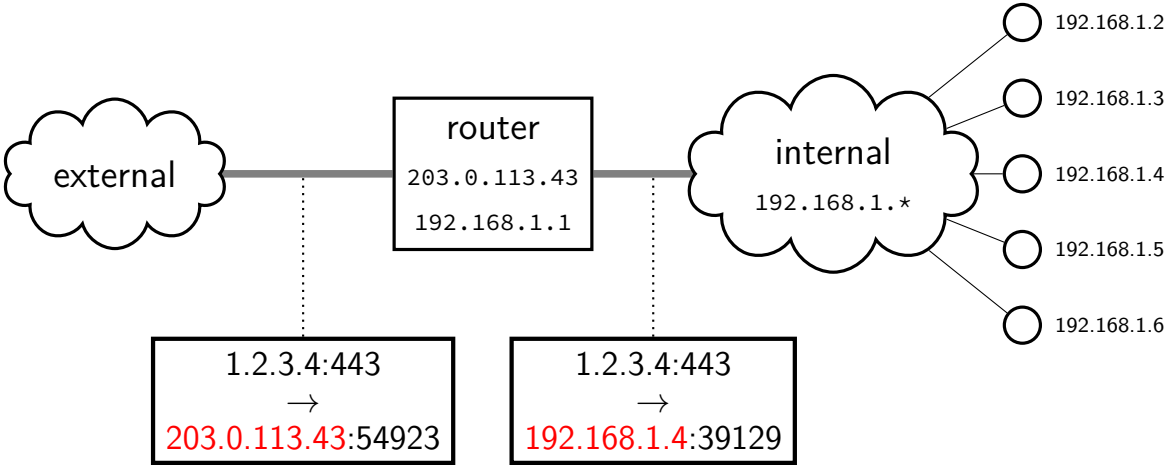
more effective when receiver does more work than sender

- reason SYN floods a common technique

also subject to “amplification”

- forging source address to make short (e.g. DNS) request
- ...that gets long response send to victim

NAT idea


external
router
203.0.113.43
192.168.1.1
internal
192.168.1.*
192.168.1.2
192.168.1.3
192.168.1.4
192.168.1.5
192.168.1.6
1.2.3.4:443
→
203.0.113.43:54923
1.2.3.4:443
→
192.168.1.4:39129

network address translation (NAT) (1)

internal network uses private IPv4 addresses

need to translate to (fewer) public IPv4 addresses

add ‘internal IP+port’ to connection tracking state

use that info to rewrite packets

proto	remote IP + port	public IP + port	internal IP + port
TCP	128.143.67.8:443	198.51.100.17:43232	192.168.1.54:59549
TCP	128.143.67.8:443	198.51.100.17:59948	192.168.1.13:59549
UDP	216.239.32.10:53	198.51.100.17:39554	192.168.1.2:31923

NAT illusion

NAT illusion:

private IP address communicating directly with public IP

inside network, talking to outside:

- use private local address
- use public remote address
- never see router's address

outside network, talking to inside

- use public local address
- use router's public address

network address translation (NAT) (2)

distribute private IPv4 addresses from on internal network

- most common use case: home routers for IPv4
- also used by many companies, ISPs, etc.

can support tons of connections with one IPv4 address

- recall: different (remote IP+port, local port) = diff connnection

can use multiple public IPs if risk of running out of port numbers

- likely common in big NAT installations

endpoint-independent mapping

recall: UDP supports receiving from multiple places with on socket

so maybe our table entry should be:

proto	remote IP + port	public IP + port	internal IP + port
UDP	(any)	198.51.100.17:39554	192.168.1.2:31923

also might make sense for TCP

- some applications may deliberate reuse source port

most common NAT implementation, but limits number of hosts that can be supported

(can also achieve this effect by choose public IP+port consistently)

- example: hash of private IP + port

mapping versus filtering

packet filtering can be separate from translation

NAT table:

proto	remote IP + port	public IP + port	internal IP + port
UDP	(any)	198.51.100.17:39554	192.168.1.2:31923

connection table for filtering:

proto	remote IP + port	public IP + port	internal IP + port
UDP	203.0.113.34:4444	198.51.100.17:39554	192.168.1.2:31923
UDP	192.0.2.99:8999	198.51.100.17:39554	192.168.1.2:31923

running servers inside NAT (1)

so far: can't accept connections on the private network

simple solution: sysadmin configures port forwarding

basically static connection table entries:

proto	remote IP + port	public IP + port	internal IP + port
TCP	(any)	198.51.100.17:443	192.168.1.100:443
TCP	(any)	198.51.100.17:22	192.168.1.100:22

running servers inside NAT (2)

often want to accept connections not configured by sysadmin

example: direct video call between two users

- would be better to send directly

some classes of solution:

- ask router to add table entry
- coordinate with other end to setup connection
- go through relay

extra issues:

- two hosts behind same NAT? nested NATs?

running servers inside NAT (2)

often want to accept connections not configured by sysadmin

example: direct video call between two users

- would be better to send directly

some classes of solution:

- ask router to add table entry
- coordinate with other end to setup connection
- go through relay

extra issues:

- two hosts behind same NAT? nested NATs?

router protocols

several (related) protocols for router-helped NAT traversal

- Port Control Protocol, NAT Port Mapping Protocol, UPnP Internet Gateway Protocol

discovered via UDP multicast and/or DHCP

all provide:

- way of learning next external IP
- way to requesting externally accessible port

typical have lease times for external ports

- host is expected to renew periodically

running servers inside NAT (2)

often want to accept connections not configured by sysadmin

example: direct video call between two users

- would be better to send directly

some classes of solution:

- ask router to add table entry
- coordinate with other end to setup connection
- go through relay

extra issues:

- two hosts behind same NAT? nested NATs?

using learned mappings

if endpoint-independent NAT:

choose local (private) IP address + port

contact external server to learn corresponding public IP address + port

- protocol for this: STUN (RFC 5389)

communicate that IP address + port to other end

- example: via video call setup server

both connect to other IP address + port

complications (1)

potential issue: firewall rule might block unsolicited packets

- might need entry in connection table to get packet not dropped

for UDP: both ends send packet first to setup firewall rule

for TCP: simulatenous connection attempts may work

- TCP state machine supports 'simulatenous open'
- probably one SYN needs to be resent

complications (2)

potential issue: "clever" NATs corrupt addresses

one bad NAT idea: automatically change private IP + port to public IP + port in packet contents

...just look for matching bytes and substitute them! (don't worry about understanding the protocol)

shouldn't do this: probably corrupt downloads/break things

- some NATs did/do it anyways

STUN workaround: 'obfuscate' address+port with XOR

running servers inside NAT (2)

often want to accept connections not configured by sysadmin

example: direct video call between two users

- would be better to send directly

some classes of solution:

- ask router to add table entry
- coordinate with other end to setup connection
- go through relay

extra issues:

- two hosts behind same NAT? nested NATs?

external relays

RFC 8656: Traversal Using Relays around NAT (TURN)

support for setting up relays dynamically

STUN + TURN often used with WebRTC

WebRTC = web browser video/audio-conference support

video conferencing provider might run STUN/TURN servers

- info passed to WebRTC-based webapp

exercise: nested NATs

how do these solutions deal with nested network address translation?

when, if ever, will these solutions break

1. A and B ask NAT for external port

2. A and B learn external IP+port and use simulatenous UDP connection

3. A and B use external relay

aside: address reuse in NAT

with nested network address translation might have...

home network (10/8) $\leftrightarrow$ ISP's NAT (10/8) $\leftrightarrow$ internet ('real')

conceptually nothing prevents this from working

- NAT box translates 10.x.y.z addresses to different 10.a.b.c address
- routing with 10.w.x.y 'gateway' also specifies interface
- (yes, can't contact other IPs in ISP's 10/8 network easily from home network, but probably don't care)

...but confuses a lot of implementations

reason for special CGNAT IP space 100.64/10

- supposed to only be used by devices that can handling nesting like this

fancier firewalls?

some additional actions we might like from firewalls:

more options for actions

- log interesting packets for later
- send alert to sysadmin about weird activity
- trigger block of host sending traffic
- ...

fancier rules

- pattern matching on TCP stream
- pattern matching on HTTP URIs
- pattern matching on app-layer stuff
- heuristics, machine-learning-based rules
- ...

IDS / IPS

IDS = intrusion detection system

- usually ‘passive’ monitoring
- logging lots of stuff for analysis later
- sometimes ‘alerting’ sysadmin/security team

IPS = intrusion prevention system

- IDS + acting on alerts automatically

example Snort rule [reformatted]

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (
msg:"MALWARE-CNC Win.Trojan.Rovnix variant outbound connection";
flow:to_server, established; http_method; content:"POST";
http_uri; content:"/vbulletin/post.php?qu=",fast_pattern,nocase;
http_header; content:!"User-Agent:"; content:!"Accept";
metadata:impact_flag red,ruleset community;
service:http;
reference:url,www.virustotal.com/en/file/a184775757cf30f9593977ee0344cd6c5
classtype:trojan-activity;
sid:34868;
rev:2;
)
```

Zeek

Zeek — old (c. 1996) open source IDS or 'Network Security Monitor'

different focus than Snort (though supports similar things)

main idea (in default config): produces log files for later analysis

examples:

- list of all seen servers and software versions
- list of all SSH connections
- copies of downloaded files
- list of all HTTP URIs

backup slides