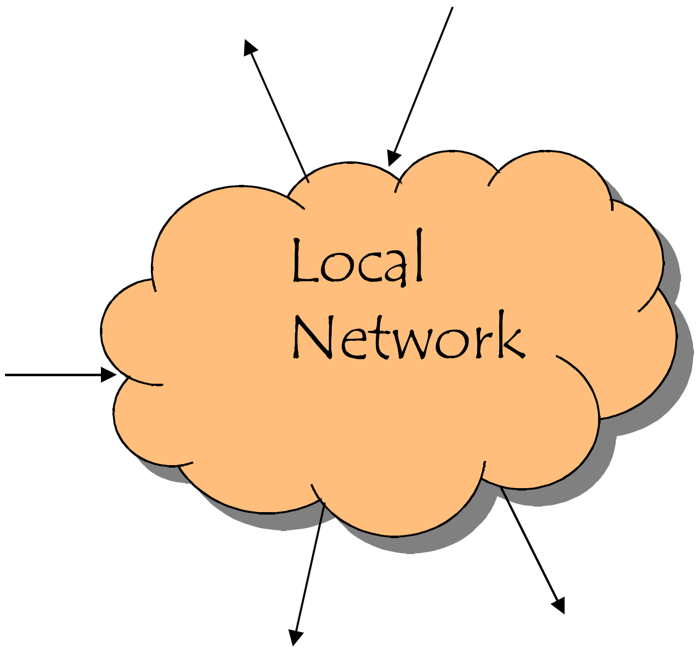
IDS

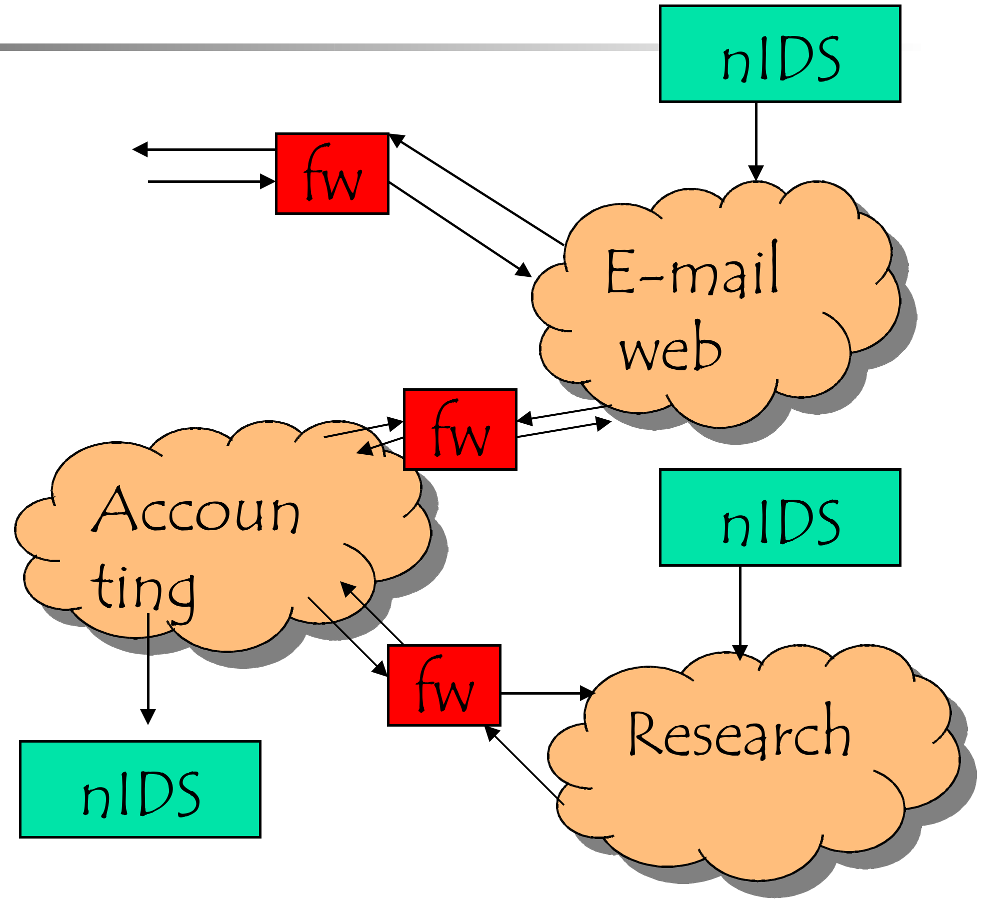
Which actions can be automatically taken as soon as an IDS discover an attack?

- any action on the target system is correct: kill an internet connection increase the amount of data recorded in a log, ends some user sessions
- No offensive security ie action against other systems, eg the attacker one, for two reasons:
  - Stepping stones
  - False positives

Intrusion Detection System

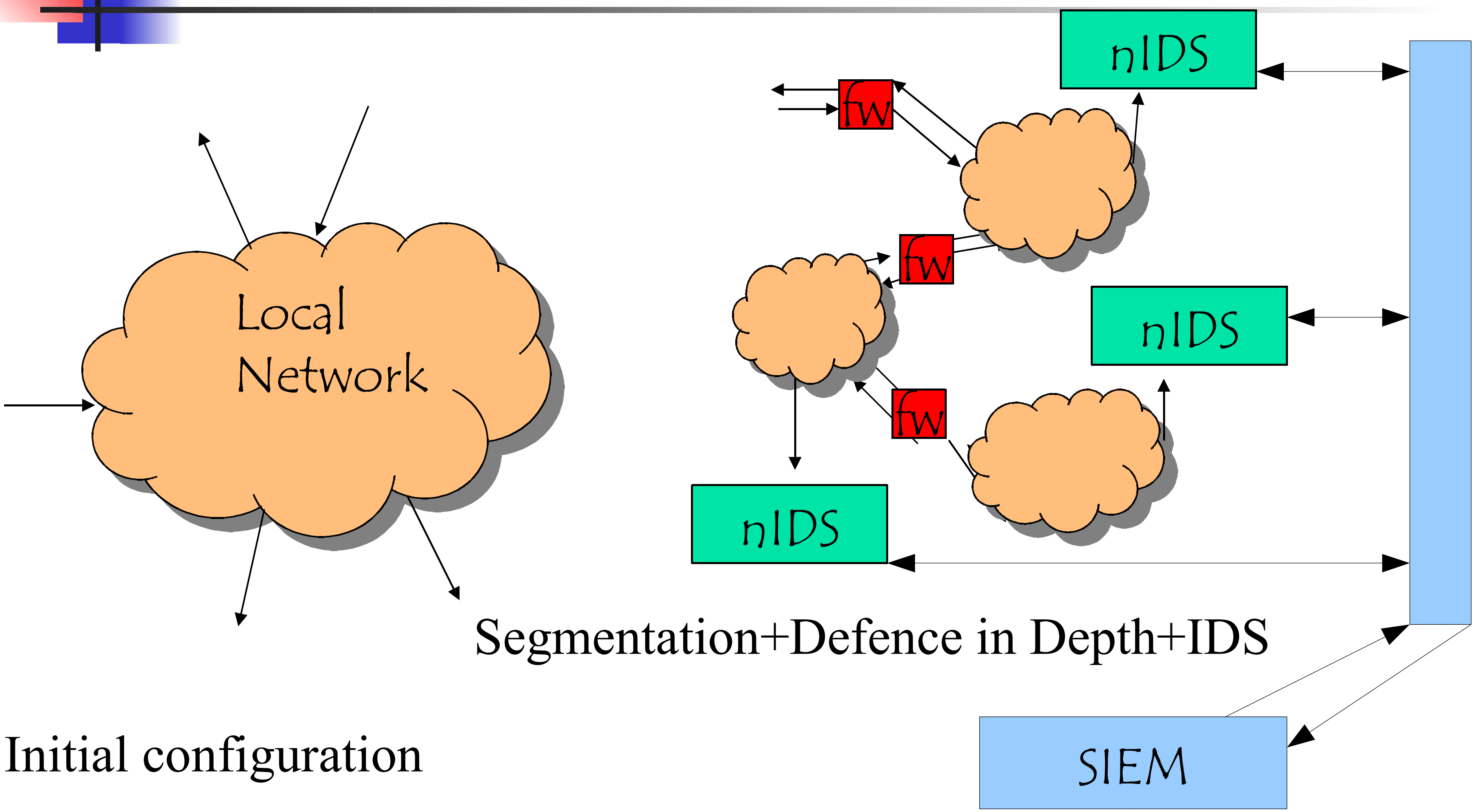


Initial configuration



Segmentation+Defence in Depth+IDS

Security information and event management =SIEM



Segmentation+Defence in Depth+IDS

Initial configuration

Sensors

- Two kind of sensors
  - off-line: analyze the system and user logs to discover attacks that have been implemented and their impact
  - real-time: analyze the current system behavior to discover ongoing attacks and stop them before they are successfull

- real time sensors
  - Some compromises have to be accepted = minimize the number of control to avoid a loss of performance
  - Hardware supports, eg similar to the routing one for NIDS

- Off line = CIDF, common intrusion detection framework standard for logs

NIDS vs HIDS sensors

- hIDS
  - It filter the requests from a user process to the OS, the OS executes only requests that it has not rejected
  - It may slow down a host but it controls any request
- nIDS is not involved in the service that manages a given packet, there is no way to slow down the receiving host

  ⇒ NIDS has to be executed on a dedicated host to analyze all the information flows

hIDS and nIDS technologies

- Base element that is analyzed
  - IP packects and protocol events for a nIDS
  - OS call for a hIDS
  - They can be generalized if the hierarchy of virtual machines is considered
    - String of vm invocations for a hIDS
    - A sequence of information for a nIDS

nIDS: some problems

- Fragmentation of IP packets
- Analysis of a TCP stream (reordering ..)
- Protocol analyis
- Normalization of a protocol to handle all those cases that are not defined by a standard (overlapping IP packets)

hIDS and nIDS technologies

- Anomaly detection
  - By observing a system, we build a database that stores the normal system behavior = measures of normal behavior
  - Signals behaviors that differ more than a predefined threshold
  - Zero day exploit can be detected (after the attack)
- Signature specification based
  - Default allow (attack signatures have to be specified)
    - A database storing attack signatures
    - At run time it signals any behavior matching one in the database
    - The update of the database is critical
  - Default deny = legal behavior has to be specified

N&H-IDS: anomaly detection

First step: interesting measures

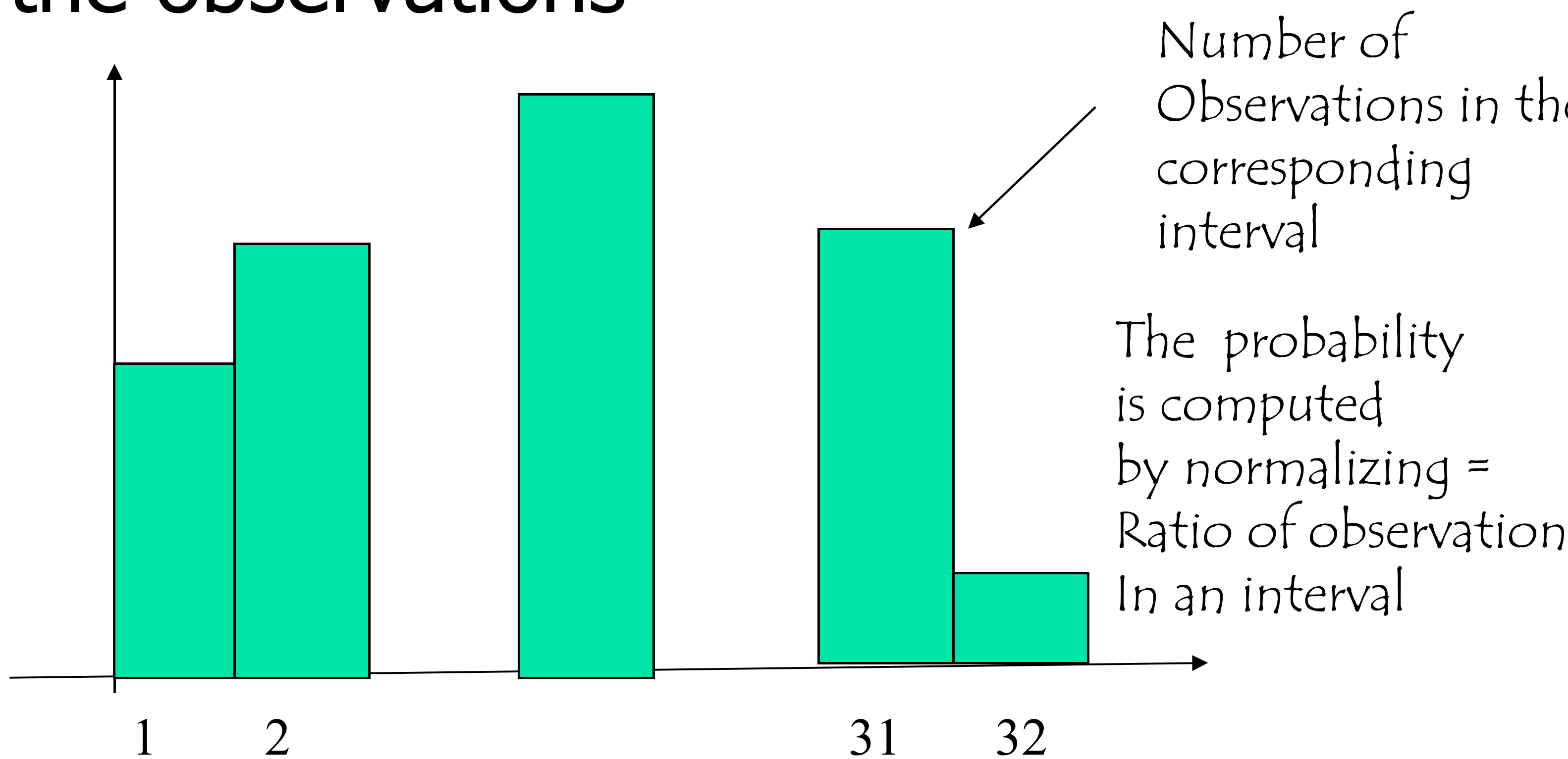
- Number of open file
  - global & for each user
- Number of open port
  - global & for each user
- Frequency of commands
- Number of connected user
- Time when a user connects
- Usage of system resources

N&H-IDS: anomaly detection

- An histogram is built by observing the system and by using a number of intervals (eg 32)
- The intervals are chosen so that the last one include less than 1% observations
- We monitor the system for a time interval (we observe the value of interest at each minute, for 30 days) and build the distribution that pairs each interval with a probability = long term distribution
- We monitor the system for a shorter interval (eg. at each minute for two hours) and build a short term distribution
- An anomaly arises if the two distributions differs

Generating a distribution

- Defined starting from an histogram of the observations



N&H-IDS: anomaly detection

- The difference between two discrete distributions is the sum of the absolute differences between the two values in the corresponding intervals
- Dist= $\Sigma \, |long_i - short_i|$
- Distinct distributions of the same measure are generated by distinct observation frequency or for distinct sets
  - Open files
    - Read the number at each minute or at each hour
    - Read the number for each user or group of users

N&H-IDS: anomaly detection

The observations collected to build the short term distribution and rise the alarms are also used to

- Update the long term distribution to mimic the system evolution (a weigthed sum is used)
- The long term distribution is updated at predefined times (eg at the end of the day) rather than in real time

N&H-IDS: anomaly detection

- The overall system behavior may be seen as a learning system
- Initially, the system learns its normal behavior = initial long term distribution
- The learning and the discovery of anomalous behavior are a life long property of the system as the long term distribution is updated

N&H-IDS: anomaly detection

- The definition of anomaly is related to a user defined threshold
- A large threshold corresponds to a large difference among behaviors $\Rightarrow$

  A few false positives, several false negatives
- A small threshold corresponds to a small difference among behaviors $\Rightarrow$

  A few false negatives, several false positives
- Different measures, different set of measures correspond to distinct ROC curves

Anomaly detection: the foundation

- Nides = next generation intrusion detection system defined in 1991
- To protect military systems
- First rigorous definitions of long and short term distributions
- Measure
  - Continuous = any value
  - Categorical = one value in a predefined range
  - Binary
  - IDS related = The IDS activity is measured as well

NIDES - SRI - Continuous - I

UCPU	User CPU time
SCPU	System CPU time
IO	Number of character exchanged in an application execution
MEMCMB	Largest amount of memory to execute the application
MEMUSE	Sum of the amount of memory used multiplied by the time it has been used = KByte*seconds.

NIDES - Continuous -II

TEXTSZ	Size of a segment
OPENF	Number of open file
PGFLT	Number of memory faults
PGIN	Number of disk pages read
PRCTIME	Elapsed time
SIGNAL	Number of received signals

NIDES - SRI - Categorical

UID	New user name if changed
HOUR	Hour when the application began
RNETHOST	Name of the remote host that has invoked the program
LNETHOST	Name of the local host that has invoked the program
RNETTYPE	Name of the application invoked by the remote host

NIDES – SRI - Binary

RNET	Application executed on a remote host
LNET	Application executed on a local host

NIDES – IDS related

INTARR	continuous	Seconds from the last record
I60	continuous	Number of audit records produced in 1 min
I600	continuous	Number of audit records produced in 10 min
I3600	continuous	Number of audit records produced in 1 hour

NIDES – Learning time - I

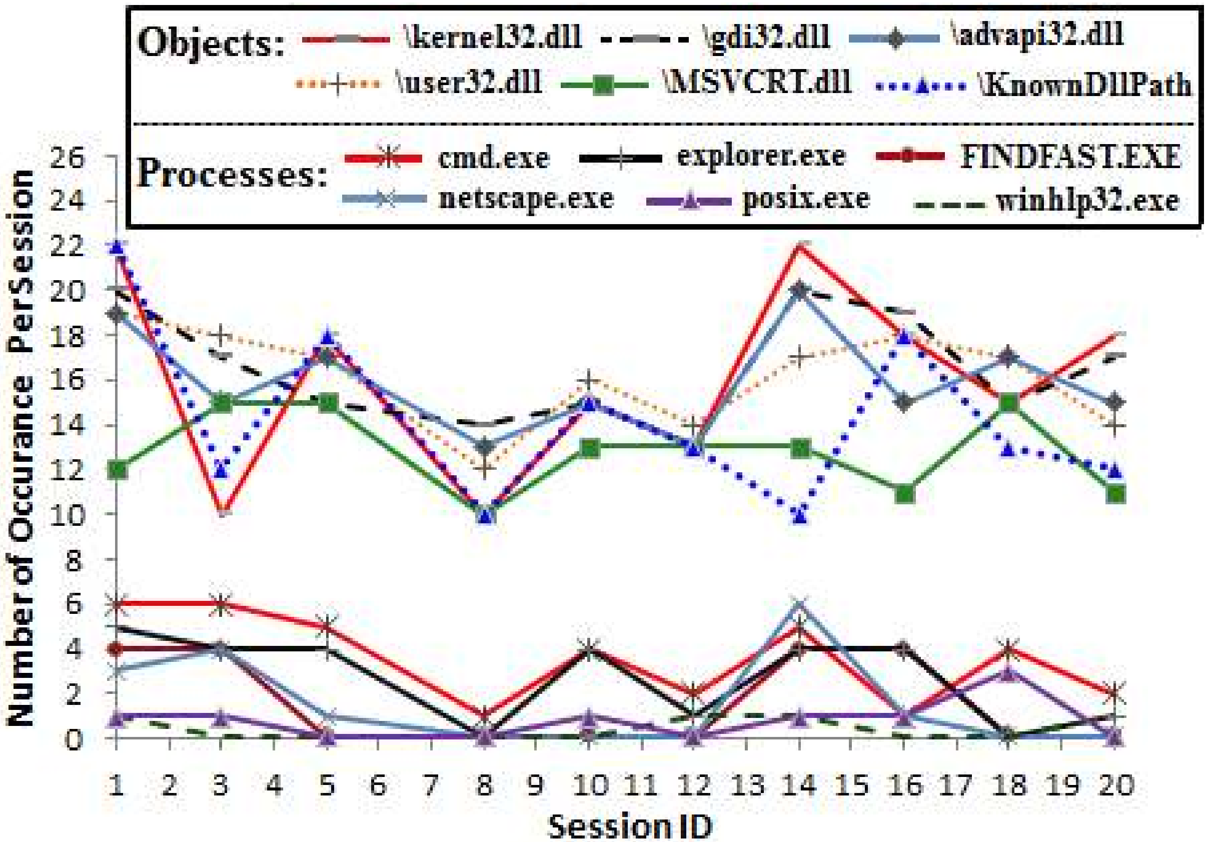
Subject (Application)	Total Records	Training Records	Testing Records	Unique Days
as	1688	1539	149	39
cat	1195	1058	137	68
ccom	886	736	150	36
compile	1010	838	172	43
cp	378	273	105	60
cpp	2625	2470	155	44
csh	909	709	200	57
diff *	690	596	94	46
discuss	1328	1040	288	60
emacs	7929	6227	1702	84
finger *	619	537	82	78
fmt	1819	1522	297	64
gawk *	613	530	83	56
getfullnm *	353	269	84	52
ghostview *	320	225	95	50
grep	5685	3474	2211	60

NIDES – Learning time - II

latex	928	758	170	52
less	5409	4709	700	80
ls	9020	7368	1652	78
mail *	613	527	86	60
make	1251	1095	156	52
man	938	708	230	60
more	8015	6497	1518	68
mymoreproc	3901	3406	495	77
pwd	1405	1181	224	62
rm	2539	2184	355	82
sed	1801	1464	337	64
sort	891	702	189	61
stty	1003	871	132	68
vi	5452	4663	789	77

Detecting Masqueraders in Clouds based on Security Events and NetFlow Data Analysis

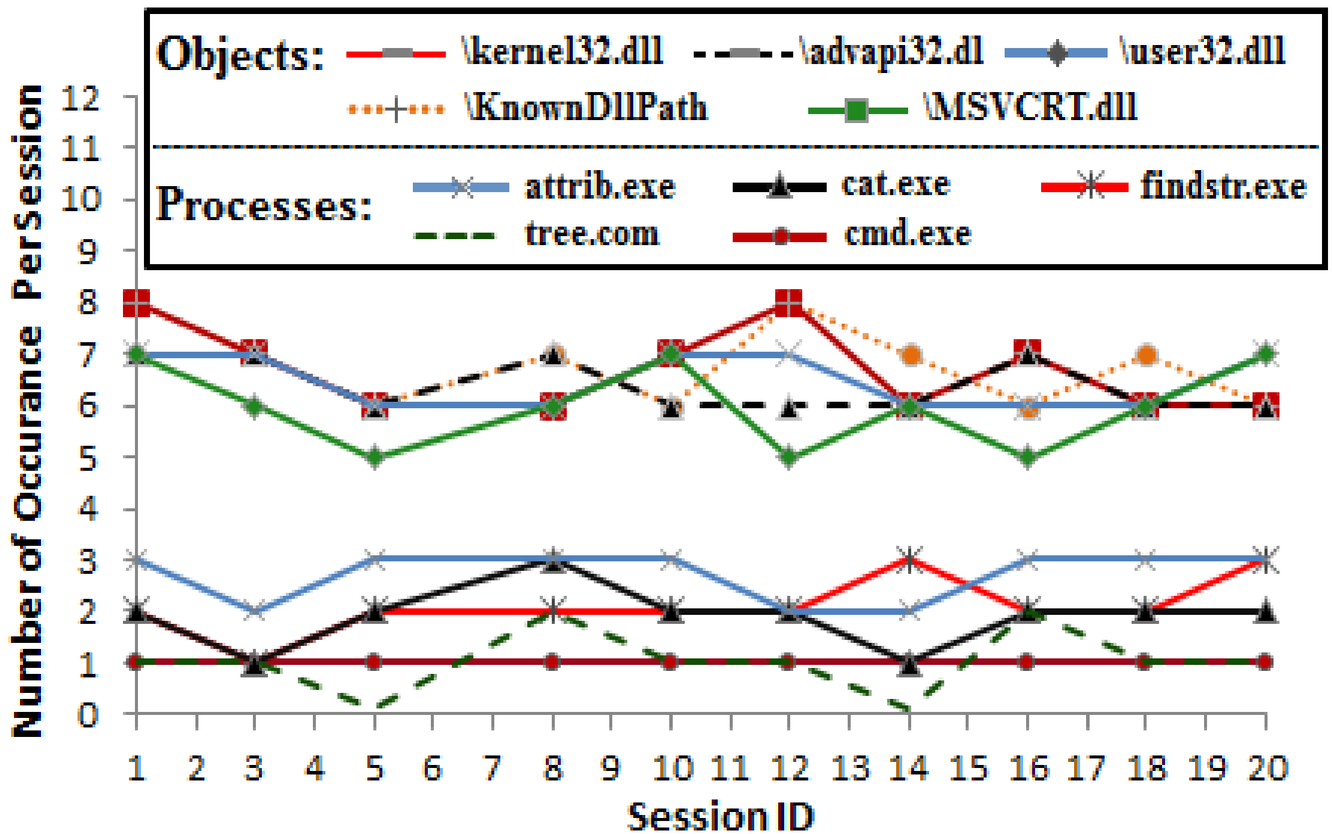
Hisham A. Kholidy, Fabrizio Baiardi, and Salim Hariri



A real user

Detecting Masqueraders in Clouds based on Security Events and NetFlow Data Analysis

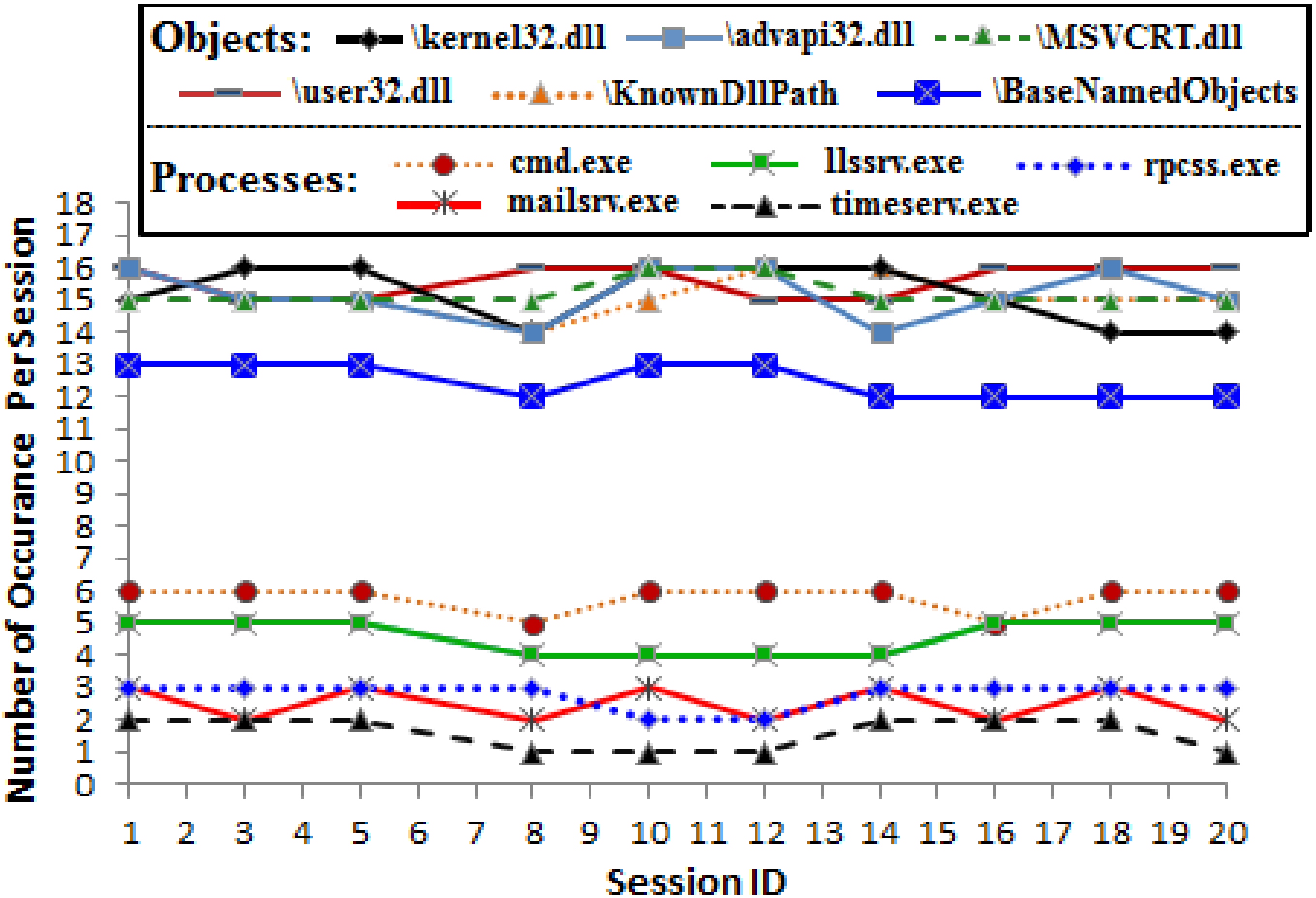
Hisham A. Kholidy, Fabrizio Baiardi, and Salim Hariri



A server

Detecting Masqueraders in Clouds based on Security Events and NetFlow Data Analysis

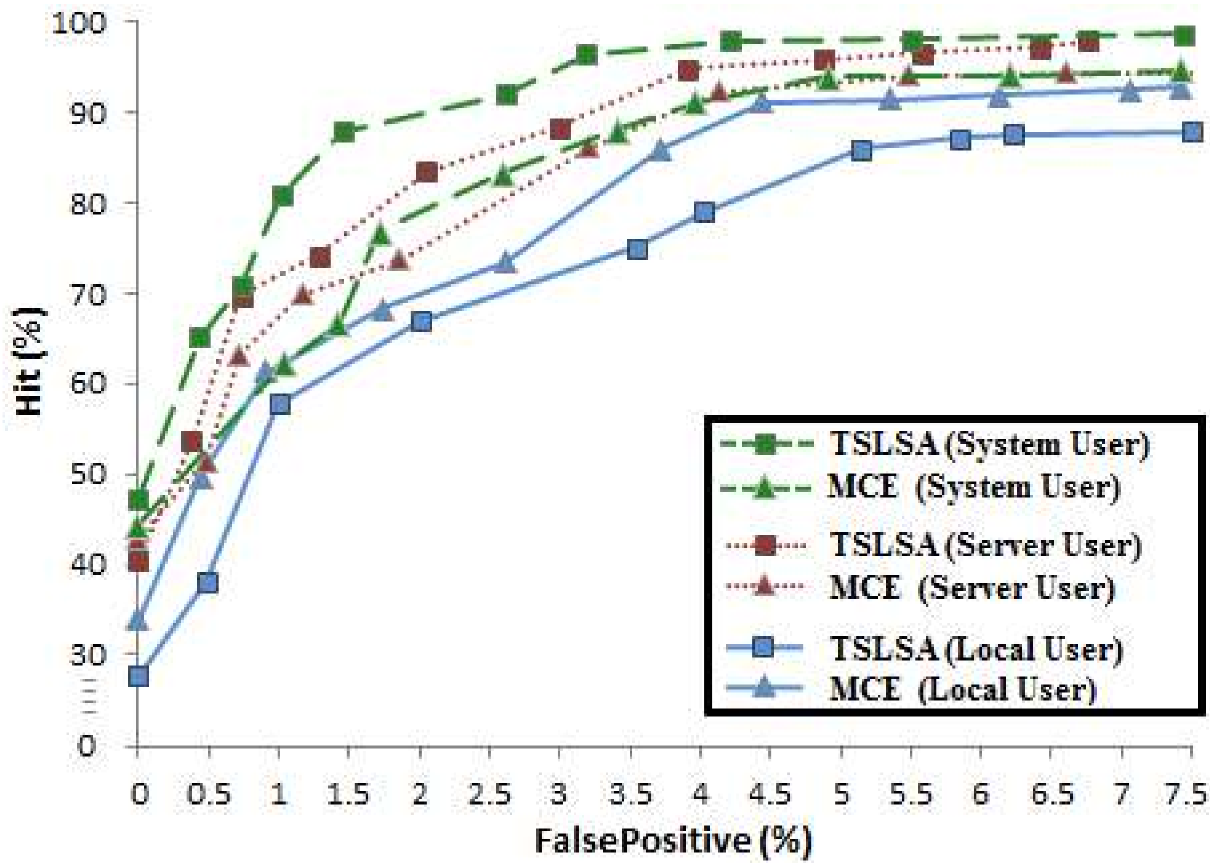
Hisham A. Kholidy, Fabrizio Baiardi, and Salim Hariri



An OS process

Detecting Masqueraders in Clouds based on Security Events and NetFlow Data Analysis

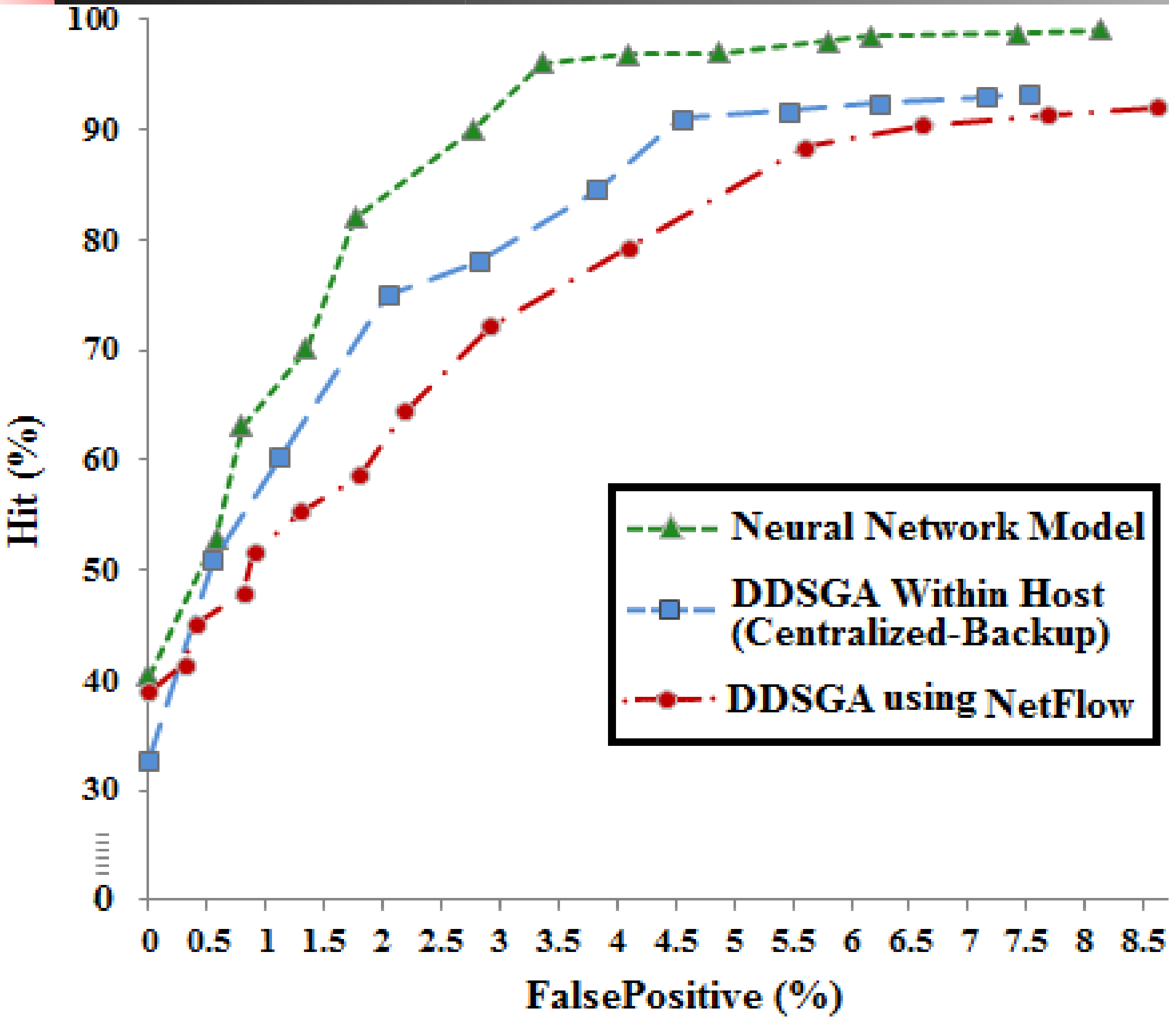
Hisham A. Kholidy, Fabrizio Baiardi, and Salim Hariri



ROC curves

Detecting Masqueraders in Clouds based on Security Events and NetFlow Data Analysis

Hisham A. Kholidy, Fabrizio Baiardi, and Salim Hariri



ROC curves
For local +networks events

N&H-IDS: signature (or misuse) detection

- The overall behavior strongly resembles an antivirus
- A pattern database (signature) for known attacks, each action is matched against each pattern
- Currently an antivirus may store the patterns in a server in a cloud that checks the actions
- Any matching is recorded
- Anytime a pattern has been fully matched, an alarm is fired

N&H-IDS: signature detection

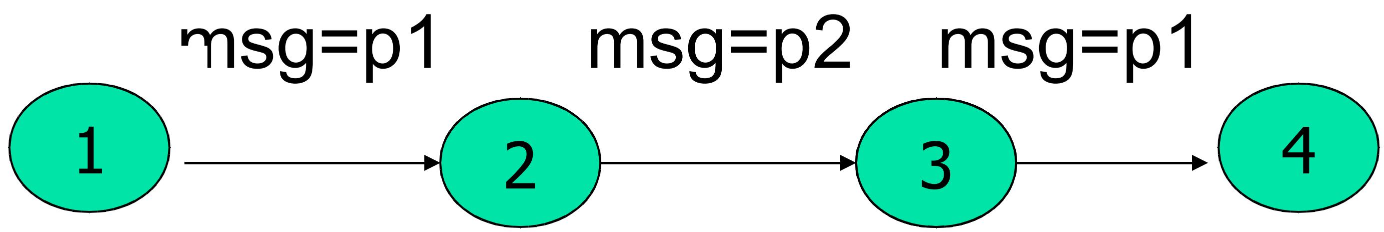
A new challenge

- Describe an attack against a system where the IDS stores its signature database in a cloud
- List some countermeasures

N&H-IDS: signature detection

- Wrt to Antivirus some differences:
  - Dynamic generation of the elements to be matched
  - Unknown time inbetween two consecutive generations
  - An element can match several patterns
- The complexity is much larger for an IDS than for an antivirus that has to match a sequence of characters in a file against a set of patterns
- Cloud power does not help an IDS

N&H-IDS: signature detection



- If the recognizer is currently in state 3 and a packet = p1 is sniffed then the next state may be
  - The one following 3 = 4
  - The one following 1 = 2
- A nondeterministic behavior is required = the status of the automata is both 2 and 4

Nimbda Signature (log)

GET /scripts/root.exe?/c+dir

GET /MSADC/root.exe?/c+dir

GET /c/winnt/system32/cmd.exe?/c+dir

GET /d/winnt/system32/cmd.exe?/c+dir

GET /scripts/..%5c../winnt/system32/cmd.exe?/c+dir

GET /_vti_bin/..%5c../..%5c../..%5c../winnt/system32/cmd.exe?/c+dir

GET /_mem_bin/..%5c../..%5c../..%5c../winnt/system32/cmd.exe?/c+dir

GET /msadc/..%5c../..%5c../..%5c/..\xc1\x1c../..\xc1\x1c../..\xc1\x1c../winnt/system32/cmd.exe?/c+dir

GET /scripts/..\xc1\x1c../winnt/system32/cmd.exe?/c+dir

GET /scripts/..\xc0/../winnt/system32/cmd.exe?/c+dir

GET /scripts/..\xc0\xaf../winnt/system32/cmd.exe?/c+dir

GET /scripts/..\xc1\x9c../winnt/system32/cmd.exe?/c+dir

GET /scripts/..%35c../winnt/system32/cmd.exe?/c+dir

GET /scripts/..%35c../winnt/system32/cmd.exe?/c+dir

GET /scripts/..%5c../winnt/system32/cmd.exe?/c+dir

GET /scripts/..%2f../winnt/system32/cmd.exe?/c+dir

HTTP-WHISKER-SPLICING-ATTACK-SPACE

Signature Snort compatible (snort,prelude,etc)

alert TCP $EXTERNAL any -> $INTERNAL 80 (msg: "IDS296/web-misc_http-whisker-splicing-attack-space"; dsize: <5; flags: A+; content: "|20|"; classtype: suspicious; reference: arachnids,296;)

Signature Dragon Sensor

T D T B 10 0 W IDS296:web-misc_http-whisker-splicing-attack-space /20

Defenseworx Signature

1 B 6 T 0 80 [IDS296/web-misc_http-whisker-splicing-attack-space] "\20"

Pakemon Signature IDS296/web-misc_http-whisker-splicing-attack-space tcp * 80 "|20|"

Shoki Signature

tcp and (dst port 80) and (ip[2:2] > ((ip[0:1] & 0x0f) + (tcp[12:1] & 0xf0) + 5)) and (tcp[13]&16!=0) 65536 SEARCH IDS296 web-misc_http-whisker-splicing-attack-space '0x20' ALL 1 NULL

N&H-IDS:
signature detection & evasion

- When sniffing a packet P the NIDS has no mean to anticipate
  - Whether P will be received
  - How P will be handled
- An attacker can iniject in the monitored network packets to hide other ones or to confuse the IDS (eg packet with a wrong checksum that the receiver will discard)
- Encrypted traffic is a further problem

N&H-IDS:
MITRE TACTIC TA005: Defense Evasion

- The rise of defense evasion is mostly due to better detection and protection technologies and increased adoption rates.
- Attacks that once slipped trivially past network and endpoint defenders are now routinely caught, and adversaries need a way of circumventing security controls
- With the exception of discovery, more techniques that relate to defense evasion are observed in systems than any other MITRE ATT&CK™ tactic.
- Persistence is also adopted to defend against better detection mechanisms

Bypassing NIDS - Fragmentation

- NIDS must reconstruct fragments
  - Maintain state = drain on resources
  - Must overwrite correctly = more drain on resources
- Target server correctly de-frags
- Attack #1 - just fragment
- Attack #2 - frag with overwrite
- Attack #3 - start an attack, follow with many false attacks, finish the first attack

Bypassing NIDS - TCP un-sync

- Inject a packet with a bad TCP checksum
  - fake ‘FIN’ packet
- Inject a packet with a weird TCP sequence number
  - step up
  - wrapping numbers

Bypassing NIDS – TTL attack

- This is an attack against the synchronization of the IDS and the end host and requires a router between the IDS sensor and the end host.
- A packet crafted with a TTL equal to the number of hops of the router will result in a packet examined by the IDS but never reaching the end host, thus desynchronizing the end host and the IDS.
- It can be thwarted by a NIDS that examines the TTL field and understands the network topology at the expense of a larger overhead

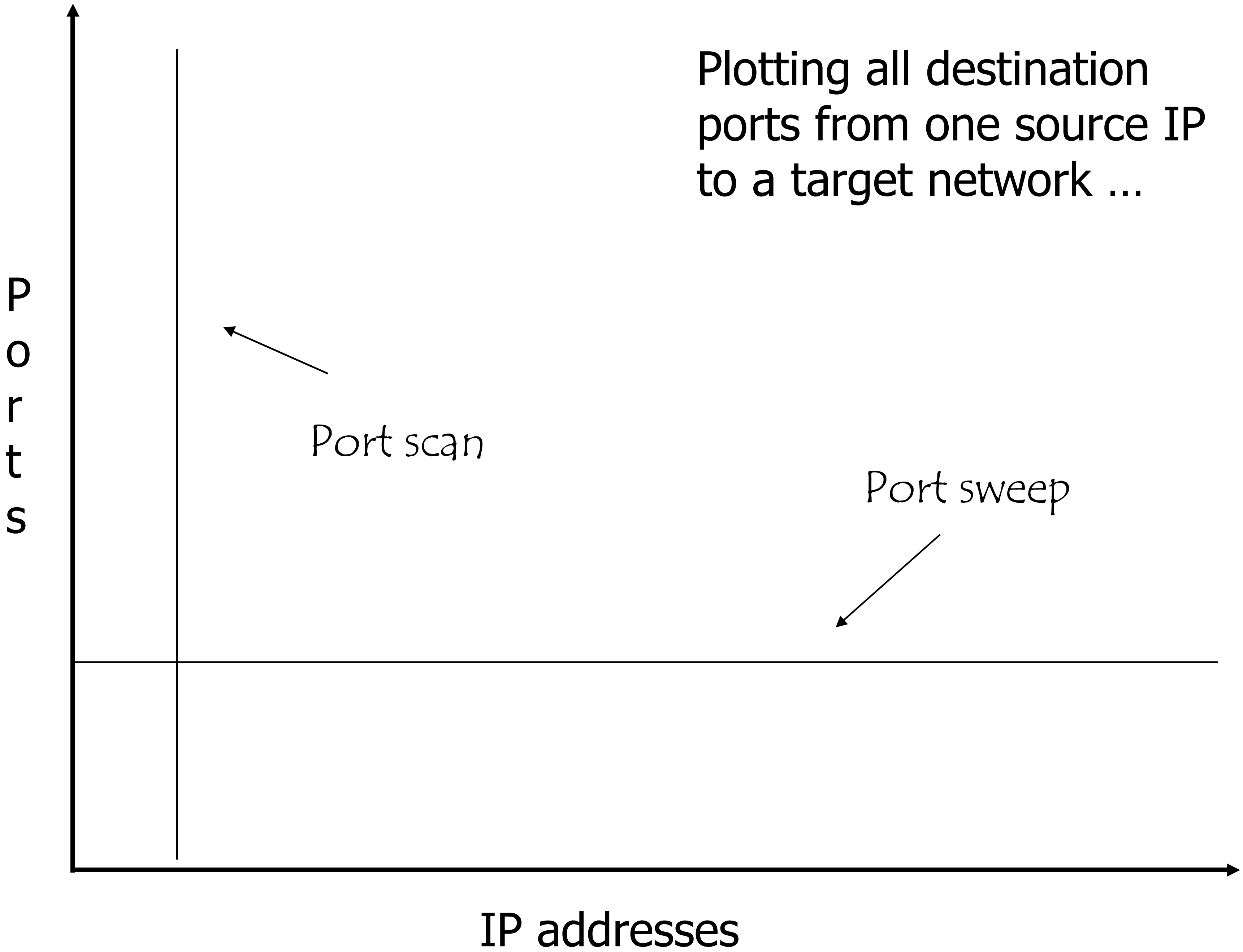
NIDS - Overwhelming

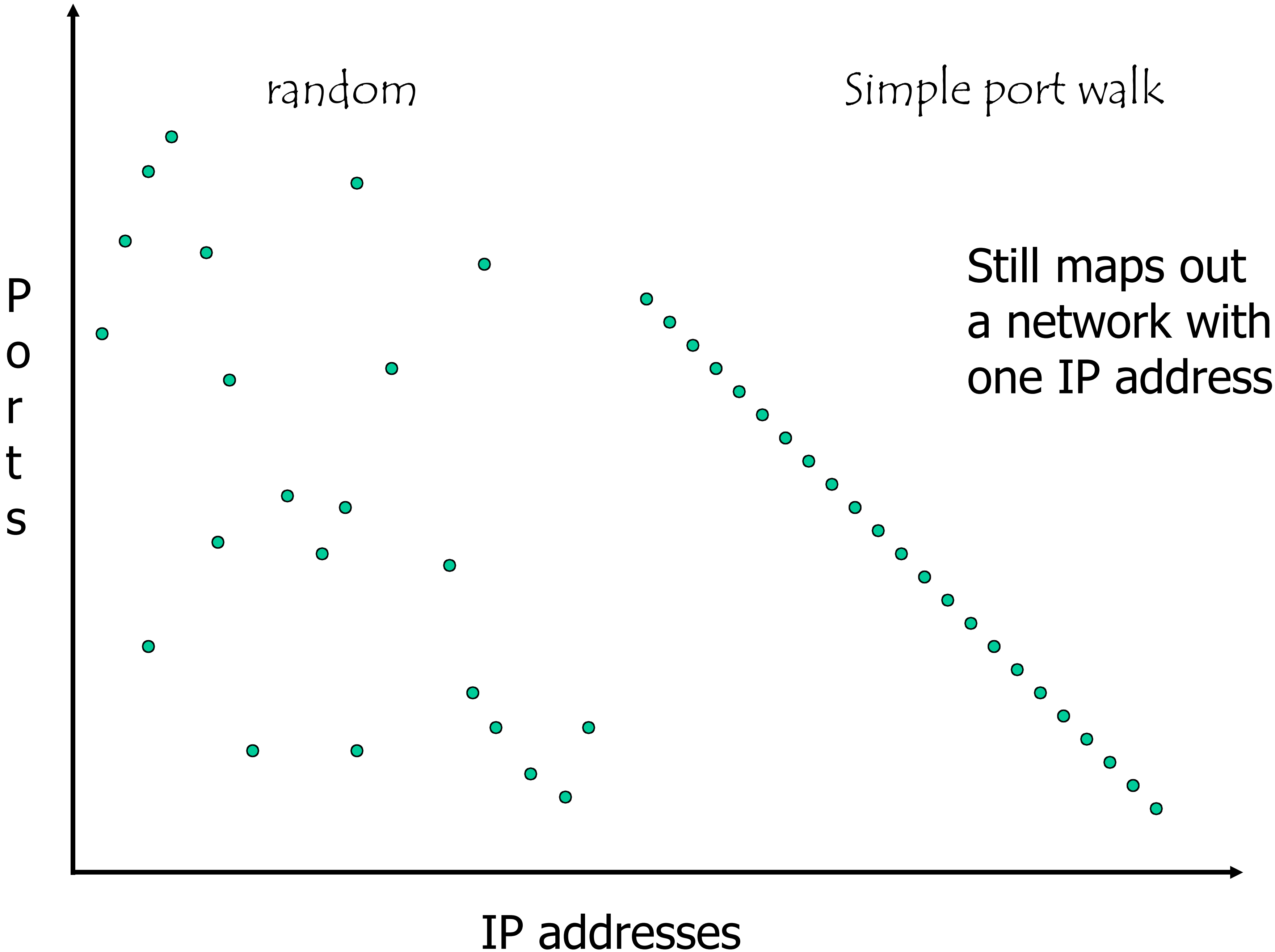
- Send as many false attacks as possible while still doing the real attack
  - May overload console
  - May drop packets
  - Admins may not believe there is a threat
- Send packets that “cost” the NIDS CPU cycles to process
  - Fragmented, overlapping, de-synchronized web attacks with the occasional bad checksum

NIDS - ‘Slow Roll’

- Detect port scans and sweeps
  - Obvious: incremental destination ports
  - Trivial: randomized ports
  - Sweep: one port and many addresses
  - Stealthy: random ports and addresses over time

Plotting all destination ports from one source IP to a target network …





random
Simple port walk
Still maps out a network with one IP address
Ports
IP addresses

N&H-IDS: signature detection

- New attacks can be detected only if the database is continuously updated and after the update
- The detection of unknown attacks is fully delegated to anomaly detection only
- Anomaly detection can discover a new attack provided that it results in some anomaly for some time

NIDS e HIDS: new attacks??

- An alternative approach considers the IDS as a rule base expert system
  - A rule database rather than a pattern database
  - Rules describe attacks and anomaly
- A generalization (abstraction) procedure can be applied to rules to discover, at least, variants of attacks that are already known

Snort

- Freeware.
- Originally designed as a network sniffer.
- Useful for
  - traffic analysis.
  - intrusion detection.
    - Warning: Has become a target of attackers!
      - What's more fun for them than to find a vulnerability in security software.

Snort

- A good sniffer.
- A detection engine, based on rules.
- Packets that do not match any rule are discarded (only from the analysis in general) or they are logged.
- Rule matching packets can also trigger an alert.

Snort Basics

- Rules try to match intrusions "signatures"
- Examples
  - Directory Traversal Vulnerability
    - Solaris Sadmind/IIS worm (2001)
      - Allowed HTTP GET requests to change to root directory with "../../".
      - Allowed to copy cmd.exe into the Scripts directory.
      - Gained control usually at admin level

GET/ scripts/../../winnt/system32/cmd.exe /c+

copy+\wint\system32\CMD.exe+root.exe

Snort Basics

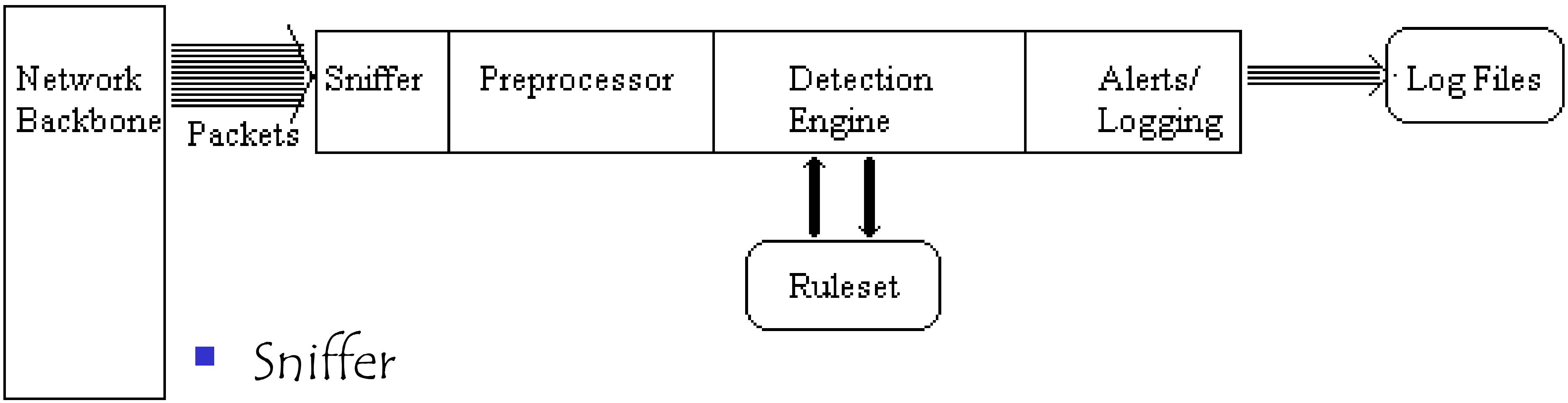
- Code Red Worm 2001
  - Exploited vulnerability in IIS 4.0 and 5.0
  - Buffer overflow vulnerability
  - Footprint:

/default.ida?
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNN
%u9090%u6858%ucbcd3%7801%u9090%u6805%ucb
d3%u7801

Using Snort

- NIDS mode
  - Load snort with a set of rules, configure packet analysis plug-ins, and let it monitor hostile network activity
- Sniff mode
- Logger mode
- IPS mode = if it filters traffic

Snort Architecture



- Sniffer
- Preprocessor
- Detection Engine
- Alert Logging

Snort: Architecture

- Packet Sniffer = Taps into network
- Packet Decode Engine
  - Uses the libpcap package
  - Packets are decoded for link-level protocols, then for higher protocols.
- Preprocessor Plug-ins
  - Each preprocessors examines and manipulates packets, e.g. for alerts.
  - RPC plug-in
  - Port scanner plug-in
- Detection Engine
  - Checks packets against the various options in the snort rules files.
- Detection Plug-Ins
  - Allow additional examinations
- Output Plug-Ins

SNORT Architecture

- Detection Engine
  - Signature-based implemented via rule-sets
  - Rules
    - Consists of rule header
      - Action to take
      - Type of packet
      - Source, destination IP address
      - …
    - And rule option
      - Content of package that should make the packet match the rule

Snort Rules

- Rules header and rule option

alert tcp !10.1.1.0/24 any -> 10.1.1.0/24 any	(flags: SF; msg: "SYN-FIN scan)

Alerts to traffic from outside the 10.1.1.x subnet to the 10.1.1.x subnet with the Syn and the Fin flags set.

The flag combination is illegal method to handle such illegal/abnormal flag combinations is not conveyed in the RFC of TCP. So, such illegal/abnormal flag combinations are handled differently in various operating systems. Different operating system also generate different kind of responses for such packets.

Rule Header

- It defines the "who, where, and what" of a packet, as well as what to do in the event that a packet with all the attributes indicated in the rule should show up.
- The first item is the rule action that tells Snort what to do when it finds a packet that matches the rule criteria.
- There are some available default actions in Snort, other can be defined

Rule header: Action

alert:	generate an alert using the selected method and log
log:	log the packet
pass:	ignore the packet
activate:	alert and then turn on another dynamic rule
dynamic:	idle until activated by a rule, then act as a log rule
drop:	block and log the packet (a filter and not a sniffer)
reject:	block the packet, log it, and then send a TCP reset if TCP or an ICMP port unreachable if UDP
sdrop:	block the packet but do not log it.

Snort Rules

- Rule Header Fields
  - Protocol Field
    - TCP for example SMTP, HTTP, FTP
    - UDP for example DNS traffic
    - ICMP for example ping, traceroute
    - IP for example IPSec, IGMP
    - Others (ARP, RARP, GRE, …) to come

Snort Rules

- Rule Header Fields
  - Source and Destination IP Address Field
    - Format: Address/netmask or any or
      - Address x.x.x.x
      - Netmask = bits of network mask
      - For example
        - 24.0.0.0/8 Class A
        - 24.3.0.0/16 Class b
        - 192.185.67.0/24 Class C
        - 192.185.67.188 host address
    - Special keywords:
      - any
      - ! (negation)
      - $HOME_NET (variable defined elsewhere)

Rule Options

- This section contains alert messages and information on parts of the packet to inspect to determine to take rule action
- All Snort rule options are separated from each other using the semicolon ";"
- Rule option keywords are separated from their arguments with a colon ":"

Snort Rules Options

- Four major categories of rule options.

General :	provide information about the rule but do not affect detection
Payload:	look for data inside the packet payload and can be inter-related
Non-payload:	look for non-payload data
Post-detection:	rule specific triggers that happen after a rule has ``fired."

Snort Rules Options

- Session Options
  - Allows to capture TCP session.
- Rest Option
  - Allows an automatic active response
- Tag Option
  - Allows to dynamically capture additional packets after a rule triggers.

Some options

- msg - prints a message in alerts and packet logs
- logto - log the packet to a user specified filename instead of the standard output file
- ttl - test the IP header's TTL field value
- tos - test the IP header's TOS field value
- id - test the IP header's fragment ID field for a specific value
- ipoption - watch the IP option fields for specific codes
- fragbits - test the fragmentation bits of the IP header
- dsize - test the packet's payload size against a value
- flags - test the TCP flags for certain values
- seq - test the TCP sequence number field for a specific value

Other options

- ack - test the TCP acknowledgement field for a value
- itype - test the ICMP type field against a value
- icode - test the ICMP code field against a value
- icmp_id - test the ICMP ECHO ID field against a value
- icmp_seq - test the ICMP ECHO sequence number against a value
- content - search for a pattern in the payload
- content-list - search for a set of patterns in the payload

Rule Order

- A packet should be checked in the order

  drop > pass > alert > log

- This scheme is the most secure since no packet passes through without being checked against all drop rules
- However most of the packets are normal traffic and do not show any intruder activity. Testing all of the packets against all alert rules requires a lot of processing power.
- A more efficient, but more dangerous order is

  Pass > Drop > Alert > Log

Snort Rules: Example

- Rule Header
  - alert tcp $External_NET any -> $Home_Net21
- Rule Options
  - (msg: “ftp Exploit”; flow_to_server, established; content: “|31c031db 41c9b046 cd80 31c031db|”; reference: bugtraq,1387; classtype:attempted-admin; sid 344; rev4;)

Snort Rules

- Rule Header
  - Action
  - tcp: Protocol being used. UDP / IP / ICMP
  - $External_NET: This is the source IP, default is any.
  - any: This is the source port set to “any”
  - ->: Direction of conversation.
  - $Home_Net: This is a variable that Snort will replace with
  - 21: Port to be monitored.
- The header concerns all tcp packages coming from any port from the outside to port 21 on the inside.

Snort Rules

Rule Options

- ( ): Rule option is placed in parentheses.
- msg: “ftp Exploit”;
- flow_to_server, established;
- content: “|31c031db 41c9b046 cd80 31c031db|”; check if the package contains this string, the dangerous payload.
- reference: bugtraq,1387; links to third-party warnings.
- classtype:attempted-admin; Class Types allow users to quickly scan for attack types
- sid 344; Snort rule unique identifier. Can be checked against www.snort.org/snort-db.
- rev4; All rules are part of a revision process to limit false positives and detect new attacks.

Snort Rules

- Rule Options
  - Msg Option = message to print

Rule:

```
alert udp any any -> 129.210.18.0 / 24 31337 \
(msg: "Back Orifice";)
```

Log:

```
    [**] Back Orifice [**]
05/10-08:44:26.398345 192.120.81.5:60256 -> 129.210.18.34:31337
UDP TTL:41 TOS:0x0 ID:49951
Len: 8
```

Snort Rules

- Rule Options
  - Logto Option
    - Specifies filename to which to log the activity.
    - Allows to separate the annoyances from the truly dangerous.

```
alert udp any any -> 129.210.18.0 / 24 31335 \
(msg: “trinoo port”; logto “DDoS”)
```

Snort Rules

- Rule Options, not paylod
  - TTL option
    - Allows to use the time to live field in packet
    - Format: ttl: number

```
alert udp any any -> 129.210.18.0 / 24 33000;34000 \
(msg: "Unix traceroute"; ttl: 1;)
```

Snort Rules

- Rule Options
  - ID option
    - 16-bit value found in the IP header of each datagram.

```
alert udp any any -> 129.210.18.0 / 24 33000;34000 \
(msg: "Suspicious IP Identification"; ID: 0;)
```

Snort Rules

- Rule Options
  - Dsize option
    - Size of payload

```
alert icmp any any -> 129.210.18.0 / 24 any \
(msg: "Large ICMP payload"; dsize: >1024;)
```

Snort Rules

- Rule Options
  - Sequence Option
    - Value of tcp sequence number
  - Ack option
    - Value of ack number in tcp

```
alert tcp any any -> any any \
(msg: “Possible Shaft DDoS”; seq: 0x28374839;)

alert tcp any any -> any any \
(msg: “nmap tcp ping”; flags: A; ack: 0;)
```

Snort Rules

- Rule Options
  - Itype and Icode Options
    - Select ICMP message type and operations code

```
alert icmp 1.1.1.0/24 any -> 129.210.18.0 / 24 any \
(msg: "port unreachable"; itype: 3; icode: 3;)
```

Snort Rules

- Rule Options
  - Flags option

```
alert tcp any any -> any any \
(msg: “null scan”; flags: 0;)
```

Snort Rules

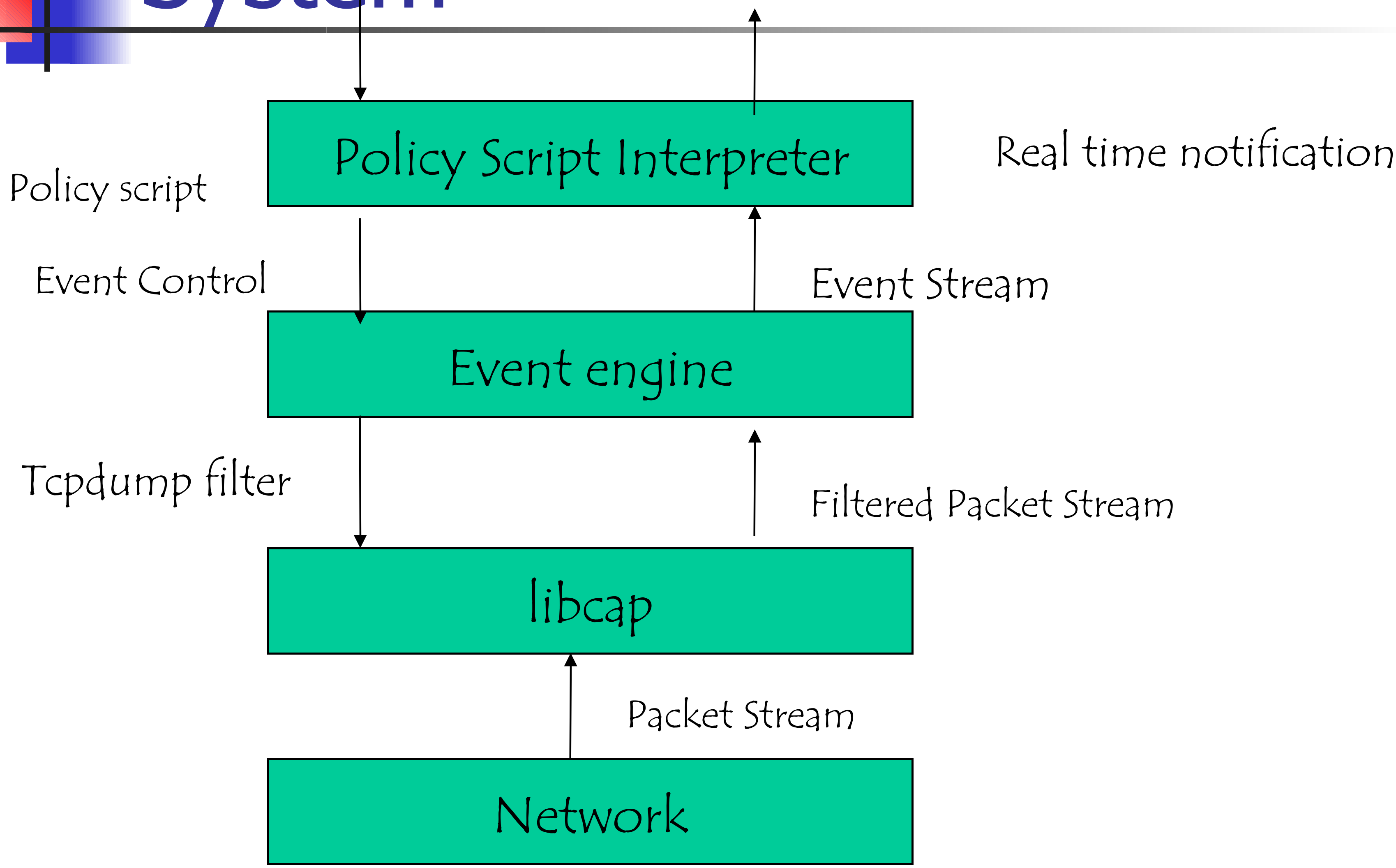
- Rule Options
  - Content Option

```
alert udp $EXTERNAL_NET any -> $HOME_NET 53 \
(msg: “Exploit bind tsig Overflow attempt”; \
content: “|00 FA 00 FF|”; content: “/bin/sh”;)
```

Zeek, previously Bro

- A network analyzer. rules-based engines aims to detect exceptions, Zeek looks for specific threats and trigger alerts.
- Used as a traditional IDS but more frequently to record network behavior, i.e. long-term records of all HTTP requests and results – or tables correlating MAC and IP addresses.
- Zeek stores the network metadata it records more efficiently so that it can be searched, indexed, queried, and reported in ways previously unavailable. This makes it especially well-suited for network anomaly detection and threat hunting.
- A disadvantage is that its deep-packet inspection is resource intensive.

Structure of the Bro (Zeek) System



Bro - libcap

- It's the packet capture library used by tcpdump.
- Isolates Bro from details of the network link technology.
- Filters the incoming packet stream from the network to extract the required packets.

  eg port finger, port ftp, tcp port 113 (Ident), port telnet, port login, port 111 (Portmapper).
- Can capture packets with the SYN, FIN, or RST Control bits set.

Bro – Event Engine

- The filtered packet stream from the libcap is handed over to the Event Engine.
- Performs several integrity checks to assure that the packet headers are well formed.
- It looks up the connection state associated with the tuple of the two IP addresses and the two TCP or UDP port numbers.
- It then dispatches the packet to a handler for the corresponding connection.

Bro – TCP Handler

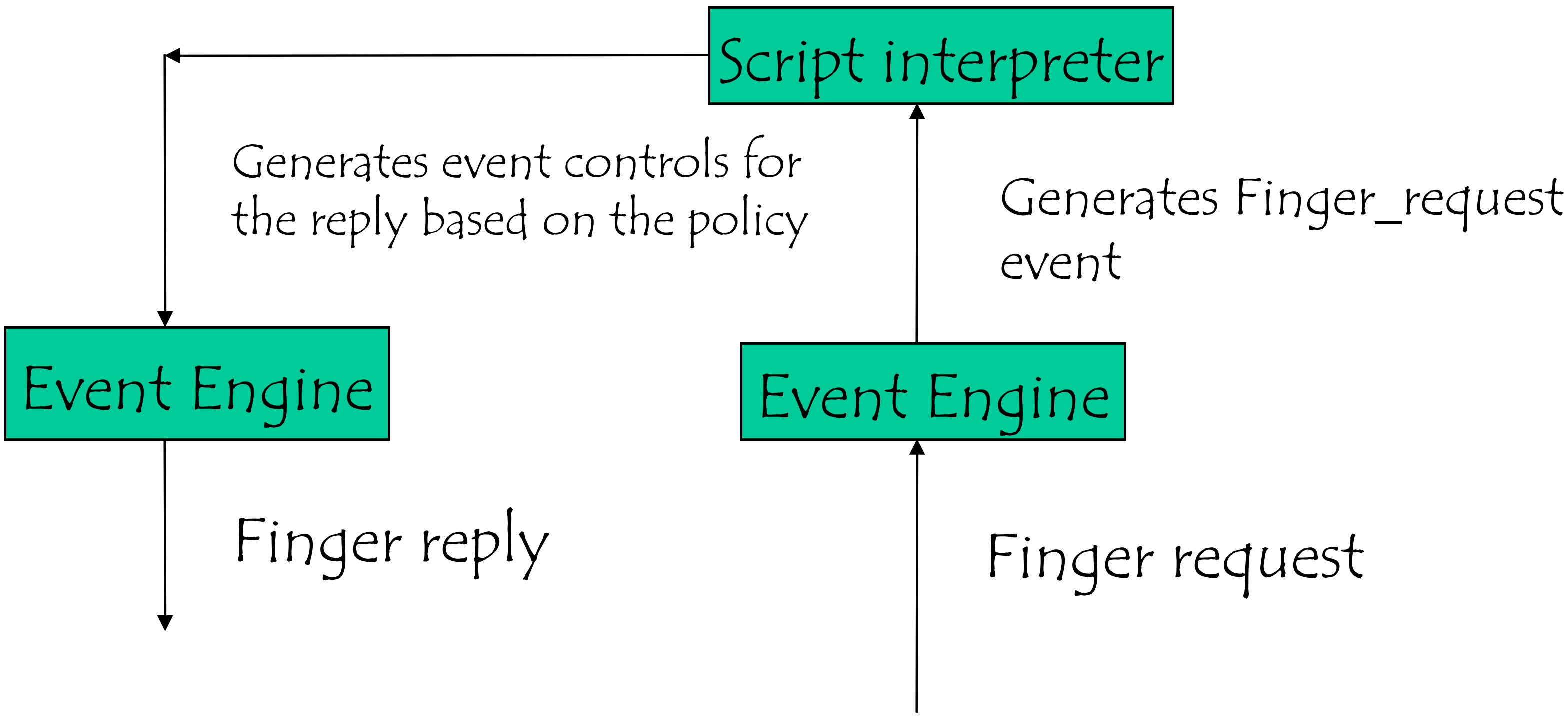
- For each TCP packet, the connection handler verifies that the entire TCP Header is present and validates the TCP checksum.
- If successful, it then tests whether the TCP header includes any of the SYN/FIN/RST control flags and adjusts the connection’s state accordingly.
- Different changes in the connection’s state generate different events.

Policy Script Interpreter

- The policy script interpreter
  - receives the events from the Event Engine.
  - executes scripts written in the Bro language which generates events like
    - logging real-time notifications,
    - recording data to disk
    - modifying internal state.
- To add new functionality to Bro we add a new protocol analyzer to the event engine and then write new events handlers in the interpreter.

Application Specific Processing - Finger

Tests for buffer overflow, checks the user against sensitive ids, etc



Suricata

- Introduced in 2009
- Rules-based compatibility with Snort Rules, multi-threading to process more rules across faster networks, with larger traffic volumes, on the same hardware.
- A multi-threaded instance will balance the processing load across every processor on a sensor to achieve 10-gigabit speeds without sacrificing ruleset coverage,"
- Incorporated the Lua scripting language for greater flexibility to create rules that identify conditions difficult or impossible with a legacy Snort Rule.
- It is a little more involved to install and the community is smaller than what Snort has amassed

Using a pubblic network

- Several institution have to connect remote, local networks into a single infrastructure
- Leased lines are too expensive
- The most convenient connection exploits a pubblic network, eg the internet
- The connection security is very low because information flows on a pubblic network
- This is an instance of the shared connection problem we will meet again in clouds

Countermeasures - Robustness

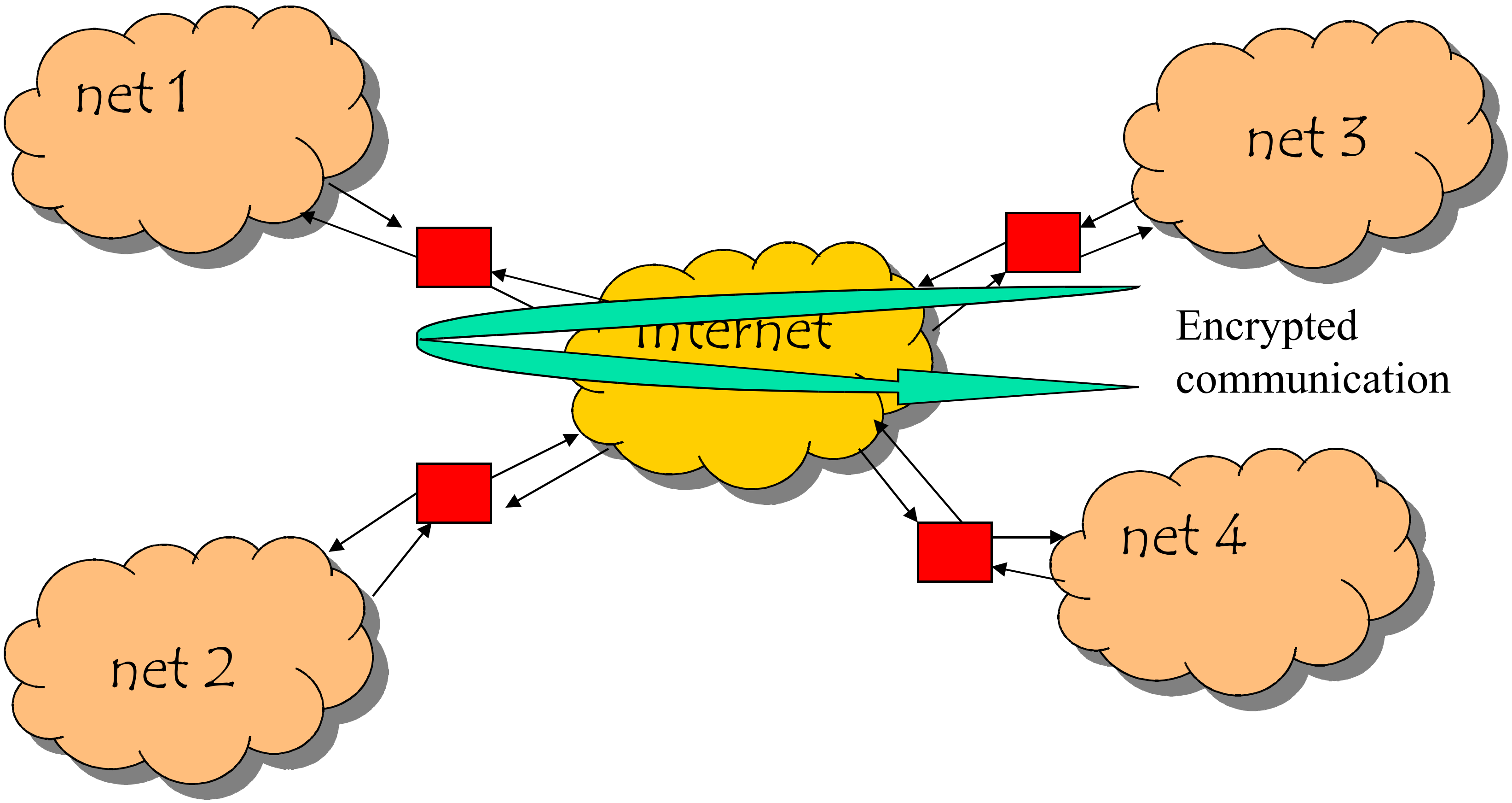
- Virtual Private Network
  - It emulates a secure connection on top of an unsafe connection
  - Assuming that each local network is protected by a firewall, secure connections are established among the firewalls
  - Secure = integrity and confidentiality are achieved by encrypting the traffic between any pair of firewalls

VPN≠VLAN

- VLAN denotes a logical network that is set up to minimise the number of conflicts
- A vlan is built by pairing
  - Transmission frequency
  - Tags

  with a subset of the nodes
- No security property is introduced only for traffic shaping

Virtual Private Network



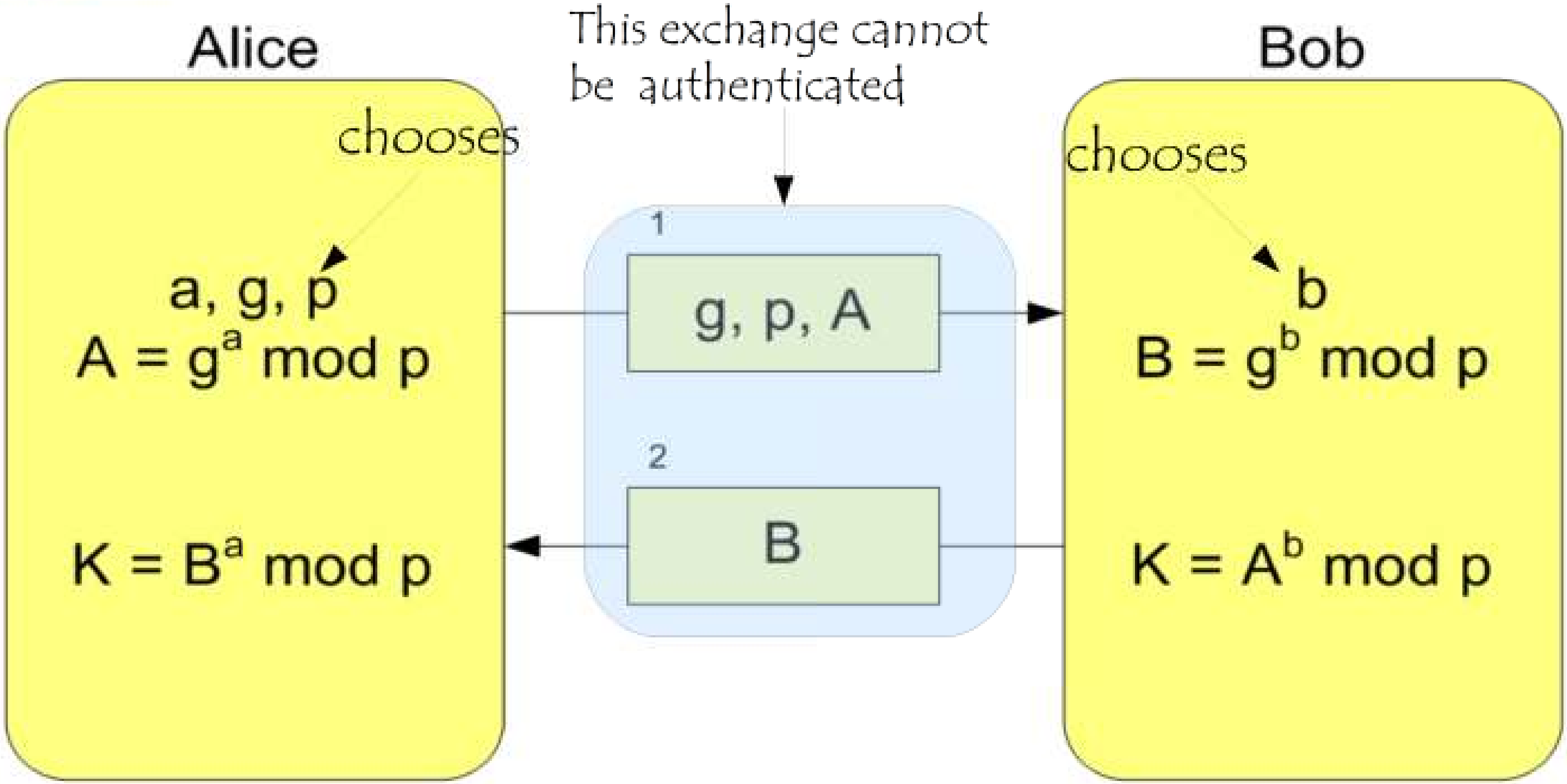
Virtual Private Network

- Symmetric Encryption due to the large amount of transmitted data
- A distinct key for each pair of firewalls
- The key is updated according to the amount of exchanged data
- The key is chosen in a preamble and update when reaching an amount of information that is exchanged

VPN and symmetric encryption - I

- The simpliest strategy to share a key without transmitting it is the Diffie_Helmann protocol
  - each firewall produces a number
  - All-to-all exchange
  - After the exchange, each firewall produce a key for each partner
  - Man-in-the-middle attack

Diffie-Hellman



$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

VPN and symmetric encryption -II

- Each firewall pubblish a pubblic key and know the corresponding secret key
- The two keys makes it possible to compute a symmetric key
- Data to be exchanged is protected with the symmetric key
- IP v6