Firewall and IDS

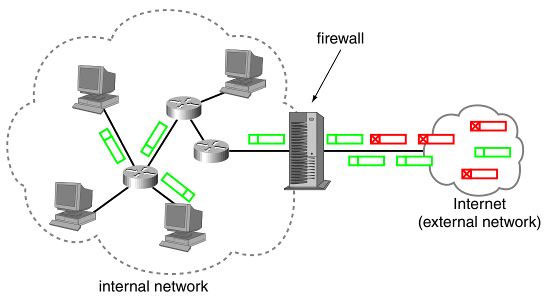
CSC 348·648

WAKE FOREST
UNIVERSITY
Department of Computer Science

Spring 2013

Network Firewalls

- A firewall is located between the Internet and internal network
  - Remains the forefront defense of most computer systems
  - Inspecting traffic, the firewall drops or accepts packets



  - As a result, firewall provides access control between networks
- The firewall applies a security policy to each arriving packet

Firewall Security Policy

- Security policy is an **ordered** list of rules
  - Rules consist of a 5-tuple: protocol type, IP source address, source port, IP destination address, and destination port
  - Fields can be fully specified or contain wildcards '`*`'

		Source		Destination		
No.	**Proto.**	**IP**	**Port**	**IP**	**Port**	**Action**
1	TCP	140.*	*	*	80	accept
2	TCP	150.*	*	120.*	80	accept
3	TCP	140.*	*	130.*	20	accept
4	UDP	150.*	*	*	3030	accept
5	*	*	*	*	*	deny

  - Every rule has an action **accept** or **deny**
- Rules are applied to every packet, (starting with the first rule)
  - If a packet matches a rule, the associated action is performed

A Simple Model of Policies and Rules

		Source		Destination		
No.	**Proto.**	**IP**	**Port**	**IP**	**Port**	**Action**
1	TCP	140.*	*	130.*	20	accept
2	TCP	140.*	*	*	80	accept
3	TCP	150.*	*	120.*	90	accept
4	UDP	150.*	*	*	3030	accept
5	*	*	*	*	*	deny

- Firewall rule consists of a 5-tuple and an action (IP networks)

$$r = (r[1], r[2], ..., r[k])$$

  - $r[l]$ can be fully specified or contain wildcards '`*`'
- Security policy is a ordered list of rules

$$R = \{r_1, r_2, ..., r_n\}$$

  - Packets sequentially compared with rules until *first match*

Matching

- Using the models, can formally describe *processing packets*
- Recall a *first-match* policy typically takes place

  Definition Packet d matches r_i if

  $$d \Rightarrow r_i \quad \text{iff} \quad d[l] \subseteq r_i[l], \quad l = 1, ..., k$$

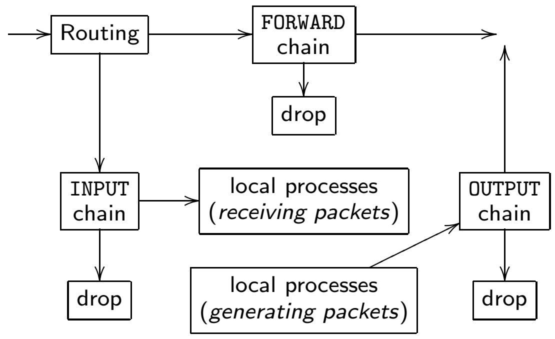
- Assume $d =$ `(TCP, 140.1.1.1, 90, 130.1.1.1, 20)`
  - Using the policy below, $d \Rightarrow r_1$ and $d \Rightarrow r_3$
  - Different actions, but this is **not** considered an anomaly

		Source		**Destination**		
No.	**Proto.**	**IP**	**Port**	**IP**	**Port**	**Action**
1	TCP	140.*	*	130.*	20	accept
2	TCP	140.*	*	*	80	accept
3	*	*	*	*	*	deny

- *Best match* is another policy, but difficult to manage policies

Linux Firewalls

- The Linux firewall mechanism is Netfilter (`iptables`)
  - Has multiple *rule chains*, where each has a policy
  - Chains include `INPUT` (local machine is destination), `OUTPUT` (local machine generated), and `FORWARD` (just passing through)
- The *general* packet flow using Netfilter is


Routing
FORWARD chain
drop
INPUT chain
local processes (receiving packets)
OUTPUT chain
drop
local processes (generating packets)
drop

  - Arriving packets pass through routing, which determines whether to send to `INPUT` or `FORWARD`

- Locally destined packets are processed by the `INPUT` chain, if accepted it is sent to the receiving process
- If the machine is allowed to forward and packet is remotely destined, then processed by the `FORWARD` chain
- Locally generated packets are processed by the `OUTPUT` chain

  So what is a chain?

- It's actually more complicated than described
  1. *Prerouting* called for forwarding and locally destined (NAT)
  2. Appropriate chain is then called
  3. If forwarding, packet managling is called, allows QoS
  4. If generated locally generated, *postrouting* is called (NAT)
  5. Packet sent to the device or process

Netfilter Rules

- `iptable` rules allow more than the 5-tuple
  - Can specify the MAC address and ToS field

    Why? Can you always filter MAC of the source?

  - `iptables` can also provide rate limiting
- Some example `iptable` commands to add different rules

```
iptables -F INPUT
iptables -P INPUT DROP
iptables -A INPUT -i eth0 -s 127.0.0.0/8 -j DENY
iptables -A INPUT -p UDP --dport 2045 -j DROP
iptables -A INPUT -p UDP -s 0/0 -d 0/0 --dport 53 -j ACCEPT
iptables -vnL
```

Default accept or deny, which is better?

Firewalls and State

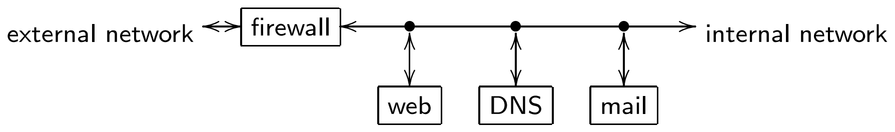
- The previous firewall examples are **packet filters/screens**
  - Process packets based on a *static* rule-set
  - No **state** information is stored
- Stateful firewalls maintain connection information
  - Assume a connection is established from the internal network
  - Firewall keeps track of this connection
  - Packets that arrive from the external network must be part of an **existing** connection
- Example state rule in `iptables`

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

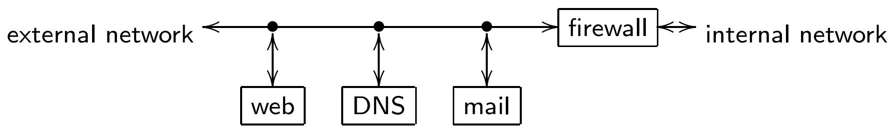
What is the disadvantage of maintaining state?

Firewall Topologies

- Consider a single firewall between internal and external networks
  - Assume there are web, DNS, and mail servers

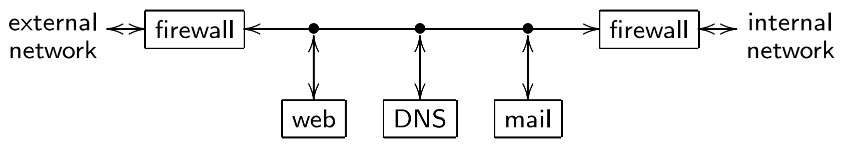


  - Simple implementation but single point of failure
- Place servers outside of the internal network



Advantages and disadvantages?

- Common to use multiple firewalls
  - Create a demilitarize zone (DMZ), containing Internet servers
    "The DMZ is a network that separates an external network from a purely internal network"



  - Firewall separates internal network from DMZ and Internet
  - Protects against compromised first firewall or DMZ
  - Can use multiple DMZ and build *enclaves*
    Advantages and disadvantages?

- *Can use personal firewalls at every machine...*

Proxy

- A firewall mediates access to a network
  - Allowing and disallowing certain type of access
- A proxy is an intermediate agent or server
  - Acts on behalf of endpoints without allowing direct connection
- A proxy firewall uses proxies to perform access control
  - Direct connections are not allowed between endpoints
  - Bases access control on content of the packets
  - Also called a *bastion host*

Other Firewall Issues

- Multiple connections per application
  - Applications may one connection for data, another for control
  - Firewall may know one connection, must be aware of the other

    For example?

- Dynamic connections per application
  - We know that *"IP addresses and TCP ports play an essential role in the binding a client and a remote target object"*
  - Applications (CORBA) may dynamically create connections
  - CORBA needs location transparency and P2P communication

  - Large range of port numbers possible *that firewall must allow*

    "A packet filter firewall located between client and server is likely to be configured in a way that remote invocations are blocked. The application will be unable to complete the request. To enable all remote invocations on objects behind the packet filter firewall, the firewall would have to be opened for connections to all hosts running CORBA servers. A broad range of port numbers would have to opened on the firewall: any port a CORBA server could be listening on which is potentially any non-privileged port."

    So what? What aboout NAT?

- *As a result firewalls may need to become application aware*

  So what? Isn't this what IDS does...

Improving Firewall Performance

- Firewalls remain the first defense for most systems
  - Applying policy to all arriving packets
  - Must manage increasing volumes of packets, increasing number of rules, and strict **latency requirements**
- Two general methods for improving performance
  - Reorganize the rule list
  - Improve rule search algorithm
  - Perform search in parallel
- In all the methods must **maintain first match**

Policy Optimization

- Reorder policy rules to reduce the average number of comparisons
  - More *popular* rules should appear first
  - Must maintain policy **integrity**
- Rule list has an implied **precedence relationship**

		Source		Destination			
No.	Proto.	IP	Port	IP	Port	Action	Prob.
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29

  - r_1 must appear before r_2, r_3, r_5, and r_5 must be last
  - However relative order of r_2 and r_3 can change

Modeling Precedence

- Precedence modeled as a Directed Acyclical Graph (DAG)
  - Nodes are rules, edges are precedence relationships
  - Edge exists between r_i and r_j, if $i < j$ and the rules intersect

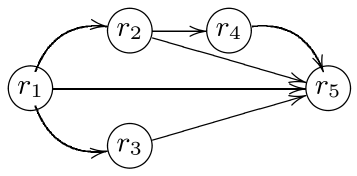
Definition The intersection of rule r_i and r_j, denoted as $r_i \cap r_j$ is

$$r_i \cap r_j = (r_i[l] \cap r_j[l]), \quad l = 1, ..., k$$

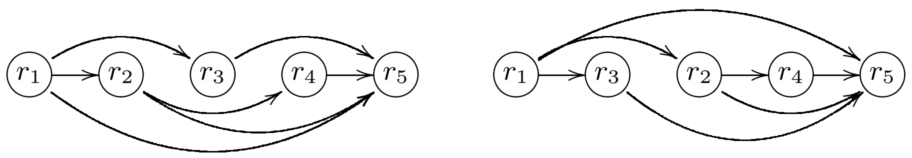
- *Why is the intersection appropriate?*
  - Intersection describes the set of packets that match both rules
  - If rules intersect, then order is significant **for maintaining integrity**

Modeling Precedence

		Source		Destination			
No.	Proto.	IP	Port	IP	Port	Action	Prob.
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29



- Seek a linear arrangement of the DAG to improve performance



- Several linear arrangements, *which is best?*

Optimality Criterion

- Every firewall rule has a match probability (*hit ratio*)
  - Develop a *policy profile* over time $\{p_1, ..., p_n\}$
  - Where p_i is the probability of matching r_i
- Want to minimize the average number of rule comparisons

$$E[n] = \sum_{i=1}^{n} i \cdot p_i$$

- Determining optimal order is same as job-shop scheduling
  - Job-shop scheduling is $\mathcal{NP}$-hard so is optimal firewall rule ordering

Simple Rule Sort

- Swap out-of-order rules if no precedence edge exists between them
  - Repeat until all rules are in order...

```
done = false
while (!done)
   done = true
   for ($i = 1$; $i < n$; $i$++)
      if ($p_i < p_{i+1}$ AND $r_i \not\rightarrow r_{i+1}$)
         interchange rules $r_i$ and $r_{i+1}$ and probabilities $p_i$ and $p_{i+1}$
         done = false
         end
   end
end
```

Any problems with the above algorithm? Will it maintain integrity?

Intrusions, Events, and Detection

- **Intrusion** *is a set of actions that attempt to compromise the integrity, confidentiality, or availability of any resource on a computing platform*
- Attacks manifest themselves in terms of *events*
  - Events can have different granularity (from packets to logs)
  - Each attack step/phase/action has some associated *event*
- **Intrusion Detection Systems (IDS)** monitor the system
  - Analyze information about system and network activities
  - Looks for evidence of malicious behavior
  - **Goal for IDS** is to analyze one or more event streams and **identify manifestations of attacks**

IDS Categories Based on Events

- IDS can be categorized based on the use of event streams
  - **Anomaly detection** or **misuse detection**
- **Anomaly detection** attempts to find *abnormal behavior*
  - Must first define *normal behavior* (based on history)
  - System attempts to identify patterns of activity that deviate
  - *Recognize normal events, not an attack*
- **Misuse detection** is the complement of anomaly detection
  - Have *known* attack descriptions (**signatures**)
  - Events stream are constantly matched against the signatures
  - *Don't recognize normal, know attack events*

IDS Categories Based on Scope

- Can further categorize IDS based on *scope*: **network** or **host**
- **Host** implemented on a single machine
  - Only responsible for the host on which it resides
  - Maintains/obeserves audit files, system calls, etc...
  - For example tripwire
- **Network** implemented in a centralized or distributed fashion
  - Only responsible for the network
  - Measures traffic and/or scans packet data
  - For example Network Flight Recorder (NFR)
- *Neither category is comprehensive... only applicable to certain types of attacks*

Basic IDS Operation

Regardless of the category/type of IDS, they all do the following

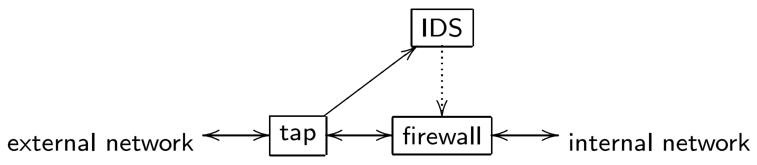
1. **Data Collection** - Collect system data
   - Network based - Collect traffic using a sniffer software
   - Host based - Process activity, memory usage, and system calls
2. **Feature Selection** - Reduce data, create feature vectors
   - Network based - Packet header information, payload, ...
   - Host based - User name, login time and date, duration...
3. **Analysis** - Determine if vector contains signature (misuse detection) or whether the data is anomalous (anomaly detection)
4. **Action** - Alert and possibly automatically stop/minimize attack

Misuse Detection

- Known attack patterns create a library of attack signatures
  - Data that matches a library entry is considered an attack
  - An example signature based NIDS is **snort**
- Snort supports header and payload inspection of network traffic
  - User can define a rule and action that is applied to packets
  - Library is rule list applied to packets, there is a x match policy
  - Actions include: alert, log, pass, activate, dynamic
  - *Rules are ordered based on the action...*
  - *Older versions of Snort had stateless inspection*

Snort Placement

- Snort (and most IDS) is most commonly used in *stealth mode*
  - Use a network tap and send duplicate traffic to IDS
    Why use a tap?
  - *Snort in-line is placed in the traffic stream...*
- Integrating firewall and IDS, Instrusion Protection System (IPS)
  - Have the IDS send rules to the firewall



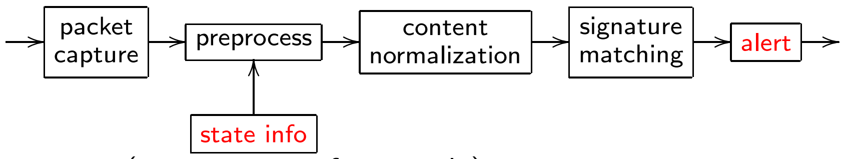
  - Can also use the firewall to *avoid* IDS for legitimate traffic

Misuse Detection Advantages/Disadvantages

- Very low **false alarm rate** (only attacks match)
  - If an alert given, high probability it is an attack
- Only applicable to known attacks
  - Attack variations can defeat detection (**neighboring attack**)
  - **Novel attacks** are not detected
- Can be resource intensive since inspecting every attack
  - Stateful systems are very slow
  - *Methods for rule optimization? High-speed IDS?*

What happens when all traffic is encrypted?

Snort Operation



- Packet capture (`libpcap`... unfortunately)
- Preprocessing performs various operations
  - Flow detection, reassembly, and manage state
- Content normalization
  - Change content to common form (e.g. '`%41`' to '`A`')
  - *Otherwise think about all the signature variations*
- Detection engine applies the set of rules to *packet streams*
  - Scan the payload for a certain signature (string match)
- Alert engine performs the matching rule action

Snort Rules

- Snort rules have two parts, **rule header** and **rule options**
  - Header describes the action and packets to consider
  - Options provides more details packet attributes (if needed)
- For example, consider the following snort rule

```
alert tcp any any -> 10.1.1.0/24 222 (content:"|00 11 22 33|"; msg:"rpcd request")
```

  - Rule header *alerts* when TCP traffic is observed originating from any network with any source port, destined for network 10.1.1.x to destination port 222
  - Keyword `content` in option field requires the payload to be searched for the pattern

More Snort Rules

```
alert tcp any any -> any 80 (content:"abcde"; nocase; offset:5
depth:15; content:"fghi"; distance:3 within:9);
```

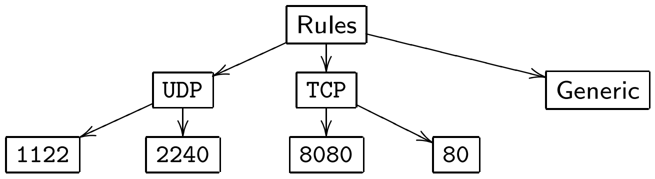
- Example above contains multiple `content`s
  - Additional keywords specify the `content`, case, and locations

```
alert tcp any any -> any 80 (content:"mode=admin";
uricontent:"/newsscript.pl");
```

- Above example contains URI content
  - URI content is normalized and processed separately
- There are over 5000-ish Snort rules, *will it ever decrease?*

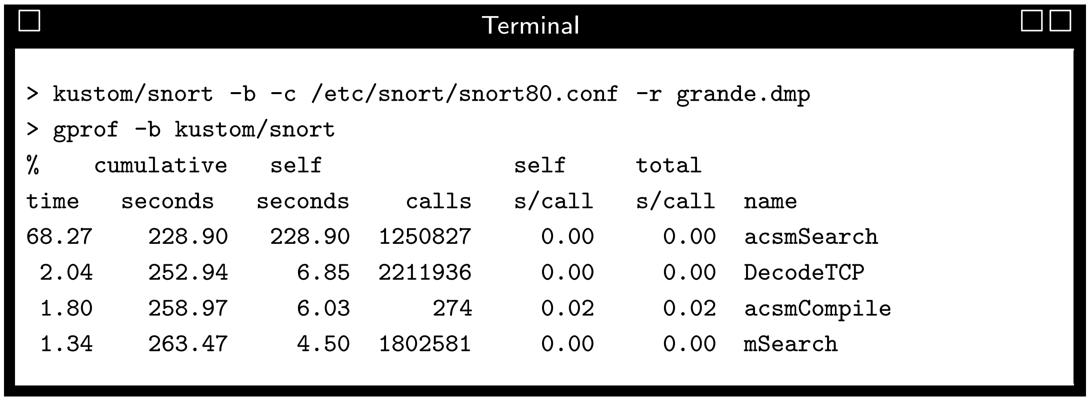
Rule Groups

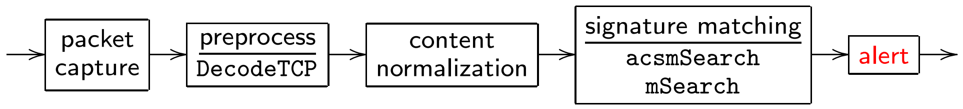
- Do not necessarily want to compare a packet against every rule
- Snort divides rules into **groups**
  - Snort supports rules for TCP, UDP, IP, and ICMP protocols
  - Within each protocol rules are divided into groups
  - Each rule is placed in a group based on source/destination port



- When a packet arrives the content is compared against rules in
  - Port groups associated with the packet
  - Generic port group

Specifically, What is the Problem?


```
> kustom/snort -b -c /etc/snort/snort80.conf -r grande.dmp
> gprof -b kustom/snort
%     cumulative   self              self     total
time    seconds   seconds    calls   s/call   s/call  name
68.27     228.90   228.90  1250827     0.00     0.00  acsmSearch
 2.04     252.94     6.85  2211936     0.00     0.00  DecodeTCP
 1.80     258.97     6.03      274     0.02     0.02  acsmCompile
 1.34     263.47     4.50  1802581     0.00     0.00  mSearch
```



- Content searching is **very** time consuming
  - Others have reported from 40% to 75%

Improving IDS Performance

- Improve the content search/match algorithms
  - Quickly search payloads for multiple signatures
  - Consider signature length when designing algorithms
  - *Good, but perhaps the improvement is not enough*
- Parallelize certain IDS components
  - Parallelization is possible at different granularities
  - Must consider the overhead of multi-threaded applications

Content Matching Algorithms

- Essential for any signature-based IDS
  - Algorithms were not necessarily motivated by IDS
  - *It is just string searching*
- Snort has incorporated various searching algorithms over time
  - Initially a simple brute force search, repeat for each signature
  - Replaced by Boyer-Moore, but still sequential
  - Snort 2.0 added Aho-Corasick and Wu-Manber (multi-pattern)
  - Snort 2.x added refinements to the existing algorithms
- Multi-pattern search has significantly increased performance

Boyer-Moore Overview

- Used to quickly find a single pattern in a text
  - Compare to last character of pattern and shift tables
- Assume pattern is length m and t is the text
  - Compare the last character of pattern to t_m
  - If not a match and t_m not in the pattern, look at t_{2m}
  - If t_m matches 4th character in pattern then look at t_{m+4}

a	n		e	x	a	m	p	l	e
a	m	p							
			a	m	p				
					a	m	p		

- *"Longer the pattern the faster the search"*, possibly sublinear
- Unfortunately, IDS needs to search for thousands of patterns

Wu-Manber Overview

- Used to quickly find a group of patterns in text
  - Use shift table from Boyer-Moore, but with multi-patterns
  - Creates hash tables for pattern look-up
- Assume smallest pattern is length m and t is the text
  1. Call shift table on t_m, which return s
  2. If $s \neq 0$ then shift and go to step 1
  3. If $s = 0$ then (potential match) call hash
  4. If entries in hash table then sequentially match pattern(s)
- Best average case performance
  - *"Short patterns inherently makes this approach less efficient"*
  - Maximum shift m is the shortest pattern in the group

Aho-Corasick Overview

- Linear time algorithm for multiple patterns
  - Based on an automata approach
  - Builds a FSM based on the characters in the patterns
  - Informally, consider building a search tree (trie)
- Best worst case performance (linear)
  - Requires more memory than other algorithms
  - Wu-Manber has a better average case due to skips

What does Snort Use?

- Snort selects an algorithm based on the number of rules
  - If there are fewer than 5 rules, then sequential Boyer-Moore
- If more than 5 rules, then use a multi-pattern algorithm
  - *The default algorithm is ...*
- Change `/etc/snort.conf`

```
# Configure the detection engine
# ===============================
#
# Use a different pattern matcher in case you have a machine
# with very limited resources:
#
# lowmem ac mwm
config detection:  search-method ac
```

`if(pattern found) validate`

- Regardless of the algorithm, if pattern found then validate
  - Initial search uses Boyer-Moore, Wu-Manber, or Aho-Corasick
  - Search for the longest `content` string in the rule (*good idea*)
- Second phase attempts to validate the initial match
  - Snort rules may contain multiple keywords, including `content`

```
alert tcp any any -> any 80 (content:"abcde"; nocase; offset:5
depth:15; content:"fghi"; distance:3 within:9);
```

  - `mSearch` verifies the remainder of the rule

Rule Groups and Small Signatures

- Assume there is a group that contains $r > 5$ rules
  - Furthermore, assume smallest signature in the group is 1-byte
- Snort will use *Wu-Manber No Bad-Character Shift* algorithm
  - Builds two hash tables, one-byte and multi-byte
  - Helps *distribute* the hash entries
- When processing packet of length n
  - Will make n calls to the *one-byte* hash table
  - Will also make $n - 1$ calls to the *multi-byte* hash table
  - *As a result, small signatures are generally avoided*

Anomaly Detection

- The normal behavior of the system is modeled
  - Patterns that *deviate* from normal are attacks
  - Premise is *malicious activity is a subset of anomalous activity*
  - Applicable to network attacks, such as DoS
- Most systems use a form of **change point monitoring (CPM)**
  - *Determine if the observed data is statistically homogeneous, and if not, determine when the change happened*
  - Collect statistics about system under normal usage (history)
  - Once statistics change, then it is/was possible attack

$$r_n = \alpha \times r_{n-1} + (1 - \alpha) \times r_n$$

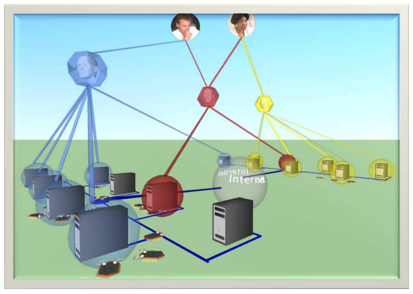
What to Measure

- **Number of unique IP addresses**
  - Large number of unique IP addresses indicate DDoS attack
- **Number of TCP SYN packets**
  - 90% of the DoS attacks use TCP, measure the number of SYN requests to a certain server
- **Compare the number of TCP SYN and SYN/ACK packets**
  - Distributed Reflector DoS (DRDoS), use routers as reflectors to send SYN/ACK packets
- **Compare the number of TCP SYN and FIN packets**
  - Should be the *relative* same number of SYN and FIN packets

Anomaly Detection Advantages/Disadvantages

- Possibility of very high false alarm rate
  - *What is normal? What is a significant change?*
  - Usage change over time, *can anomaly detection differentiate?*
  - Would consider a *flash crowd* and attack
- Does not depend on specific attack signatures
  - Attack variations can be detected and possibly novel attacks
- Not as resource intensive since measuring aggregates
  - May still have difficulty in high-speed environments
  - *Can one IDS see a DDoS?*

Swarm Intelligence (*PNNL Project*)



- Defense using swarm intelligence and simple software agents
  - Swarm of digital ants, each finds evidence per machine
  - Group of findings will indicate the actual problem
  - Movement based on pheromone, swarm an infected machine
- Better (faster and more robust) than having an IDS per machine?

Swarm Design

- Actually a hierarchy of agents, lower two levels...
  - **Sentinel** - resident per machine receives information per agent
  - **Sensors** - wander the network, there are several types of Sensors each looking for a certain type of evidence
- General operation is as follows
  - When a Sensor arrives to a computer, it performs a simple test
  - Test results given to Sentinel, determine if system is *healthy*
  - If results are helpful, then reward Sensor which attracts others

  What type of IDS is this? (Note, "failed" is not an answer)

  What are the advantages?

New Directions

- Combining multiple IDS types
  - Combine signature and anomaly at host and network
  - Detect new attacks with low false alarm rate
  - **Intrusion Detection Alert Correlation** considers multiple event streams from different IDS
- Specification-based intrusion detection
  - Describe attacks in more *general* terms (unlike snort)
- Attack prediction
  - Using on-line statistics *is it possible to predict an attack?*
- Integrated IDS and firewall system

Honeypot

- A system (set of computers) to *encourage* intrusion attempts
  - Watch attackers, learn new methods and exploits
  - Learn attacker identity and block
  - *Use as a diversionary tactic*
- For example Honeyd can mimic difference system types
  - Considered a *virtual honeypot*

    "Honeyd is a small daemon that runs both on UNIX-like and Windows platforms. It is used to create multiple virtual honeypots on a single machine. Entire networks can be simulated using honeyd. Honeyd can be configured to run a range of services like FTP, HTTP, or SMTP. Furthermore, a personality can be configured to simulate a certain operating system. Honeyd allows a single host to claim as many as 65536 IP addresses."

  - This includes platforms, OS, and applications

    Honeyd emulates operating systems by responding with appropriate packets to Nmap and Xprobe fingerprinting packets.

```
create sticky
set sticky personality "Linux 2.2.14"
set sticky default tcp action tarpit open
set sticky default udp action block
bind 192.168.1.110 sticky
```

- Honeynet places the honeypot across network addresses
  - Unused address space utilized by honeypot
  - Can be used to detect worm activity