IDS

CSC 790

WAKE FOREST
UNIVERSITY
Department of Computer Science

Fall 2014

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

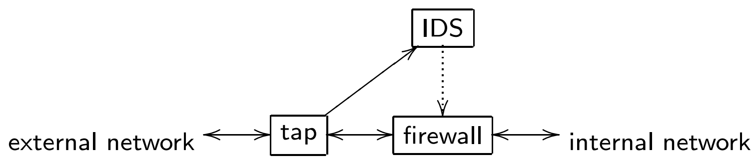
- Known attack patterns create a library of attack signatures
  - Data that matches a library entry is considered an attack
  - An example signature based NIDS is **snort**
- Snort supports header and payload inspection of network traffic
  - User can define a rule and action that is applied to packets
  - 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



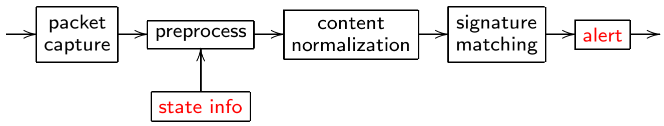
  - Can also use the firewall to *avoid* IDS for legitimate traffic

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);
```

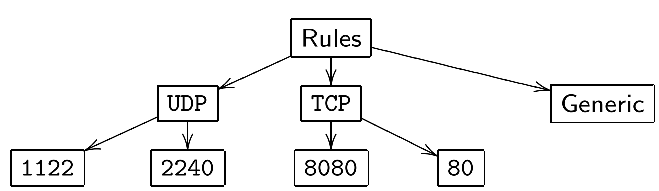
- Example above contains multiple `contents`
  - Additional keywords specify the `content`, case, and locations

```
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

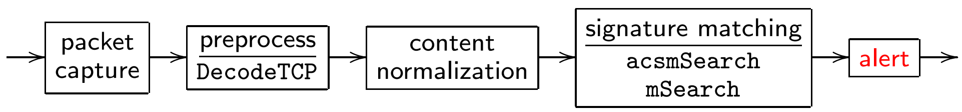


- When a packet arrives the content is compared against rules in
  - Port groups associated with the packet
  - Generic port group

Specifically, What is the Problem?

```
Terminal

> 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

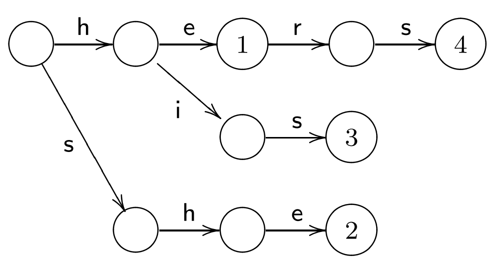
- Linear time algorithm for multiple patterns
  - Based on an automata approach
  - Builds a FSM based on the characters in the patterns
  - Refinement of a **keyword tree** (*trie*)
- Best worst case performance (linear)
  - Requires more memory than other algorithms
  - Wu-Manber has a better average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

Keyword Tree

- Keyword tree (or a trie) for a set of patterns P is a rooted tree K such that
  - Each edge of K is labeled by a character
  - Any two edges out of a node have different labels
- The label of a node v is the concatenation of edge labels on the path from the root to v, and denote it by $L(v)$
  - For each $p \in P$ there is a node v with $L(v) = p$
  - The label $L(v)$ of any leaf v equals some $p \in P$

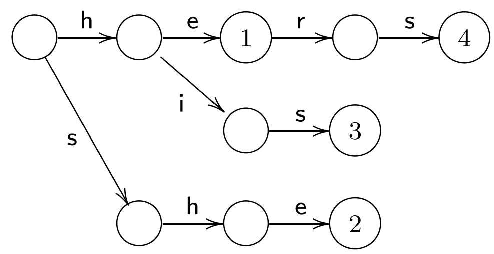
Example Keyword Tree Construction

- Keyword tree for $P = \{\text{he}, \text{she}, \text{his}, \text{hers}\}$



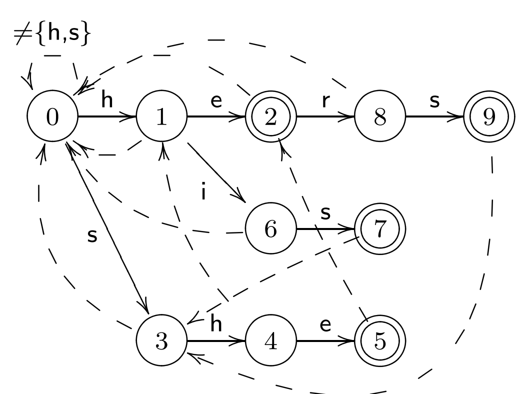
- Construction of the tree for $P = \{p_1, p_2, ..., p_k\}$, starting at the root, follow the path labeled by chars of p_i
  - If the path ends before p_i, continue it by adding new edges and nodes for the remaining characters of p_i
  - Store identifier i of p_i at the terminal node of the path
- Requires $O(|p_1| + |p_2| + ... + |p_k|) = O(n)$ time

Keyword Tree Lookup



- Lookup of a string s: start at root, follow the path labeled by characters of s as long as possible
  - If path leads to node with identifier, s is keyword in the dictionary
  - If the path terminates before s, the string is not in the dictionary
- Takes $O(|s|)$ time in the best case, *very efficient lookup method*
- *Worst case?* We can create an automaton to keep linear...

String Matching



- Automaton consists of normal and *failed* (dashed) transitions
  - Double circles indicate matched strings
  - Node labels are the longest proper suffix
- Consider processing the string "ushers"

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

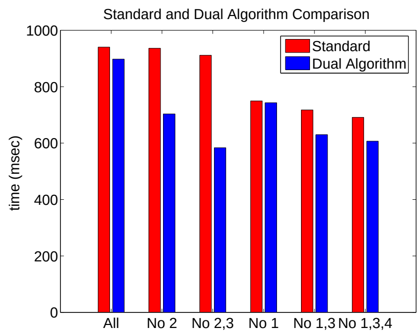
- Assume there is a group that contains $r > 5$ rules
  - Furthermore, assume smallest signature in the group is 1-byte
- Snort will use *Wu-Manber No Bad-Character Shift* algorithm
  - Builds two hash tables, one-byte and multi-byte
  - Helps *distribute* the hash entries
- When processing packet of 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*

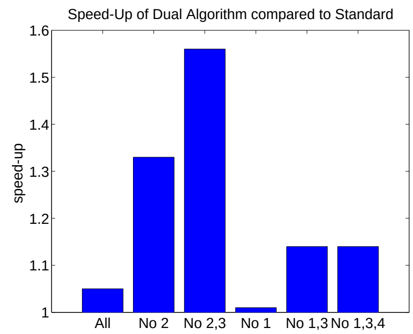
Proposed Dual Algorithm

- If a group has 1-byte and multi-byte patterns, then separate
  - Sub-group for 1-byte patterns and another for multi-byte
  - *Original thought was to process groups in parallel*
- Process the payload twice
  - For the 1-byte group, use B-M or W-Mnbcs
  - For the multi-byte group, use W-Mbcs
  - *Yes, the payload is processed twice*
- *As a result, we hope...*
  - Greatly reduce the number of multi-byte hash calls

Experimental Results

- Patrick has extensive results comparing dual to standard
  - 12 pages of results varying different parameters
- Used Snort 2.5.4 match component with traffic traces





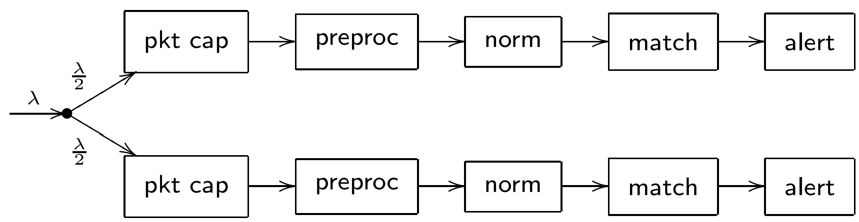
- As ASL increases, the dual algorithm should perform better
  - *Greater the gap, the better the performance*

Parallel IDS

- IDS in a high-speed environment is very difficult
  - Algorithm improvements may not be sufficient (*sorry Patrick*)
  - Faster hardware is only a temporary solution
  - Would like to parallelize IDS, parallel $\neq$ distributed
- Given an array of processing elements, you can
  - **Data parallel** - divide data across equal processing elements, reduces the arrival to any one element (improves throughput)
  - **Function parallel**- divide work across the processing elements, reduces the work at any one element (reduces latency)
- Reducing latency is the objective $\rightarrow$ function parallel
  - Difficult to implement... *where/what?*

Parallel Taxonomy

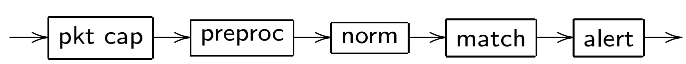
- Three levels where parallelization is applicable
  - System, component, and sub-component
- **System level** parallelization
  - Just duplicate the entire IDS and split the traffic... easy



  - Scalable and robust, but difficult to maintain state info
  - This is data parallel, *what is the function parallel version?*

Component Level

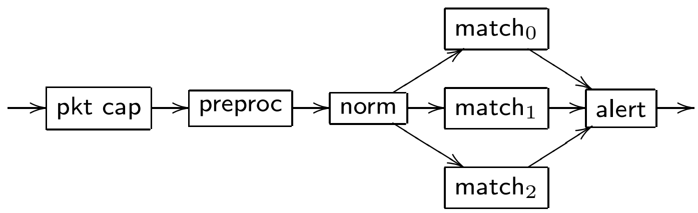
- Parallelize the different **components** of one IDS



  - Isolate components, more like *pipelining*
- Preprocessing, matching, and alert are good possibilities
  - Processing can be divided into reassembly and port-scan
- Must consider the support and design required for threads
  - Data structures, semaphores, and context switching
  - *Will the threads be serialized?*

Sub-Component Level

- Parallelize (duplicate) **components** of one IDS



  - For example, parallelize matching component (original idea)
  - Duplicate all groups or different rule groups per thread
- Does not work with all components, *consider reassembly*
  - Same problem as system level, must maintain state information
- *Are these examples of data or function parallel?*

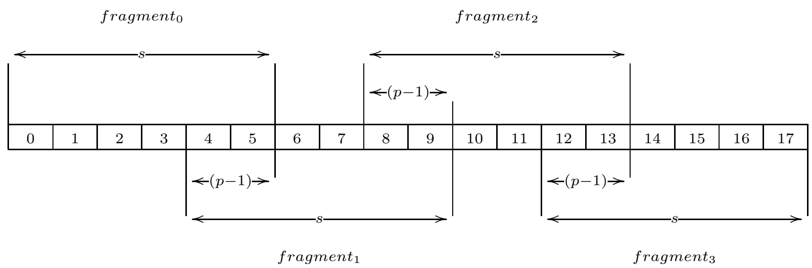
Data Parallel String Matching

- Standard data parallel approach
  - Distribute packets across processing elements
  - Reduces the arrival to any one processing element
  - Requires load balancing
- An alternative is Divided Data Parallel (DDP)
  - Divide data of one packet across processing elements
  - Each processing element has a smaller part (fragment)
  - Shorter inputs are better, *right?*

    What are the problems associated with dividing the payload?

Dividing the Payload

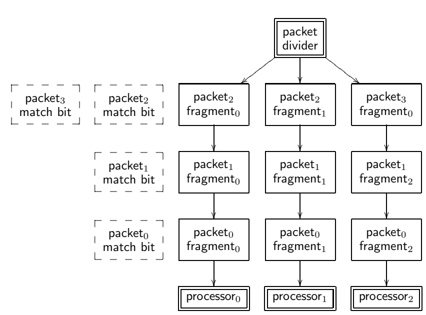
- DDP divides the packet payload into fragments
  - Fragments given to processing elements, complete payload scanned
- Signature (malicious content) may span fragment
  - Single processor may not see complete signature
  - Must overlap fragments to prevent false negatives



- Overlap dependent on largest signature, p, (example above $p = 3$)
  - Overlap is $(p - 1)$ with leftmost fragment

Match This

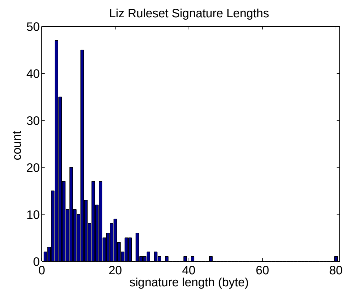
- Only considering the initial match
  - Once a match made, can skip remaining packet
  - Original DDP does not



- Once a match is made, set match bit for the packet
  - Match bit indicates do not process remaining fragments
- Allows processors to skip fragments

DDP Performance

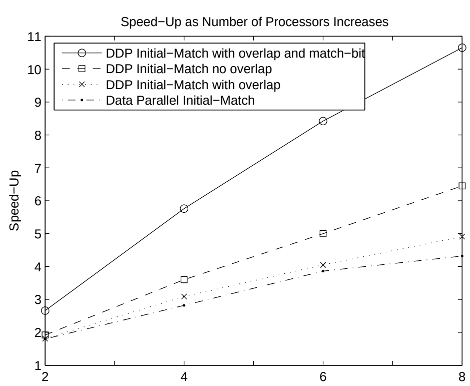
- Compare different parallel match performance
  - Data parallel and forms of distributed data parallel
- Created a multi-threaded match for Snort 2.6
  - Traffic traces from a university
  - Rules from government agency (345 total web-rules)

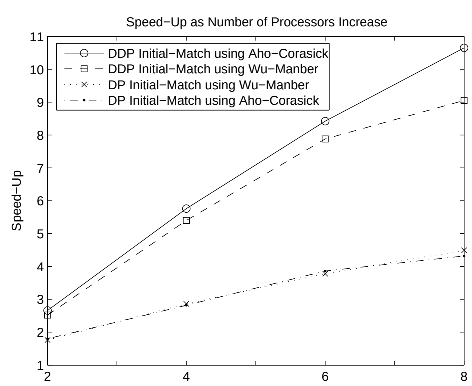


- Recorded processing time and speed-up

Some Results

- DDP always performed better than traditional data parallel





- Speedup using match bit was the best

Types of Parallelism

	Data Parallel	Function Parallel
System	replicate IDS	?
Component	?	isolate IDS components
Sub-Component	duplicate match	distribute rules across match

- Of course multiple levels/types of parallelization are possible
- Must consider overhead associated with parallelization
  - Queues for moving data from one thread to another
  - Inter-thread communication and context switching
  - Semaphores and potential serialization

Problems with Signatures

- IDS discussed thus far relies on signature matching
  - Look for suspicious packet headers and/or payloads
- Great for finding known threats
  - Not so great for zero-day threats...

Detection Using Machine Learning

- It may be possible to use machine learning to find malware
  - Assumption is malware looks different than non-malware
- Supervised learning could be used
  - Train IDS to identify normal/legit conditions
  - Should be able to identify never-before-seen conditions

  Of course the problem is?

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?*

IDS Performance Metrics

- **True Positives** (TP) is number of correct malware classifications
  - **True Positive Rate** (TPR) is $\frac{TP}{TP+FN}$
- **True Negatives** (TN) is number of correct non-malware classifications
- **False Positives** (FP) is number of incorrect classifications of non-malware as malware
  - **False Positive Rate** (FPR), or false alarm rate, $\frac{FP}{FP+TP}$
- **False Negatives** (FN) is number of incorrect classifications of malware as non-malware

A Few More Performance Metrics

- **Recall** is the fraction of correct instances among all instances that actually are positive (malware)
  - Calculated as $\frac{TP}{TP+FN}$

    How can you get perfect recall?
- **Precision** is the fraction of correct instances (malware) that algorithm believes are positive (malware)
  - Calculated as $\frac{TP}{TP+FP}$

    Is perfect precision always best?

Base Rate Fallacy

- Base rate fallacy (base rate bias), is an error in thinking
  - When given related general, generic information (base rate info) and specific information, we tend to focus on the specific
- Suppose you develop a vampire test that falsely indicates a vampire in 5% of the cases; however, it never fails to detect a real vampire
  - Assume in a town, 1/1000 of people are vampires
  - Suppose you stop a random person and the test indicates vampire
    What is the probability the person is a vampire?

Bayes' Theorem

- Need to use Bayes' theorem, which can find the probability of B given A is also true
  - For us, we want the probability of the person is a vampire given the test indicated vampire
- The equation for our problem is

$$p(vampire|test_{vampire}) = \frac{p(test_{vampire}|vampire)p(vampire)}{p(test_{vampire})}$$

  where
  - $p(vampire) = 0.001$
  - $p(normal) = 0.999$
  - $p(test_{vampire}|vampire) = 1$
  - $p(test_{vampire}|normal) = 0.05$

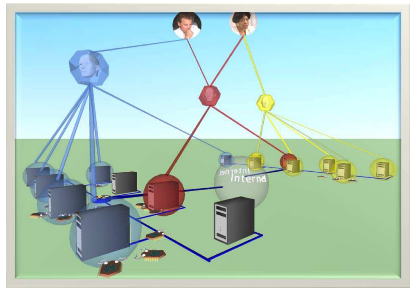
– $p(test_{vampire}) = p(test_{vampire}|vampire)p(vampire) + p(test_{vampire}|normal)p(normal) = 0.05095$

- Therefore $p(vampire|test_{vampire}) = 0.019627$
- A more intuitive explanation, on average for every 1000 people tested
  – 1 person is a vampire, and 100% certain that for that person there is a true positive test result, so there is 1 true positive test result
  – 999 people are not vampires, and among those people there are 5% false positives, so there are 49.95 false positive results
  – Therefore therefore the probability that one of the people among the 1 + 49.95 = 50.95 positive test results really is a vampire is $\frac{1}{50.95} \approx 0.019627$

Problem with IDS

- Very difficult to have good recall and precision
- Malware typically is a small percentage of traffic
  – Looking for one packet in a million (or billion)?
- Insufficiently sensitive, IDS will miss the malware (low recall)
- Too sensitive, IDS will have too many alerts (low precision)

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?*
- Better integrated IDS and firewall system