SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset
    - Tuning the Snort ruleset has the greatest impact on Snort's performance and the number of false positives.
    - If we can apply the knowledge of our network infrastructure and IDS policy to the ruleset, we can achieve a high performance of the IDS operation.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Snort rules are made up of two components:
      - the **Rule Header** and
      - the **Rule Option.**
    - The Rule Header defines the type of alert and which protocols, IP addresses and IP protocol ports are to be monitored for the signature.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - The Rule Header can be described as metadata that lets Snort know where to apply the signature.
    - The Rule Header is essentially everything that comes before the first parentheses.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - The Rule Option is the actual signature and assigned priority of the attack.
    - The Rule Option also contains links to external documentation resources on the Internet.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Example:
      - The Rule Header:
        - » alert tcp $EXTERNAL_NET any->$HOME_NET 22
      - The Rule Option:
        - » (msg:"EXPLOIT ssh CRC32 overflow /bin/sh"; flow: to-server, established; content: "/bin/sh";)

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - The Rule Headers and Rule Options are mapped into an internal data structure when the ruleset is loaded into memory.
    - The Rule Header is mapped to an internal data structure within Snort known as a *Rule Tree Node* (RTN).
    - The RTNs are linked together into one dimension on a three-dimensional linked list.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Each protocol (TCP, UDP and so on) has its own linked list made up of the corresponding RTNs.
    - The second dimension is mapped from the Rule Option in the form of an *Option Tree Node* (OTN).

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - The third dimension is a group of function pointers that determine which options should be applied to a packet to be inspected.
    - This linked list of RTNs, OTNs, and function pointers is essentially the data structure that the detection engine uses.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - When the detection engine processes a packet, it first checks to determine what protocol the packet uses.
    - After the protocol is determined, the packet is sent to the corresponding linked list.
    - The packet is then checked against each RTN until a match is found.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - After a match is found, the packet is passed by the OTNs.
    - OTNs that utilize Boolean or mathematical operators are executed in a short time with little overhead.
    - OTNs that are composed of only these types of tests are not computationally expensive and execute quickly.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Example:
      - The IP reserved bit rule:
        - » alert ip $EXTERNAL_NET any -> $HOME_NET any (fragbits: R;)
      - This rule checks to see only whether a packet has the Reserved Bit set, which would indicate suspicious traffic.
      - This rule does not check any contents, so its execution is fast.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - OTNs that utilize any of the content checking options (*uricontent*, *content-list*, *content*) are much more computationally expensive and require more resources than OTNs that do not.
    - Content options are expensive because they force Snort to make use of the pattern matching engine, which is resource intensive.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - When a packet matches an OTN, an alert is generated and passed to the output stage.
    - If the packet does not match an OTN, it is flushed from memory.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Thus, if it is necessary to reduce the load on Snort, the rules that utilize content checks should be examined.
    - Those rules that are unnecessary should be removed.
    - Unfortunately, about 70 % of the Snort rules make use of one of the three content options (uricontent, content-list, content).

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - If Snort is dropping packets, the rules should be prioritized into categories to identify content rules that are not critical for the protected environment.
    - One should begin by removing rules that alert to non-malicious behavior, such as inappropriate Internet activity.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Sometimes it is also necessary to inspect the *.rules* files for content rules that alert to less serious activity.
    - These rules should be disabled only if it is absolutely necessary (for example, to prevent packet loss).

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - The rules should be organized in a logical, efficient manner.
    - The goal of organizing is to have rules that utilize content options execute last.
    - The packets should be inspected against rules that are not resource intensive first, with the hope that they will trigger on an OTN before reaching the computationally expensive content options.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - When Snort is tuned properly for the network and is no longer dropping packets, the next activity is to reduce false positives.
    - It is possible to reduce some false positives by configuring network variables and preprocessors.
    - A popular way to remove false positives is to create so called pass rule.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - A pass rule is one of the possible rule categories, such as alert and log.
    - It is the inverse of an alert rule; a pass rule tells Snort to ignore any packets that match the pass rule.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - It is possible to use a pass rule to ignore certain types of traffic from specific hosts.
    - For example, it is possible to ignore SSH traffic sent from a single server with a pass rule.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Example:
      - If we are getting a number of false positives from a single host, we could write a pass rule to ignore all traffic from that host.
      - The following pass rule tells Snort to ignore all TCP traffic from a host located at 192.168.1.1:
        - » pass tcp 192.168.1.1 any -> any any;

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - It is possible to get more granular if necessary.
    - Example:
      - If we want to ignore traffic from the same host destined for Telnet servers:
        - » pass tcp 192.168.1.1 any -> any 23;
    - It is also possible to append content options to pass rules.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - If the traffic matches the rule header and the content option, it will be ignored.
    - If the same host were to constantly issue false positives relating to unauthorized Telnet login attempts, we could add the following content rule:
      - pass tcp any 23 -> 192.168.1.1 any (content: "Login failed"; nocase; flow: from-server, established;)

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Pass rules can be used in many different situations to eliminate repetitive false positive offenders.
    - However, the order in which Snort processes rules must be changed.
    - By default, Snort processes alert rules, then pass rules, and finally log rules.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - If we create a pass rule that matches an alert rule, packets that match both the alert and pass rule will still be logged as an alert to the output plugin.
    - This processing order holds to avoid false negatives. It protects the system from accidentally creating a bad pass rule that would inadvertently cause Snort to ignore traffic that it should not.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Example:
      - Suppose that in the previous pass rule example, we had forgotten to specify the IP address of the host we wanted to ignore.
      - Then we could accidentally ignore all TCP traffic.
      - This rule would do exactly that:
        - » pass tcp any any-> any any;

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - The order in which Snort processes alerts is changed in such a way that pass rules are processed first.
    - In that case the alert order would be pass, alert, log.
    - We can do so by running Snort with the –o command line option.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - It is possible to develop a *targeted ruleset* that will alert only on services and hosts that actually exist in the protected network.
    - This can reduce the ruleset's size considerably.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - With a targeted ruleset, it is less likely to discover attempted attacks.
    - The attacker would have to attempt to attack a legitimate service on a legitimate host to be noticed by Snort.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - The targeted ruleset is also useful in cutting down false positives.
    - If we are monitoring a network that generates a lot of false positives, a targeted ruleset will greatly reduce the amount of false positives we receive.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - For small networks, the targeted ruleset can be generated manually, by first performing a portscanning and then disabling the rules targeted at inactive ports.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - For large networks for which we would like to build a targeted ruleset, we can make use of a tool, *snortrules*.
    - Snortrules takes the output from an NMAP scan and edits a Snort rules file.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - It takes action on rules that do not match a particular service.
    - Snortrules can either remove rules or flag them as not applicable.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Limitations with the targeted ruleset implementation:
      - Network configurations are rarely static for any lengthy period.
      - If we compile the list of available services one day, the network could change on the next day, making the targeted list out of date.
      - If we decide to use the targeted ruleset method for the sensor, we should adopt a regular schedule to update the list as appropriate.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Limitations with the targeted ruleset implementation (cont.):
      - Another, more dangerous possibility is that an attacker would manage to utilize a port or host we are not monitoring in some phase of an attack.
      - This is possible if the port was not open at the time of scanning, but was subsequently opened by the attacker.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Limitations with the targeted ruleset implementation (cont.):
      - If the attacker were able to install a Trojan, either by tempting an unsuspecting person to open an e-mail attachment, or by sitting at the console and installing it, we would not be able to detect the intrusion.
      - The attacker would be able to carry out any sort of remote control tasks on the compromised host without our knowledge.

SNORT

- Fine tuning SNORT after the installation (cont.)
  - Refining the ruleset (cont.)
    - Some other elements of the SNORT system could also be tuned for a better performance:
      - The database (MySQL, etc.)
      - ACID
      - Caching system
      - Etc.

SNORT

- SNORT custom rules
  - The goal in creating effective signatures is to write rules that match exclusively the network traffic we want to discover.
  - Unfortunately, this goal is almost impossible to attain; each rule is likely to trigger on other traffic too.

SNORT

- SNORT custom rules (cont.)
  - When writing a rule, one should make a best effort to narrow down the rule to trigger on only the isolated traffic patterns of which one wants to be alerted.
  - One should also take care not to add too many traffic properties, which would cause some attacks to not match the rule.

SNORT

- SNORT custom rules (cont.)
  - To write rules that will trigger only on the traffic we intend them to, we must research and discover properties of the traffic that are unique.
  - The individual properties of the traffic need not be unique themselves, but the combination of them should be.

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting
    - Cross-site scripting (XSS) occurs when a Web site allows malicious script to be inserted into a dynamically created Web page.
    - If user input is not properly checked, the attacker can embed script that will force the Web application to act in an unintended manner.

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - XSS attacks can be used to steal cookies used for authentication, access portions of the Web site that are restricted, and otherwise attack Web applications.

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - The majority of XSS attacks require scripting tags inserted into a particular page request.
    - We can use this feature of XSS attacks to write a rule.
    - Tags such as <SCRIPT>, <OBJECT>, <APPLET>, and <EMBED> are required to insert an XSS script into a Web application.

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - We can now create a rule that should trigger when the <SCRIPT> tag is discovered.
    - First we create a rule to trigger on traffic with "<SCRIPT>" content:
      - alert tcp any any -> any any (content: "<SCRIPT>'; msg: “WEB-MISC XSS attempt”;)

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - This rule triggers on XSS attacks, but unfortunately also triggers on many other types of benign traffic.
    - If someone were to send an email with embedded JavaScript, the alert would be triggered, causing a false positive.

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - To prevent this type of false alarms from happening, we need to change the rule to trigger only on Web trafic:
      - alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (content: “<SCRIPT>” msg: “WEB-MISC XSS attempt”;)

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - This rule triggers only when the <SCRIPT> content is detected in relation to an HTTP session from a Web server.
    - It triggers when the traffic originates at an external IP address ($EXTERNAL_NET), and is sent to our Web servers ($HTTP_SERVERS) on the ports on which an HTTP service runs ($HTTP_PORTS).

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - However, after loading this rule, a large number of false positives are generated whenever a page is requested that contains JavaScript.
    - We need to further refine the rule and discover properties of XSS traffic that are unique.

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - XSS occurs when the client embeds the <SCRIPT> tag in a request.
    - If the server sends the <SCRIPT> tag in response to a request, it is probably benign traffic (JavaScript).

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - We can use this property of an XSS attack to further refine the rule:
      - alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg: “WEB-MISC XSS attempt”; flow: to_server, established; content: “<SCRIPT>”;)
    - This revised rule makes use of the flow option, which uses Snort's TCP reassembly features to identify the direction of traffic flow.

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - The flow options specified, to_server and established, apply the rule only to sessions that originate at the client and are sent to the server.
    - This is where an XSS attack will occur: Traffic flowing in the opposite direction is likely to be a normal HTTP session containing JavaScript tags.

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - We also need to make sure an attacker cannot evade the rule by taking advantage of case sensitivity.
    - The content option is case-sensitive, whereas HTML is not, so an attacker could evade this rule by changing the script tag to be <ScRipt> or <script>.

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - To remedy this, we make the content option not case-sensitive:
      - alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg: “WEB-MISC XSS attempt” ; flow: to_server, established; content: “<SCRIPT>”; nocase;)

SNORT

- SNORT custom rules (cont.)
  - Example - cross-site scripting (cont.)
    - Finally, we assign the rule a high priority:
      - alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msq: “WEB-MISC XSS attempt”; flow: to_server, established; content: “<SCRIPT>”; nocase; priority: 1;)

SNORT

- SNORT custom rules (cont.)
  - Snort rules have a basic syntax that must be adhered to for the rule to properly match a traffic signature.
  - Violating the Snort rules syntax can cause a rule to not load into the detection engine.
  - Even if such a rule does manage to load, incorrect rule syntax may result in unpredictable and unintended consequences.

SNORT

- SNORT custom rules (cont.)
  - The rule could trigger on a large amount of benign traffic, causing a lot of false positives.
  - This could potentially overload the intrusion database.
  - The rule could trigger on randomly occurring traffic patterns, which have the potential to cause unnecessary panic when an alert is generated.

SNORT

- SNORT custom rules (cont.)
  - Some rules load, but never trigger on the traffic they are designed to detect.
  - The IDS operator may assume the rule is functioning correctly and miss out on the alert.
  - The same scenario can occur in the case of a pass rule, where a poorly written rule can cause a significant amount of potentially malicious traffic to be ignored.

SNORT

- SNORT custom rules (cont.)
  - It is therefore important to make sure the custom rules are written in the correct syntax.
  - It is a good practice to check rules over and test them before implementing the rules in a production situation.

SNORT

- SNORT custom rules (cont.)
  - The most basic syntactical requirement of a Snort rule is that it be in a single line.
  - If we must separate the rule into more than one line, we must append a backslash to the end of the line to let Snort know to continue on the next line.

SNORT

- SNORT custom rules (cont.)
  - The syntax of the rule header is:
    - Rule_action protocol source_address_range source_port_range direction_operator destination_address_range destination_port_range
  - The rule action, protocol, and direction operator are normally chosen from a static list of possible values.

SNORT

- SNORT custom rules (cont.)
  - Snort dictates these statically because the rule can trigger only a limited number of possible actions, and Snort can monitor for only a limited number of protocols.
  - The remaining parameters can be assigned to a variable (such as $HOME_NET), an IP address or port, or a range of IP addresses and ports.

SNORT

- SNORT custom rules (cont.)
  - The rule option is the actual signature and the assigned priority.
  - The signature portion of the rule option is represented with one or more option keywords.

SNORT

- SNORT custom rules (cont.)
  - These option keywords are used to build the traffic signature for which one would like the detection engine to monitor.
  - When more than one option keyword that relates to a signature is used, they can be considered to form a logical AND statement.

SNORT

- SNORT custom rules (cont.)
  - There are essentially three methods to write Snort rules:
    - To modify or add to an existing rule, in order to tune Snort and make it more efficient - easiest.
    - To create a new rule by using the knowledge of our network - relatively easy because no extensive traffic analysis is required.
    - To create a new rule by examining network traffic - the most difficult.