Network Intrusion Detection Systems

Created by: Peter A. H. Peterson and Dr. Peter Reiher, UCLA.
{pahp, reiher}@cs.ucla.edu

Put your due date here if you are including the exercise in your class.

Contents

Overview

The purpose of this lab is to introduce you to Network Intrusion Detection Systems (NIDS) and give you an opportunity to use them to investigate threats and attacks, and to diagnose network health. In order to complete these labs, you need to have a working knowledge of TCP/IP, the client-server model, and the Unix command line. Experience with tcpdump and iptables is a plus.

After successfully completing this lab, you should be able to:

1. understand and describe the generic components, structure and use of NIDS
2. use NIDS to identify network traffic patterns, including:
    1. healthy traffic
    2. unwanted (but permitted) traffic
    3. malicious traffic (such as DDos and worm traffic)
    4. misconfigured network equipment
    5. other patterns not listed here
3. identify specific kinds of traffic based on packet signatures
4. recognize false positives, including:
    1. erroneous rules
    2. naive matches
    3. paranoid matches
    4. ambiguous rules
5. compose and/or change rules to
    1. target traffic patterns currently being missed

    2. ignore traffic patterns currently being flagged
    3. modify existing rules to perform differently
6. examine traffic traces and analyze them

Required Reading

Network Intrusion Detection System

A Network Intrusion Detection System (NIDS) is a specialized form of an Intrusion Detection System (IDS), that is used to detect threats, generate alerts, and sometimes respond to network-based threats (although system response typically falls into the category of Intrusion Prevention Systems). Threats typically appear in the form of attacks, but NIDS are also very useful for network traffic and health analysis because they can be configured to raise alerts on any traffic with any characteristic that can be recognized by the NIDS software. As in antivirus systems, expressions of the "identifying characteristics" are called "signatures". NIDS typically log threats recognized by the signautres to disk, and commonly have a GUI or other application for analysis and inspection.

IDS are commonly built of three kinds of abstract components -- *sensors*, *engines*, and *consoles*. Sensors generate events, engines analyze those events and log them, while consoles allow us to interface with the stored data and in some cases control the sensors or engines. You will work with the popular open-source combination of Snort and the Basic Analysis and Security Engine (BASE).

Snort is a console application that utilizes tcpdump's libpcap and it's own engine and plugin system to efficiently capture and analyze traffic on a network interface, generating alerts and storing them in a file or database such as MySQL. Snort can perform simple inspection of packets for threat signatures (i.e., looking for a sequence of bytes unique to a particular kind of attack) or more complex methods involving history or statistical analysis.

BASE is a PHP frontend to Snort's alert database (stored in MySQL) that allows administrators to examine all alerts down to the packet level, sort and search for trends, and investigate unknown events. In our lab system, Snort serves as our sensor and our engine; BASE serves as our console.

Rules

No single set of rules is sufficient for all networks. In fact, tuning NIDS rules to suit your network can be a lengthy and painstaking process because every network is different and there is no formula for creating good rules.

Sometimes, simple rules are exactly what is requred. For example, if a network has no public telnet interface, any attempts to log in on port 23 indicate a potential compromise attempt. Likewise, if a network has only one DHCP server with a fixed address, DHCP OFFERs from any other address are indicative of a problem and should be flagged. Both of these rules can be expressed in terms of simple network filters.

Unfortunately, sometimes simple rules aren't enough -- and even worse, simplistic rules can actually be a liability. Consider a CGI script called "wrapper" that is known to have serious remote vulnerabilities -- unfortunately, if the rule language is not expressive enough or the rule is too broad (e.g. searches for the string "wrapper" in a web request), your NIDS might not be able to distinguish between a request for wrapper.cgi or one for wrapper.html -- both have "wrapper" in the request. Additionally, a NIDS cannot typically evaluate software patch levels. While some versions of `wrapper` are vulnerable, most have probably been patched and so are no longer vulnerable. But if the NIDS can't tell which is which, it will often flag every attempt just in case. Writing good rules to accurately match these scenarios often requires in-depth knowledge of the protocol and vulnerability, in addition to significant processing resources dedicated to inspecting and analyzing packets.

Conversely, a NIDS evaluating more complex or sophisticated rules will often generate false positives because some kind of valid event on your system matches a signature authored by a developer on a different system. These alerts are frightening and confusing because they might indicate something that seems impossible but could indicate complete compromise (like MySQL traffic on a server without a MySQL daemon) or worse, something that seems plausible (compromised SQL traffic when you *do* have a MySQL server).

Finally, sometimes an alert that is *extremely suspect* on one network is *totally unremarkable* on another. For example, some wireless internet services perform ARP spoofing in order to redirect your traffic to their paywall. Or, for the network with a telnet daemon, login attempts on port 23 aren't necessarily suspicious.

In order to properly configure a NIDS, an administrator *must* carefully consider and **understand** why an alert has been raised before making design choices, especially choices to disable rules. This often results in some detective work focused on trying to determine if an alert is a false positive, meaningless, or is

truly significant. As a result, any NIDS operator will typically spend considerable time watching network results and carefully considering and implementing edits to the rules in order to improve the signal to noise ratio. This usually takes the form of judiciously disabling alerts, editing others, and creating some new ones. While you don't want any false positives, you also don't want to miss anything important.

We'll go into more specific detail regarding Snort and BASE below.

Snort

Snort is an Intrusion Detection System that uses `tcpdump`'s libpcap to capture packets, after which they are efficiently processed by a number of modular preprocessors, processors, filters, and rules. Snort is GPLed Free Software, but its maker, Sourcefire also sells proprietary versions, hardware, and support.

Snort can do basic stateless packet analysis, but also has more advanced features like protocol dissection, stateful inspection, and packet reassembly. This means that Snort can understand traffic flows and even put together fragmented TCP/IP packets. (Earlier versions could not do this, and as you might expect, this represented a security vulnerability.) Additional plugins (like advanced protocol dissectors) can extend the power of Snort's rule engine. This also allows administrators to streamline Snort's hardware requirements by eliminating unnecessary rules and processors.

Snort Rules

Similar to iptables, Snort rules can specify different actions for matching rules. In `iptables` the default actions are DROP, REJECT, FORWARD, etc., but in Snort the standard actions are *alert*, *pass*, or *log*. Our installation is configured to log alerts and information to a MySQL database, although flat files and other databases are supported.

Alert rules create events in the alert database. *Log* rules just specify that the packet is to be logged. *Pass* rules specify that a packet is to be ignored. Snort rules are processed in lexical order, this means that rules are processessd in the order they are added to the system (just like `iptables` rules). This is handy, because it allows *pass* rules and customized rules to be put in one local configuration file that is processed first in order to give local customizations first priority. This file is normally named something like `000-local` so that it will be first in an alphabetical list of the separate rules files. Putting them in a custom file

(rather than modifying rule files distributed with `snort`) also keeps them out of the upgrade path; this means that an administrator can download daily updates to the Snort rules, while knowing that their exceptions and customizations will remain in effect.

Rule Format

The Snort rule format is very simple and borrows much from the libpcap format used in `tcpdump`. For example, Snort rules are able (like `iptables`) to differentiate between established network connections and raw packets.

```
alert tcp $HOME_NET any <> $EXTERNAL_NET 6666:7000
(msg:"CHAT IRC message"; flow:established; content:"PRIVMSG
"; nocase; classtype:policy-violation; sid:1463; rev:6;)
```

For example, the above rule raises an alert any time when, on an established inbound or outbound connection on a TCP port between 6666 and 7000 includes the case-insensitive content "PRIVMSG". This is an IRC chat command used for private communication. Snort calls this alert "CHAT IRC message". It references sid (Snort ID) #1463. Snort classifies it as a "policy-violation" because it falls into the class of traffic that is not inherently a security hazard, but may be prohibited by work or school policy.

Snort ID numbers can be used to look up more detailed and up to date explanations of the event on the Snort website. Furthermore, additional rule id numbers can reference events and documentation from other sources, such as Bugtraq, CVE, Nessus, and others. BASE uses these references to point investigators to information online.

Different modules use additional special rule elements, for example, this web-specific rule matches the *content* of the URI, not the *payload*:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-ATTACKS /bin/ps command attempt";
flow:to_server,established; uricontent:"/bin/ps"; nocase;
classtype:web-application-attack; sid:1328; rev:6;)
```

The above rule makes use of variables which are defined in `/etc/snort/snort.conf` and matches network traffic from the $EXTERNAL_NET network to internal $HTTP_SERVERS and $HTTP_PORTS. It matches any URL that contains the string "/bin/ps" (which is probably an exploit attempt). The alert raised by the rule is called "WEB-ATTACKS /bin/ps command attempt", sid #1328, and is a member of the "web-application-attack" class.

For further information, see the online documentation, or look at a collection of Snort rules yourself via DETER. (Snort rules are located in `/etc/snort/rules` by default.)

BASE

BASE is the Basic Analysis and Security Engine, which is an active project built on top of the older and inactive ACIDlab (Analysis Console for Intrusion Detection) project. BASE is a web-based PHP frontend that interacts with the MySQL database that Snort creates. BASE is released under the GPL.

To access BASE and the Snort database, swap in your experiment, set up an ssh tunnel and navigate to `http://localhost:PORT/base/`.

You might want to do that now, so you can try out BASE as you read about it.

Using BASE

BASE is a "drill-down", query-creating, graphical frontend to the data that Snort stores in SQL. It only presents and organizes alerts; it does not do any alert creation or analysis per se. Instead, BASE constructs SQL queries from the filtering criteria the user selects. As such, it is important to understand that clicking on links in BASE typically narrows the selection of alerts. This process increases the specificity of the SQL queries defining the current view; the more specific the query, the fewer alerts are selected. While at first this functionality may seem limited, Snort itself generates such sheer volume of information that a user-friendly interface is a necessity.

BASE uses four criteria to build queries: Meta Critera, IP criteria, TCP criteria, and Payload criteria, with queries built by navigating the interface.

- **Meta Criteria** include things like time, uniqueness, and alert name.
- **IP** and **TCP** criteria specify network ports and addresses.
- **Payload** criteria match packet payload data.

Once the drill-down query process has selected an individual packet (an easy way is to select an individual ID# from a list of alerts), BASE provides an simple HTML-based packet level view that gives the user a direct look into the structure of the TCP/IP header and payload in a typical "hex and ASCII" display, similar to `tcpdump` with the -X option, or hexedit.

Filtering

The BASE main page is a summary of the current database with a collection of links to basic queries and statistics. For example, it shows links to "Today's alerts", "Last 24 hours alerts", "Most Frequent Source Ports", etc. Each of those links represents a specific query on meta, address, or payload criteria.

Further down on the page, a simple bar graph shows the distribution of alerts by protocol. Each protocol can be clicked on to see a query selecting all alerts of that protocol. (This section also includes "portscan traffic".)

Finally, a statistics area shows the total number of alerts, unique alerts, categories (classtypes), ports, etc. Each of the relevant categories can be clicked on to select packets matching that category; for example, when the "categories" link is clicked on in the statistics area, one is presented with a list of categories of alerts, e.g. *policy-violation*, *unclassified*, *attempted-recon*, etc. As above, these are all ultimately drill-down style queries using the above four criteria.

Examples

Here are some concrete examples of BASE's "drill down" interface in the form of a question and a method for finding the answer, starting from the BASE "home page":

What is the single most common alert generated by Snort for this alert database?

- Click on the "unique alerts" in the statistics area of the main page.
- Click on the "Total #" column to sort by number of alerts.
- The top result represents the most common alert.

What remote host created the most number of alerts in the last 24 hours?

- Click on the "Last 24 Hours alerts" by "Source IP" link

- Sort by "Total #".
- The top result represents the most common alert.

If the administrator wanted to see the address that created the most *unique* alerts in the last 24 hours, they could click on "Unique Alerts" instead of "Total #".

NOTE: BASE provides a "Back" button in its interface; this is used to remove the last query change in a string of queries. This has the effect of "backing up" one level. Clicking the "back button" in your browser may not have the same desired effect.

Cross-referencing

Often, when a security administrator is trying to determine if an attack is more than just a simple scripted or random attack, it is useful to look at the database from "multiple angles" in order to have a better idea of what is happening.

For example, if a certain alert is being raised many times (such as a DoS alert), are all the alerts coming from the same host, or multiple hosts? Sometimes reconnaisance attempts are a prelude to a full-blown attack. If a particularly worrisome alert is raised by a particular source IP address, have other alerts been raised by that address in the past? Similarly, sometimes alerts are raised due to the actions of regular, authenticated users and aren't security violations at all. Discerning between these false alarms and real attacks from unauthenticated users requires deeper investigation and knowledge that Snort does not have unless it is encoded into the rules.

Deleting Alerts

Once a query has been selected, the "action" menu at the bottom of the page can be used to perform multiple actions. Commonly, when an administrator is convinced that a particular query is not a threat, he or she deletes all the alerts matching the query to eliminate "noise" from the collection of alerts. When an administrator is convinced that a certain query is *always* noise, it may be a good idea to create a "pass" rule in the Snort engine to eliminate these unnecessary alerts, or tune the alert to eliminate the false positives (for example by "whitelisting" the hosts causing the alert).

Alert Links

Each kind of alert references some Snort rule ID (sid). Many such rules also reference alerts described by other organizations, such as Bugtraq, Nessus, etc. Links on each result row lead to online resources describing those threats in greater detail. The online nature of these links allows those third parties to keep information up to date and involve the security community in the collaborative development of those descriptions, without requiring users of Snort/BASE to continually update local copies of the information.

Similarly, when a query has selected a specific host by IP, multiple Internet query tools are available, including geolocation, DNS, Extended whois, etc. Other tools like nmap or OS fingerprinting software can be run from the administrator's host. These tools can be very useful in determining the nature and severity of a threat.

Some alerts do not have useful alert links.

Classifying Network Traffic

There are many potential threats that a NIDS may recognize, including worms, denial-of-service attacks, and other attacks. Identifying these events is a matter of writing rules incorporating unambiguous signatures whenever possible.

As explained above, most worms, denial of service attacks, and the like have strategies that target vulnerabilities in specific versions of applications that can often be recognized by simple signature matching rules. For example, Code Red, or the so-called Ping of Death can be easily recognized through simple stateless packet analysis. (The simpler the analysis, the less load on the sensor.)

Other attacks cannot be so easily recognized. For example, an attack that is successful against a certain version of a web application is not likely to be successful against later versions of the same software. Unfortunately, the NIDS does not know whether you are running a vulnerable version of the software. Further complicating this matter is that many web applications use similar names, such as "calendar.php" -- while some versions of a project may be vulnerable, others are not related beyond the filename.

In the case that a system *is* using software named "calendar.php", the administrator is likely to receive many alerts that represent proper and expected use. It is up to that administrator to either disable the alert, tune it in some way, or put up with the alerts generated. It is preferable to tune the alert if at all

possible. Sometimes the attack signature itself cannot be made more specific, but perhaps all valid users come from some finite IP range, or a limited set of IP addresses that could be coded into the rules.

It is always preferable to tune a rule rather than disable it if there is any chance that the rule could represent a real threat. Disabling the rule blinds the sensor and eliminates the protection it may provide now, or in the future.

False Positives and other Problems

For all these reasons and more, The major cost of running a NIDS on your network is the time cost of properly (and continuously) configuring and tuning the rules to reflect your network. Obviously, it's important to craft rules to capture new, emerging threats. However, a more immediate problem is that certain deficiencies can result in an overwhelming number of false positives, which in turn reduces the usefulness of a NIDS.

If a NIDS is to be useful, the rules it uses must be tuned to the local network and the kinds of traffic it hosts. Similarly, when rules cannot be tuned or eliminated, in-depth knowledge of the local network is critical for correctly interpreting the alerts. Besides the obvious issue of "bad rules" which are not well formed or are simply wrong, we discuss one general problem (*erroneous rules*) and three different kinds of false positives NIDS rules commonly present: *naive rules*, *paranoid rules*, and *ambiguous rules*.

Erroneous rules are a result of Snort rules not being written correctly. This can result in false positives, but just as likely will result in false negatives -- when an alert should be raised but isn't. *Naive rules* attempt to define a threat but are not comprehensive and thus miss potential attacks. On the other hand, *paranoid rules* classify anything that *might* be a threat as a genuine threat. *Ambiguous rules* result when rules cannot be made completely unambiguous. All three false positives create alerts whose significance is obscured by ambiguity or other issues.

Erroneous Rules

Erroneous rules simply result from programmer error either in the conception, design, or execution of the creation of a Snort rule -- it's a rule with a bug. This can result in rules that fail to match what they should, or match the wrong things. Regardless, erroneous rules are separate from the following kinds of false positives:

Naive Rules

Sometimes a single rule isn't expressive enough to identify an entire class of attacks. In order to express the class, a set of rules must be defined. An example of a naive *ruleset* would be one designed to match SQL injection attempts merely looked for SQL comment symbols ("--") in a URI request. This is naive because not all SQL injections require the use of SQL comments. It may be a useful rule -- but it is a fatal mistake to think of this rule as being comprehensive.

Paranoid Rules

In some ways, *paranoid rules* are the opposite of *naive* rules. A *paranoid rule* is a rule that matches any traffic that is even remotely like a known threat. For example, some exploits rely on escaping characters in HTML queries that will be expanded by a script or the web server to some nefarious effect. A paranoid rule might raise an alert for any escaped HTML character. While the character *could* realistically be a threat, it is most likely not. As another example, a *paranoid* rule could easily be defined to flag **any** URI with more than 150 characters a "buffer overflow attempt". While most CGI scripts do not require 150 characters of input, the mere existence of a 150-character URI request does not equal a buffer overflow attempt.

By now you should be able to see another challenge of designing good NIDS rulesets -- the difference between a naive rule and a paranoid rule depends strongly on the network being protected and the other rules in place. For example, if it is known by an administrator that *no* proper CGI queries should *ever* be longer than 64 characters, then it should be a policy violation to have longer CGI queries and it is perfectly reasonable to create a rule flagging longer queries. Likewise, if the same administrator is *certain* that "../" is *never* used in URIs, it is reasonable to raise alerts for that event (as long as you are sure not to think of that as a comprehensive ruleset). On the other hand, for an administrator with a server that uses escaped HTML entities or may have valid URIs containing "../", these rules will probably create a lot of noise -- unless they are tuned to ignore the cases where those actions are expected. This problem is exacerbated by hosts that serve many purposes -- comprehensively describing their traffic may be very difficult when a single-purpose server would be *much* easier to accurately profile.

Ambiguous Rules

Ambiguous rules usually arise because sometimes the best signature available for a threat may also have benign (or even desirable) occurances -- sometimes there is no way to more specifically define the threat. As a result, occurrances of those alerts must generally be investigated even if it is unlikely that they represent real violations. As mentioned above, sometimes ambiguous rules can be tuned for the local network by whitelisting certain addresses, etc., but the defining characteristic is that the signature itself cannot be improved.

Introduction

You are the system administrator at a small consulting firm. The firm has a Linux server connected to the Internet for all its internal functions. This server serves personal and company web pages, sends and receives email, hosts a web forum, image gallery, acts as a file server, firewall and NAT proxy, and also serves as an interactive shell server for the employees. **(For more specific system details, see the section NIDS Network Information below.)**

Like every other computer, router, and toaster on the Internet, this server is subject to fairly constant attacks. Most of these attacks are automated, but some recent attacks seem like they might have a live human on the other end. You keep the server up to date, and you do your best to manage the computers behind the firewall, but you can't be everywhere at once, and users are... users. They can't really be trusted.

As it turns out, your company *has* been doing some consulting for a few high-profile clients, and your boss has suddenly become very concerned about the nature and quality of these attacks. Are they typical Internet "background radiation"? Or are they the work of skilled hackers? What other threats are out there that we can't see? He asks you to set up a Network Intrusion Detection System, let it run for a while, and draft report detailing what the NIDS illuminates.

Your boss has asked you to write a report based on what you discover with your new NIDS. We have done the work of setting up and configuring the NIDS, and let it run for a while collecting packets. Your job is to use the NIDS frontend and shell access to the NIDS backend to diagnose the kinds of traffic and attacks you see appearing in the alert database.

Assignment Instructions

NIDS Network Information

In order to make informed decisions regarding these rules, you need to know something about the network that generated the alerts. Accordingly, the following information is a brief summary of the machine being protected by the NIDS. You should use this information when considering how to classify the alerts in the Snort database. For example, an alert regarding an IBM Lotus Notes server would be irrelevant because *this system does not use IBM Lotus Notes*.

Network Overview

The host being protected is at IP address **150.156.40.132**. It runs several services on its internet-accessible interface, as well as serves as a NAT gateway (like a DSL router with potentially many computers behind it). It has a local interface (an IP address on the internal LAN) and some friendly local neighbors (computers on the LAN) in the **10.10.x.x** private range.

Network Services

The server runs these services:

- Postfix SMTP server
- Apache2 Web server
- Shell access for users
- MySQL (only available to localhost)
- IMAP mailbox server

In addition to typical network traffic for its services, the server checks alleged SMTP servers to see if a mail server is actually running on port 25.

The Apache2 Web server:

- serves a web forum.
- serves several wikis using the MoinMoin wiki program.
- Provides webmail via Squirrelmail
- Hosts several domains, including:
    - lowtek.net
    - hightek.net
    - saskatoon.org
    - electroproject.org
    - rockinrobin.com
    - janejorgenson.com

Friendly Hosts

There are a number of known external (but friendly) hosts that regularly use our services. They include:

- lebistro.org
- isoamerica.net
- frobozzco.net
- zongo.org

Setup

1. If you don't have an account, follow the instructions in the introduction to DETER document.
2. Log into DETER.
3. Create an instance of this exercise by following the instructions here, using `/share/education/NIDS_UCLA/nids.ns` as your NS File.
   - In the "Idle-Swap" field, enter "1". This tells DETER to swap your experiment out if it is idle for more than one hour.
   - In the "Max. Duration" field, enter "6". This tells DETER to swap the experiment out after six hours.
4. Swap in your new lab.
5. After the experiment has finished swapping in, log in to the node via ssh.

You can view the tcpdump logs via less or your favorite editor.

> You probably want to create an ssh tunnel to the experimental node so that you can access the web interface (see below).

Tasks

Part 1: Analyze Alerts

To access BASE and the Snort database, swap in your experiment, set up an ssh tunnel and navigate to `http://localhost:PORT/base/`.

Answer the following questions for each of the 15 unique alert types:

- Is this alert important or unimportant?
- Should it be disabled or tuned?
- Is this rule proper, or would you describe it as erroneous, naive, paranoid, or ambiguous?

Your report should include a list of every unique kind of alert generated by the NIDS followed by a brief description of the alert, gathered from the SID or other Internet resources. *Limited* quotations from online resources is acceptable, but you **must** either cite your sources or rephrase the information in your own words. Some rules do not have clear descriptions online, and do not have rules in the database because they are part of Snort itself. If you can't find any information online, you should still analyze the rule and write a description based on your understanding.

Answer in this format:

ALERT: 1463 "CHAT IRC message"

- Description: This alert is raised any time a "private message" (PRIVMSG) is sent over the network. It is classified as a policy violation and has no specific security implications.
- Unimportant: I see no specific reason why IRC should be disallowed.
- Tuned: PRIVMSG is not the best indicator of an IRC chat session, because it is just one kind of IRC message (a private person-to-person message).
- Ambiguous: If PRIVMSGs are *really* a policy violation, ok, but... this seems a little unecessary. (I could also call it paranoid.)

Make sure you use the NIDS network information when considering the rules. Some alerts may not be applicable to this network! Finally, we recognize that this is somewhat open-ended -- any reasonable and supported answer will be accepted.

Part 2: The Big Picture

Synthesize the "big picture" of the network, including:

1. What is the most common kind of alert?
2. What percentage seem to be attacks versus false alarms? (Your research in part 1 should have identified some alerts as clearly false

alarms).
3. In your opinion, what are our most critical threats at this time?
    - which (if any) attacks have a meaningful chance of success?
    - can we mitigate those threats?

Note: If you lack complete information, say so and explain how more information would be used. **For example**, if you see a traffic trend from a particular address, but you don't know if that address is friendly or not, explain what difference more information could make.

Part 3: Creating Snort Rules

Snort rules are not "compiled" like etterfilter but instead are more like `iptables` rules. For reference, we have copied the set of rules used to create the Snort database into /etc/snort -- you can find all the rules there, and it is probably easiest to start with an existing snort rule and change it.

Write and include in your submission two "new" Snort rules:

1. Create a pass rule for ICMP PING packets from the host generating the most ICMP PING messages. (Hint: Start with the ICMP PING rule on the NIDS server.)
2. Create a rule to log DHCP traffic. This rule could be used to detect spurious DHCP servers on the network.

Note: Snort is not actually installed on the `nids` node, so you will have to manually "syntax check" your rules to make sure that they are correct.

Part 4: Reading Network Traces

A friend of yours brought over some tcpdump traces he made, and he wants to know what they mean, what's happening in them, and whether there are any important security implications with them.

There are 3 `tcpdump` traces located in /root/ on `nids`. Examine each one and *briefly* answer these questions for **each** (a sentence or two per question):

1. **What is likely to be happening in this capture? (The more information, the better.)**
2. **Would this traffic typically be considered a threat, normal behavior, or neither? Why?**

3. **Who are the likely source(s) and destination of this traffic?**
4. **What protocols (beyond TCP, UDP, and IP) are involved in this traffic?**
    - One trace appears to include *many* protocols, but it really doesn't. What's happening instead?

You should now understand enough about ports, services, and viewing packets to do this work -- but if you are lost, please contact the TA.

What can go wrong

There's not much that can go wrong with this lab.

Submission Instructions

Submit a .doc or .pdf file containing of all your answers and relevant materials to your instructor.