CS473 - Homework # 1 Supplement

A primer on network traffic analysis tools

1 Wireshark and tshark

Two useful tools for completing the homework are Wireshark and tshark. Wireshark is an open-source program for graphically viewing and analyzing packet traces: `http://www.wireshark.org/`. Wireshark (formerly known as Ethereal) is the most popular tool of this type. It runs on all major operating systems. Go ahead and install it on your machine. A companion tool that comes with Wireshark is `tshark`—Wireshark's textual command-line counterpart.

Wireshark allows you to use a GUI to manually explore packet traces, so it is mot convenient for interactive use. tshark can be helpful if you want to analyze the trace with a shell script. Another tool similar to tshark is `tcpdump`, which is older and more well-known/wide-spread.

All of these tools (and Bro, as discussed below) can be used in two modes: live capture of packets from the network interface of the machine running the program, and reading a trace from a file. For this homework, you will only need to use them in the latter mode. (Note that live capture often requires administrator access due to its security/privacy implications.)

We recommend you begin the homework by loading a trace into Wireshark and spending a little time looking through it and familiarizing yourself with Wireshark's features. You might then try a similar exploration using tshark. You invoke them as `wireshark` and `tshark`, respectively.

Here are some more tips to get you started with Wireshark:

- One of Wireshark's most important capabilities is its filtering system. Filtering lets you display only a subset of the packets. This is very helpful when dealing with large traces, to let you focus on a small subset of the packets. You can configure a filter by typing an expression into the box at the top of the window, to display only the packets relevant to what you are investigating. Wireshark provides a special language for these expressions, which you may want to learn to some extent.

- Try clicking on the "Filter:" button to see a list of examples of filtering expressions. Select each of the filters listed in the popup box one by one and take a look at the expression that appears for each in the bottom box.

- Expressions can specify a protocol (e.g., `http`). You can also filter on values in headers (e.g., `ip.src==1.2.3.4` or `ip.src==1.2.0.0/16` or `tcp.port==80`).

You can combine filters using logical expressions (e.g., `http || telnet` or `http && ip.src==1.2.0.0/16`).

- For a complete list of the supported protocols, click the "Expression..." button in the main window. The vast majority of these won't be useful on this homework, but the list will give you an idea of how comprehensive the tool is.

- To get an overview of the protocols that are used in the trace, you might try "Statistics" then "Protocol Hierarchy." You can right-click on an individual protocol, then click on "Apply as Filter" and "Selected" to add a filter that selects just that protocol.

- To get a list of the endpoints that appear in the trace, you can click on "Statistics" then "Endpoints", then select either the "IPv4," "TCP," or "UDP" tab at the top. You can right-click on individual end host addresses to add a filter that selects just packets associated with that endpoint.

- Another useful feature is Wireshark's ability to reassemble TCP streams. Try right clicking on a TCP packet and selecting "follow TCP stream." You can use this feature to read the contents (HTML and the like) of a web page someone loaded over HTTP, for example.

- You will probably want to maximize the Wireshark so that it uses the entire display, to maximize the number of packets you can view at a time.

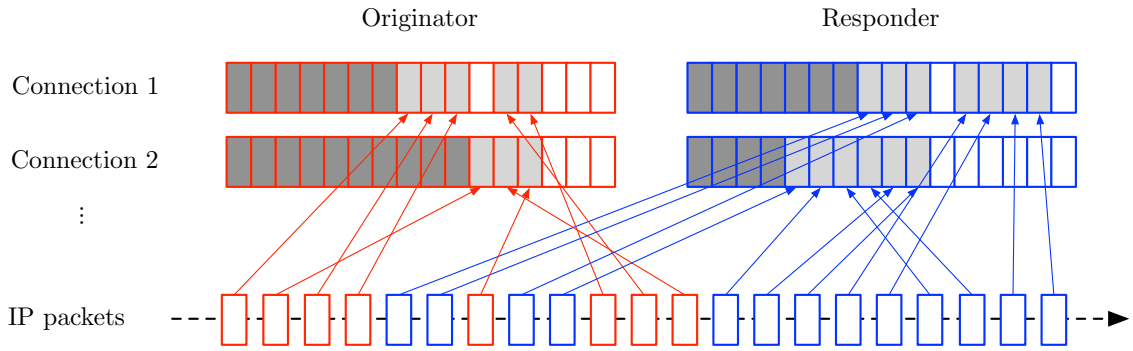
You can learn more about using Wireshark and tshark at `http://www.wireshark.org/docs/`. Note, however, that we designed the homework so that the bulk of the analysis is particularly well-suited to tackle using Bro, which we describe in the next section.

2 The Bro NIDS

The Bro [**?**] network intrusion detection system (NIDS) provides a powerful framework for analyzing network traffic. In this homework, you will use Bro to analyze packet-level traces that contain vulnerabilities and attacks.

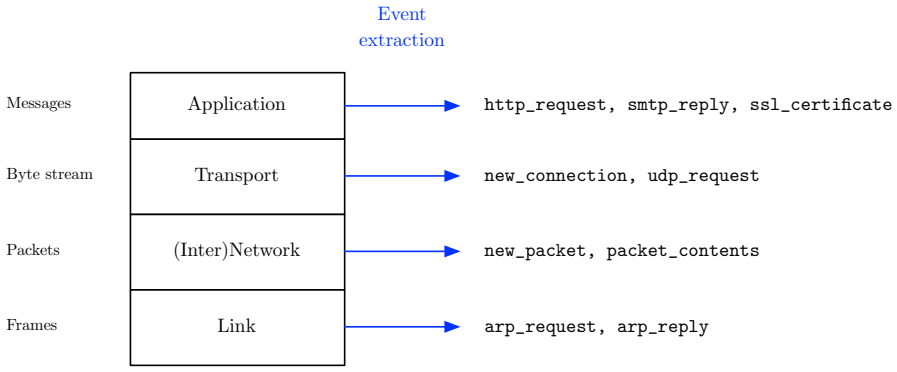
Bro is quite different from other popular network analysis tools such as Wireshark or Snort [**?**]. It is not limited to basic filtering via regular expressions but has its own rich scripting language geared towards in-depth analysis of traffic.

From a high-level perspective, Bro works as follows: it reads packets from a source, such as a network interface or PCAP trace, and reassembles them into a *connection* that represents communication between two endpoints. This process is illustrated for TCP below.



Bro demultiplexes the IP packets captured from the network source into their corresponding connections, which are defined by the 4-tuple of originator address, originator port, responder address, and responder port. Bro determines the application protocols (e.g., HTTP or SMTP) present in a given connection and runs corresponding *analyzers* on the data transmitted using the connection. These analyzers generate *events* that summarize the particular type of network activity. For example, when Bro sees a new TCP connection, it generates an event with the IP addresses and ports of the endpoints. In general, events are "policy neutral" in the sense that they merely reflect what's going on in the network, without a determination of whether it reflects some sort of security problem.

The figure below illustrates the event generation process, which occurs for multiple network protocol layers:



To the right the figure lists as examples just a few of the many events that can occur reflecting activity at the given layers.

Users write *policy scripts* that contain the counterpart to events, *handlers*. Handlers are code blocks that execute when an event is generated. For example, if you want to track the number of rejected connections, you would write an

event handler that increments a counter when the event `connection_rejected` is generated.

Numerous analyzers and policy scripts for a wide range of transport and application-layer protocols ship with Bro. For the most part, you will use existing scripts and do not need to write your own ones.

2.1 Installation

Bro requires a Unix-like platform such as Linux or Mac OS. You can build it directly by following the instructions in the next section. Note, however, that it will not build under Windows or Solaris, so you will need a VM to build it.

2.1.1 Installing it by yourself

Installing Bro follows the standard autotools procedure. That is, (i) you have to choose an installation *prefix*, i.e., the directory under which you like to install the files, (ii) build the program, and (iii) copy it into the prefix. You can use the following steps to install Bro under the prefix `PREFIX`:[1]

```
$wget ftp://bro-ids.org/bro-1.5.3.tar.gz
$tar xzf bro-1.5.3.tar.gz
$cd bro-1.5.3
$./configure --prefix=PREFIX --disable-broccoli --disable-broctl
$make
$make install
```

If all went well, you should now have a Bro executable at `PREFIX/bin/bro`. If you installed Bro using a different prefix from `/` or `/usr/local`, you should add the absolute path to your prefix to the `PATH` environment variable, so that you can directly invoke `bro` from the command line. Finally, you need to set the `BROPATH` environment variable to tell Bro where to find your own policy scripts, say `$HOME/scripts`. In Bash-like shells, you can perform these two steps as follows:

```
$export PATH=$PATH:PREFIX/bin
$export BROPATH=PREFIX/share/bro/policy:$HOME/scripts
```

To test whether everything worked out as well, you can type `bro -h` to display the usage information.

2.2 Invocation

Bro can read network traffic from a PCAP trace file or from a network interface. To read packets from a trace, we pass Bro the trace file via the `-r` switch on

[1] The default prefix is `/usr/local/bro`. You can get a list of full configuration options with `./configure --help`.

the command line.[2] We also have to tell Bro which policy scripts we want to load. To this end, specify a space-separated list of script file names at the end of the command line. For example, to activate the connection, DNS, and HTTP analyzer (request and replies only), you could use the following invocation:

```
bro -C -r trace.pcap conn dns http-request http-reply
```

The `-C` switch tells Bro to ignore checksum errors. (That is, Bro will still process packets whose checksums fail to verify, even though normally systems will discard such packets.) This option is necessary for this homework because some of the packet traces have such errors due to the process by which they were created.

If you are interested to see a list of all of the available analysis scripts, look in the directory `PREFIX/share/bro`. All files ending in `*.bro`, as well as those in your `$BROPATH`, are valid names to pass on the command line. The `.bro` suffix is optional, so for example the name `conn` in the example above will cause the file `conn.bro` to be loaded.

When you run Bro, you will often not see much output on the console. Instead, Bro creates log files ending in `.log` in the current directory. For instance, `conn.log` is the file of the connection analyzer, `dns.log` of the DNS analyzer, and `http.log` of the HTTP analyzer.

2.3 Log Analysis

After having generated several log files, let us now understand how to interpret them. We will look at some particularly handy log files, `conn.log` and `http.log`.

2.3.1 The Connection Analyzer

The `conn.log` file concisely summarizes the network activity of communicating endhosts. Each line represents a single connection. Here is an example:

```
1258133122.311639 0.351037 192.168.1.2 199.7.58.72 http 1247 80 tcp 548 2057 SF X %2
1258133122.233438 18.981099 192.168.1.2 63.245.209.91 https 1245 443 tcp 1094 5652 SF X @93
1258133305.205123 0.000000 192.168.1.2 192.43.244.18 ntp 4234 123 udp 48 ? S0 X
```

The columns have the following meaning.

[2] Although not required for this homework, you might find it interesting to monitor your own network traffic. To this end, replace `-r pcap.trace` with `-i` *`interface`*, where *`interface`* is the name of your network card. Depending on your operating system, this could be for example `eth1`, `wlan0`, or `en1`.

Column	Meaning
1	Timestamp (epoch seconds)
2	Duration (seconds)
3	IP address of the connection originator
4	IP address of the connection responder
5	Application-layer protocol
6	Port of the originator
7	Port of the responder
8	Transport protocol
9	Bytes sent by the originator
10	Bytes sent by the responder
11	Connection status flag
12	Direction flag
13	Application-layer protocol identifier

Column 12 tells the TCP connection status as seen by Bro. Here are some important values (the complete list is available at [**?**]):

State	Meaning
S0	Connection attempt seen, no reply.
SF	Normal establishment and termination.
REJ	Connection attempt rejected.
RSTO	Connection established, originator aborted via RST.
RSTR	Connection established, responder aborted via RST.

This convenient line-based and space-separated format facilitates the quick extraction of the desired information with Unix utilities. For example, to get a quick histogram of the application-layer protocols in the trace, use this one-liner:

```
$awk '{ print $5 }' conn.log | sort | uniq -c
```

This is how you list the top-10 connections by duration:

```
$sort -rn -k 2 conn.log | head -n 10
```

To filter a specific IP address, you could do either of the two:

```
$fgrep 1.2.3.4 conn.log
$awk '$3 == "1.2.3.4" || $4 == "1.2.3.4"' conn.log
```

(Note that the first of these can have "false positives," for example by matching activity involving a host `71.2.3.48`. Use of `fgrep` rather than `grep` helps avoid additional false positives; `grep` interprets its argument as a regular expression, so '`.`'s can match any character, but `fgrep` treats the argument as a fixed string.)

2.3.2 The HTTP Analyzer

A number of the attacks in this homework concern web traffic, which generally means use of the HTTP protocol. Consider the following example entries from the `http.log`, as generated when loading the scripts `http-request` and `http-reply`:

```
1297822344.650231 %2 start 128.32.131.208:64574 > 74.125.224.20:80
1297822344.652659 %2 GET / (200 "OK" [318] google.com)
1297822344.986041 %3 start 128.32.131.208:64575 > 74.125.224.20:80
1297822344.989589 %3 GET /favicon.ico (200 "OK" [329] google.com)
```

The first line shows the initiation of a new HTTP connection, denoted by `start`. The syntax is `source:port > destination:port`. The corresponding TCP connection for a HTTP session is referenced in the second column. Note that this globally unique identifier (`%2` or `%3` in this example) is also used in other log files, such as `conn.log`. Each line that has the same ID belongs to the same connection.

The log represents requests and responses using the following syntax:

```
METHOD path (response_code [bytes transferred] host)
```

The part in parentheses contains the response to the request and only appears when loading the script `http-reply` in addition to `http-request`. For example, ignoring timestamp and connection ID, the line

```
GET /favicon.ico (200 "OK" [329] google.com)
```

represents a GET request for the file `favicon.ico` from `google.com`, which was successfully returned (200) and has a size of 329 bytes.

To get a list of URLs for GET and POST requests, you could use the following one-liner:

```
$awk '/GET|POST/ { sub(/\)$/, "", $NF); print $NF $4 }' http.log
```

Another important HTTP script is `http-header`, which displays each header of request in response in a single line. Here is some sample output:

```
1302221602.533038 %1 start 128.32.45.53:60883 > 128.32.244.172:80
1302221602.533038 %1 > HOST: www.eecs.berkeley.edu
1302221602.533038 %1 > CONNECTION: keep-alive
1302221602.533038 %1 > USER-AGENT: Mozilla/5.0 (Macintosh; U; Intel Mac OS ...
1302221602.533038 %1 > ACCEPT: application/xml,application/xhtml+xml,text/html;q=0.9,...
1302221602.533038 %1 > ACCEPT-ENCODING: gzip,deflate,sdch
1302221602.533038 %1 > ACCEPT-LANGUAGE: en-US,en;q=0.8
1302221602.533038 %1 > ACCEPT-CHARSET: ISO-8859-1,utf-8;q=0.7,*;q=0.3
1302221602.533038 %1 > COOKIE: __utma=42; __utma=4711
1302221602.649961 %2 start 128.32.45.53:60884 > 209.160.22.33:80
1302221602.649961 %2 > HOST: virusscan.jotti.org
1302221602.649961 %2 > CONNECTION: keep-alive
1302221602.649961 %2 > REFERER: http://virusscan.jotti.org/en
```

```
1302221602.649961 %2 > USER-AGENT: Mozilla/5.0 (Macintosh; U; Intel Mac OS ...
1302221602.649961 %2 > ACCEPT: */*
1302221602.649961 %2 > ACCEPT-ENCODING: gzip,deflate,sdch
1302221602.649961 %2 > ACCEPT-LANGUAGE: en-US,en;q=0.8
1302221602.649961 %2 > ACCEPT-CHARSET: ISO-8859-1,utf-8;q=0.7,*;q=0.3
1302221602.649961 %2 > COOKIE: sessionid=b45777470be76f47110b5faaedb0ece172c1e; lang=en
1302221602.690460 %1 < DATE: Fri, 08 Apr 2011 00:13:22 GMT
1302221602.690460 %1 < SERVER: Apache
1302221602.691400 %1 GET /Scheduling/CS/ (200 "OK" [15128] www.eecs.berkeley.edu)
```

The additional header lines are demarcated with an additional '>' or '<' for an HTTP request and an HTTP response, respectively.

References

[1] Bro connection states. http://www.bro-ids.org/wiki/index.php/Reference_Manual:_Analyzers_and_Events#Connection_summaries.

[2] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. 31(23–24), 1999. Bro homepage: http://www.bro-ids.org.

[3] Martin Roesch. Snort: Lightweight Intrusion Detection for Networks. In *Proceedings of the Systems Administration Conference*, 1999. Snort homepage: http://www.snort.org.