Paxson
Spring 2011

CS 161
Computer Security

Project 2

Due: Thursday May 5, 11:59pm

Updated 22Apr11: We linked the two Bro scripts at the end of Section 2. Thus far they were only available in the VM at `/root/bro-scripts`.

1 Background

Huge Big Dairy is a farming and poultry conglomerate run out of Madison, Wisconsin. They pride themselves on their yogurts, brie cheeses, and buffalo wings (made out of Real Buffalo™). However, Huge Big has many detractors who allege that the company not only manifests incompetence when it comes to dairy products, but also a propensity towards venal undertakings.

During an ill-advised television interview, Huge Big's CEO, Chuck "Mondo" Cheeze, brashly trumpets his company's expertise, not only in all things dairy, but their emarketing prowess and homegrown Internet security savvy. Mondo's biggest gaffe, however, is to imply that he does not consider cows as bovines—instantly incurring the wrath of the shadowy underground hacker group *Synonymous*, whose members unite in their violent objection to any terminology errors that confuse whether two words have the same meaning.

Synonymous decides to humiliate HBDairy, exposing their secrets and incompetence, and disrupting the activity of their employees. In a series of Internet attacks that HBDairy finds itself powerless to counter, Synonymous deeply embarrasses the company. Eventually, HBDairy must admit they have been outmatched, and in desperation they turn to expert outside help: **you**. They commission your team to analyze how Synonymous achieved their exploits. Luckily, the one facet of computer security they managed not to screw up is logging: they have full packet traces of all of the systems in question.

One gloomy morning as the end of the semester looms, you head out to Richmond Field, board the HBDairy corporate jet, and 5 hours later find yourself at their offices in Madison, armed only with a trusty VM image that contains all of your analysis tools. You need to complete your forensic analysis, file your report, hop back on their jet, and return to Berkeley—with enough time left in RRR Week so you can adequately prepare for your final exams.

2 The Project

The goals of this project are to build up your familiarity with both (1) how real network attacks manifest, and (2) how to effectively employ some widely available tools for analyzing network activity.

Collaboration. We intend for you to work on this project in **teams of two**. Beyond your team, you may not collaborate with other students. You can share general information on how to use

Wireshark, tshark, and Bro with other students if it is not specific to the questions on this project, but you must not share tips, advice, hints, etc. on how to solve any of the questions on this project.

You must write up your solutions entirely between the two of you on your team. You must never read or copy the solutions of other teams, and you must not share your own solutions—including partial solutions—with other students. If you have any questions, please contact the instructors.

VM. To minimize the steps required for you to have a working analysis environment, we have pre-built VirtualBox and (soon-to-be-available) VMware virtual machine (VM) images that come with both the network traces and the analysis tools pre-installed.

VirtualBox is freely available open-source software running on Windows, Linux, Macintosh, and Solaris. You can download it from:

http://www.virtualbox.org/wiki/Downloads

VMware is installed on the instructional machines *iserver1.eecs.berkeley.edu*, *iserver2.eecs.berkeley.edu*, and *iserver3.eecs.berkeley.edu* (all running Windows). You can log into any of those remotely with a Remote Desktop Client, as discussed at http://inst.eecs.berkeley.edu/connecting.html#labs (and in particular at http://inst.eecs.berkeley.edu/cgi-bin/pub.cgi?file=microsoft-rdc.help).

The VMs are based on the BackTrack Linux distribution. You can login with username `root` and password `toor`. After logging in, you can type `startx` to launch the X window system or SSH to the machine (the steps to obtain the IP address are described below). If you choose to work remotely via SSH, use the `-X` switch to enable X forwarding, allowing you to run graphical tools like Wireshark on your local machine over the SSH connection.

You will perform all actions as user `root` whose home directory is `/root`. Inside is a `bro-scripts` which contains two Bro policy scripts you might find useful. If you opt to work with the VM over SSH, you can login remotely via `ssh -X root@`*IP-address-of-VM*.

Depending on the virtual machine software you wish to use, follow the corresponding instructions below:

VirtualBox: After having installed it, you should change the IP address of the VM host to be static. To this end, launch VirtualBox and click on "Preferences" and then "Network," where you should see a list of "Host-only Networks." Usually you are presented with one single interface `vboxnet0`; if not, select the interface you want to choose with the guest VM. Go ahead an click the little screwdriver button to edit the interface settings. In the "Adapter" tab, change the IPv4 address to 10.1.1.1 (and leave the subnet mask at 255.255.255.0). In the "DHCP Server" tab, uncheck "Enable Server" to disable the DHCP server. Next you need to download a copy of the VM image, which you can download from:

http://www.eecs.berkeley.edu/~mobin/teaching/cs161vbox.tar.bz2

Once you have downloaded it, extract the image and import it into VirtualBox by clicking "File" followed by "Import Appliance." The import should also take place by double-clicking the file `cs161vbox.ova`. To launch it, simply press the "Start" button.

VMware: A copy of the VMware image will soon be available at:

http://www.eecs.berkeley.edu/~mobin/teaching/cs161vm.tar.bz2

To find out the IP address of the VM, run the command `ifconfig` from the terminal.

Traces. You can retrieve the HBDairy traces from:

http://www.eecs.berkeley.edu/~mobin/teaching/cs161traces.zip

If you decide to use the VM environment we have prepared for you, then it also comes pre-loaded with the traces at `/root/traces`. These traces store the corresponding packets in the PCAP[1] file format, a simple and widely used standard.

Tools. Once you have obtained the traces, you can use whatever tools you wish to analyze them and come up with your answers to the forensic challenges that appear below. You will not need to submit any code with your answers, though we will ask for a list of the tools you used.

In general, we believe you will find two tools particularly helpful. The first is *Wireshark*, an open-source program for graphically viewing and analyzing packet traces, and its command-line counterpart, *tshark*. Wireshark and tshark are particularly useful for packet-level analysis. The second tool is *Bro*, an open-source scriptable program/environment for network traffic analysis. For the most part, Bro operates at a higher semantic level than individual packets, though it has some support for per-packet analysis. We intend for Bro to suffice to solve most of the forensic challenges, though Wireshark and tshark may also prove helpful, and in some cases we have them in mind as the tool of choice. We wrote two Bro scripts that you might find useful during your analysis. If you're using the VM, they reside in `/root/bro-scripts` but you can also download them at:

- http://cs.berkeley.edu/~mavam/teaching/cs161-sp11/facebook.bro
- http://cs.berkeley.edu/~mavam/teaching/cs161-sp11/mime-attachment.bro

[1] If you're curious, you can find a description of the format at the following address, although you will not likely want to write any code that reads the file directly: http://wiki.wireshark.org/Development/LibpcapFileFormat.

3 Some Tips

We have in general designed the forensic questions to *not* require exhaustive manual analysis to answer. You should instead determine how to use various tools in an effective manner.

Further, some questions require the knowledge of specific header fields (which we might not have talked about in the class) to locate a "needle in the haystack." We recommend that before diving into a question, you spend some time thinking about your *analysis plan*, i.e., how you can explore the logs in an efficient manner. This process might benefit from some (often light) web surfing to get ideas about how to locate particular features. You will likely find that Unix utilities such as `grep`, `awk`, `sort`, and `uniq` can prove to be your best friends for this project.

In general, we anticipate that the biggest pitfall for completing this project effectively is the temptation to poke through the logs "blindly" rather than thoughtfully. That often won't work effectively here—and in actual practice will work even less well, because the logs you'll encounter in real life will be *much* larger than those here.

The next sections frame the forensic questions and how to submit your team's solutions. After that comes two sections that discuss use of *Wireshark/tshark* and *Bro*.

4 Submission Instructions

You will be examining a different scenario in each problem. For each scenario, you will submit the summary of your forensic analysis. Use the questions in each problem as a roadmap to present your case. We expect you to provide as strong a case as you can by including the relevant information from the log files formatted according to the following skeleton:

- For the scenarios involving attacks:

```
<Name of the attack>
<Attacker: IP address and domain (if relevant)>
<Victim: (User account | document | IP address)>
<Action of attacker: description supported by evidence from the relevant
log files>
```

- For the scenarios involving vulnerability,

```
<Vulnerability: description of the vulnerability>
<IPs of vulnerable hosts supported by evidence (e.g., a successful attack)>
```

- For scenarios involving information leakage due to sniffing, provide the following:

```
<Name/Email address of the sender>
<Leaked data: Name & contents of the document (as specified by the question) |
Account, password>
```

All submissions for this project will be electronic. For each of the problems, create a text file named `q⟨#⟩.txt`. So, for example, if you submit feedback for the final, optional question, you will put it in the file `q10.txt`.

Please also submit a file `collaborators.txt` that contains the class accounts of the team members.

To submit, put all these files into a directory, and then run `submit proj2`. It suffices to submit the project from the account of one team member.

5 Forensic Questions

Each of the traces is labeled with the corresponding problem number.

Question 1 ***The vulnerable DNS clients*** **(12 points)**

Recall that most DNS clients now select a random UDP source port when making DNS queries to help defend against the Kaminsky attack. When you mention this threat to HBDairy's IT staff, they reply "The *K-huh-what* attack?" Thus, you view it as prudent to assess whether any of their clients making DNS queries lack UDP source port randomization.

Analyze the *q1* trace to assess the sequence of source ports seen for each resolver that makes more than one query. If you find they all appear to be okay, then list the resolvers you analyzed. If you find some that appear to not use randomization, list those and the corresponding evidence.[2]

Note, we have designed this problem to be suitable for familiarizing yourself with Wireshark/tshark, though as in general you can solve it however you wish.

Question 2 ***Information theft from a web server*** **(12 points)**

Mr. Cheeze informs you that some of the sensitive information that Synonymous leaked came from a file on the HBDairy web server, though the file was not part of the content to which the server was supposed to provide access. He is reluctant to tell you the specifics of the information, but wants you to determine how the theft occurred.

Analyze the web accesses in the *q2* trace. Determine the type of attack used to access the file. What was the filename? How do you know it was successfully accessed?

Question 3 ***Email leakage*** **(12 points)**

Due to some other information leaked by Synonymous, HBDairy is certain that someone carelessly forwarded a sensitive document using unencrypted email. Analyze the SMTP traffic in the *q3* trace to locate the document and determine who sent the email. Who appears to have authored the document? What are the two links contained in the document? Explain how

[2] You will find that some of the traffic uses *MDNS*, a form of DNS activity sent using "IP multicast," a type of transmission similar to IP broadcast. For this problem, ignore such traffic.

you located the document. Your method should be significantly more efficient than manually examining every email in the trace in its entirety.

Question 4 ***The mysterious wall post*** **(12 points)**

One fine morning, Udder Kaos, an HBDairy employee, noticed that a secret message that he had sent last night to his colleague Fro Yo using Facebook messaging (i.e., sent to his friend's inbox) was subsequently posted using his account to his friend's wall.

Examine the web traffic in *q4* trace to find evidence of the attack used for the wall post. Sketch the steps of the attacker.

Question 5 ***YouTube becomes NoTube*** **(12 points)**

One of the competitive benefits that HBDairy provides to its employees is on-the-job access to YouTube. Lately, many disgruntled employees have complained that they have lost this benefit because their browsers report "page could not be loaded" when they try to access YouTube.

Analyze the web traffic in the *q5* trace to find out how Synonymous disrupted the YouTube access. How much downtime did this result in?

Question 6 ***The mysterious DairyStock transaction*** **(12 points)**

DairyStock is a stock management web application favored by HBDairy employees that allows registered users to buy and sell stocks and transfer them to each other. Synonymous denounces its use as an example of HBDairy's ineptitude when dealing with Internet security issues, and states that as a demonstration they arranged to introduce a bogus transaction for a "modest" sum of money.

Examine the traffic in the *q6* trace to find the unauthorized transfer Synonymous refers to. Sketch the attacker's steps.

Question 7 ***Ill-Considered Authentication*** **(12 points)**

You discover that the users at HBDairy don't have much sense when it comes to passwords. One employee uses a single password for many different web services, and it appears clear that one of those services uses a weak form of standardized web authentication because Synonymous was able to public demonstrate that they possess the user's password.

Examine the web traffic in the *q7* trace to discover which web server used the weak authentication scheme. What was the user's password?

Question 8 ***The Leaked URL*** **(12 points)**

Among the information leaked by Synonymous is a URL that an employee swears was only shared in an encrypted form. Sure, he talked about it during an unencrypted Facebook webchat with Mr. Cheeze, but the URL itself was encrypted.

Analyze the chat traffic in the *q8* trace. Determine how by sniffing the session Synonymous could have extracted the URL. What do you find when you visit the URL?

Question 9 ***Tools used*** **(4 points)**

Write up a brief description of the tools you used for your analysis (including Wireshark, tshark,

and Bro, if you indeed used them), and your views on their strengths and weaknesses. You do not need to include standard Unix utilities like `grep` or `awk`.

Question 10 ***Optional Feedback*** **(0 points)**

This is a brand new project. We (naturally) hope it proved quite beneficial to undertake, but realize it may have shortcomings, both regarding specific points, and possibly in its overall inception. Accordingly, it would be very helpful to hear any feedback you would like to offer: was it fun, frustrating, annoying, stimulating? How worthwhile was it in terms of learning? What would you change about it in the future? How much time did you spend on it in comparison to Project #1? What were the best and worst facets of the project? Anything else you want to add?

This question is wholly optional, and your comments will not factor into the grading of your project in any way.

6 Wireshark and tshark

Two useful tools for completing the project 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. It comes preinstalled on the VM image (see below); or you can go ahead and install it on your own 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 project, 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 project 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. Both tools are available in the VM; 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 project, 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.

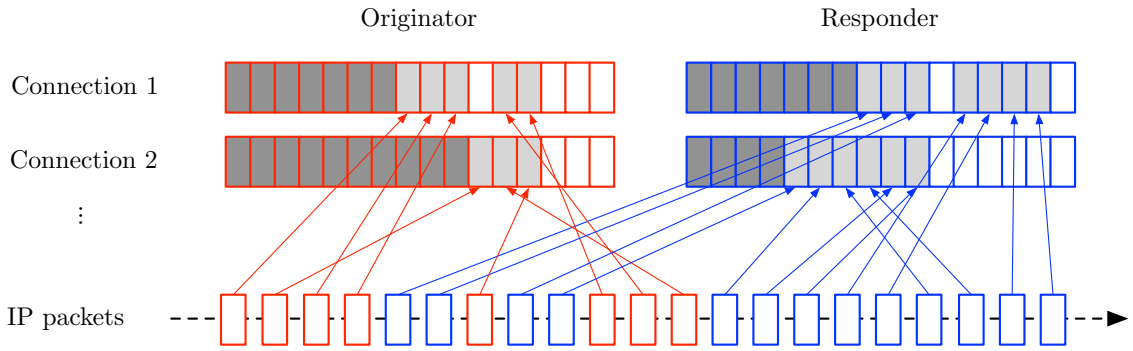
You can learn more about using Wireshark and tshark at `http://www.wireshark.org/docs/`. Note, however, that we designed the project so that the bulk of the analysis is particularly well-suited to tackle using Bro, which we describe in the next section.

7 The Bro NIDS

The Bro [2] network intrusion detection system (NIDS) provides a powerful framework for analyzing network traffic. In this project, you will use Bro to analyze packet-level traces that contain the above attacks.

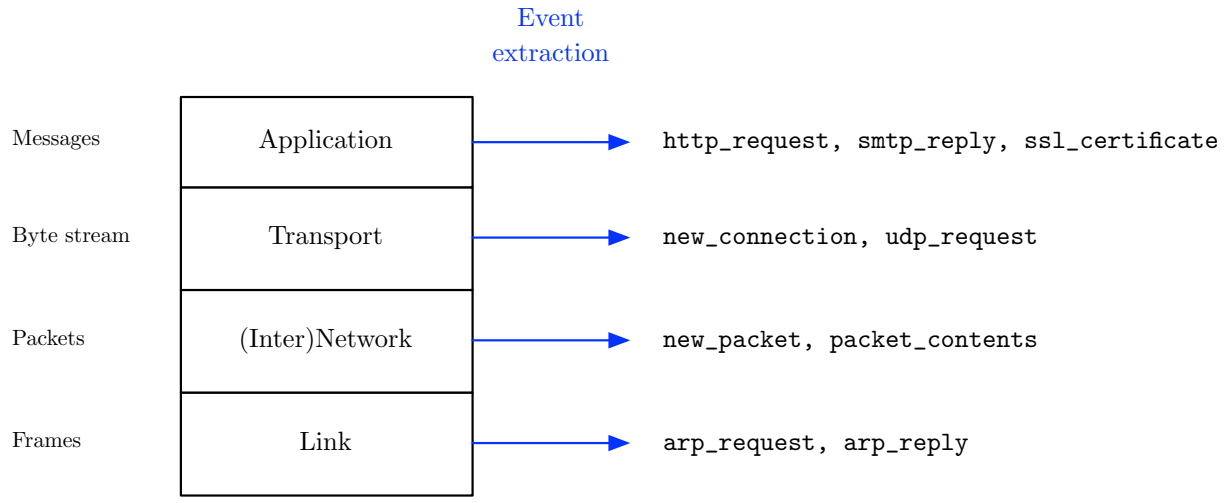
Bro is quite different from other popular network analysis tools such as Wireshark or Snort [3]. It is not limited to basic filtering via regular expressions but has its own rich scripting language geared towards in-depth analysis of traffic.

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.

7.1 Installation

Bro requires a Unix-like platform such as Linux or Mac OS. We have pre-installed it in the VM image mentioned above using the prefix `/usr/local`. You can find the policy scripts under `/usr/local/share/bro`. You can also build it directly if you wish by following the instructions in the next section. Note, however, that it will not build under Windows or Solaris, so you cannot use it directly on the instructional machines other than by using the VM.

7.1.1 Installing it by yourself

You can skip this section if you will use the version of Bro present in the VM.

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`:[3]

```
$ 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
```

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

```
$ 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.

7.2 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 the command line.[4] 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 project 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.

7.3 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`.

7.3.1 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:

[4] Although not required for this project, 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`.

```
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.

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 [1]):

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.)

7.3.2 The HTTP Analyzer

A number of the attacks in this project 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/ { print sub(/\)$/, "", $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=b45777470be76f47110b5faaedb0ece172a25c1e; 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.

7.4 Writing Policy Scripts

We finish with a brief look at how to write your own simple policy scripts. Our main goal with doing so is to familiarize you with the scripting language a bit so you can get a sense of how the different policy scripts provided with the Bro distribution work. You might also find it quite helpful for your forensic analysis to write some simple scripts of your own.

Bro has a richly typed domain-specific language. The language reflects Bro's asynchronous, event-based execution model. A typical Bro script contains several event handlers that fire whenever Bro detects the corresponding activity in the network. Here is a small, fully functional example:

```
global cnt = 0;

event connection_established(c: connection)
    {
    ++cnt;
    }

event bro_done()
    {
    print cnt;
    }
```

This script initializes a global counter to 0 and defines event handlers for the events `connection_established` and `bro_done`. The former executes whenever Bro sees a successfully established connection, and simply increments a counter. The latter prints the counter when Bro terminates. Note the syntax: the argument to `connection_established` is of type `connection`

and named `c`. It is a *record* type, which is a heterogeneous tuple akin to a C struct. You can find its definition in the file `PREFIX/share/bro/bro.init`, where it looks like:

```
type conn_id: record {
       orig_h: addr;
       orig_p: port;
       resp_h: addr;
       resp_p: port;
};

type connection: record {
       id: conn_id;
       ...
       start_time: time;
       duration: interval;
       service: string_set;  # if empty, service hasn't been determined
       ...
};
```

Accessing record fields in Bro is done using the `$` operator (cf. dot in C). The next example uses this feature and is a little more elaborate:

```
global hosts: table[addr] of count &default=0;

event connection_established(c: connection)
    {
    local orig = c$id$orig_h;
    ++hosts[orig];
    }

event bro_done()
    {
    for ( h in hosts )
        print h, hosts[h];
    }
```

The first line declares a global table whose keys are addresses and whose values are counters (i.e., unsigned integers). The `&default=0` part is an *attribute* that specifies a default value for table values. The `connection_established` event handler first extracts the connection originator by `c$id$orig_h` (mnemonic: originating *host*) and then increments a per-address counter. When Bro terminates, it executes any event handlers for the `bro_done` event. Here, we simply print each key-value pair in the hash table.

7.5 Summary

Here we have covered the basic architecture of Bro, the installation procedure, and how to load policy scripts and analyze their output. We also looked briefly at the Bro scripting language to get a flavor of its asynchronous event-driven style. This should suffice to get you started. Naturally, we also expect questions to arise, with Piazzza providing a good place to discuss them.

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.