UC Berkeley CS 168, Fall 2015

CS168 Project 4

(Version 1.0)

Overview

Recall that in Project 3, you implemented a stateless passive firewall: that is, your firewall could do its job by considering each packet individually, and it did not generate traffic.

In Project 4, you will be extending your solution for Project 3 to make a stateful active application-layer firewall. You will use the same framework, VM, test harnesses, tools, and as in Project 3. Now, your firewall should generate packets in response to denied packets. Upon completing this part, you should:

- Be familiar with the HTTP and DNS protocols.
- Understand the difference between stateful vs. stateless, active vs. passive firewalls.

Besides writing code, you will need to (and should) spend a lot of time to understand protocol specifications, to design algorithms, and to test your application. Start working on the project as soon as possible.

Have fun!

Changelog

Always check the latest version of this document. It is your responsibility that your solution conforms to the latest version of the spec.

v1.0 (11/12/2015)

- First release

Requirements

Overview

In the project, you must implement three new rules (The rules filename will still be given with `config['rule']` as in 3).

1. `deny tcp <IP address> <port>`
2. `deny dns <domain name>`
3. `log http <host name>`

The format of `IP address`, `port`, and `domain name` are defined in the same way as in Project 3. For `host name`, see below.

Firewall behavior from Project 3 will remain the same, for instance you should still "pass a packet if none matches" and "follow the verdict of the last matching rule". **However, for Project 4, we will neither test firewall rules defined in 3 (`pass`/`drop` rules) nor country-based matching,** so your grade for Project 3 will be mostly decoupled from your grade for Project 4.

For simplicity, you can make the following assumptions:

- The rules file has always correct syntax, as defined in Project 3.
- All packets seen by the firewall are neither corrupted nor malformed.
- All TCP connections with external port 80 are valid HTTP connections.
    - However, your HTTP header parser should not be overly restrictive. Many web servers in the wild have slightly different implementations, and your parser should be flexible enough to parse them correctly. For example, `content-length` (not `Content-Length`) is a valid header field name. When in doubt, consult RFC 2616.
    - Also, you should be able to deal with out-of-order TCP packets, caused by reordering, drop, or loss. See below for details.

1. Injecting RST Packets: `deny tcp`

In addition to dropping a matching TCP packet, respond to the initiator `(src addr, src port)` with a TCP packet with the RST flag set to 1.

If you simply drop these packets (with a `drop` rule), then the client application will try sending SYN packets several times over the course of a minute or so before giving up. However, if you also send a RST packet to the client (with a `deny` rule), the application will give up immediately.

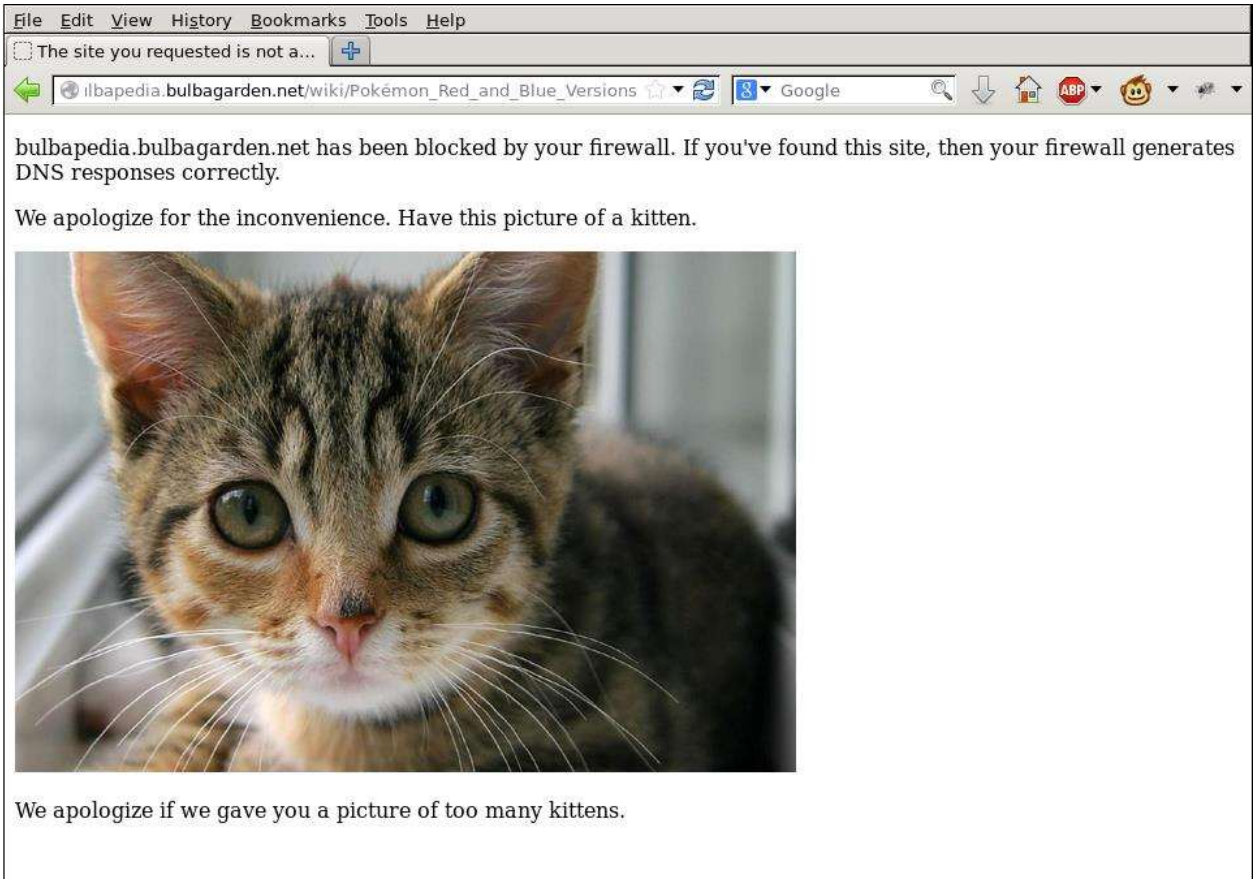
When generating the reset packet you must carefully compute both the TCP and IPv4 checksum; consult Wikipedia or the relevant RFCs for details. http://locklessinc.com/articles/tcp_checksum/ also has some C code (go through the `checksum1()` function) that might help you figure out how to implement these. Please do not copy checksum code directly from the Internet, as we will be using tools to detect copied code.

2. Injecting DNS Response Packets: `deny dns`

In addition to dropping a matching DNS request, send a DNS response to the internal interface pointing to the fixed IP address `169.229.49.130`. Consult section 4.1.1 and 4.1.3 of RFC 1035.

If the QTYPE of a matched DNS request is **AAAA**, drop the packet, and do not send a response.

If you implement this correctly, then your browser will direct you to a placeholder website. Your response should be in the **answer section** should have type **A** (i.e. it is an address) and a TTL of 1 second. Make sure you copy the ID field as appropriate and the RCODE field as appropriate.



If your firewall generates DNS rules properly, you should see a page similar to this.

3. HTTP Log: `log http`

Log Format

For matching host names, log HTTP transactions over TCP connections with external port 80 to `http.log` in the current directory. An HTTP transaction is defined as a **pair of an HTTP request and an HTTP response** (they share a TCP connection). For each transaction, leave a log according to the following format (space-delimited) in a single line:

```
host_name method path version status_code object_size
```

To understand what these values log, consider the following HTTP request:

```
GET / HTTP/1.1
Host: google.com
User-Agent: Web-sniffer/1.0.46 (+http://web-sniffer.net/
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
Cache-Control: no-cache
Accept-Language: de,en;q=0.7,en-us;q=0.3
```

And the corresponding response:

```
HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
Content-Type: text/html; charset=UTF-8
Date: Mon, 18 Nov 2013 23:58:12 GMT
Expires: Wed, 18 Dec 2013 23:58:12 GMT
Cache-Control: public, max-age=2      592000
Server: gws
Content-Length: 219
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Alternate-Protocol: 80:quic
```

For this example you would log the following (word in this section means words as in parts of a sentence, i.e., space separated values):

- `host_name`: Use the value of `Host` request header field. If it is not present, use the external IP address of the TCP connection. (In the above `host_name` is `google.com`)
- `method`: The first word of the request line (e.g. `GET`, `POST`, `PUT`, `DROP`).
- `path`: The second word of the request line. (`/` in this case)
- `version`: The third word of the request line (e.g., `HTTP/1.0` or `HTTP/1.1`) (in this case HTTP/1.1)
- `status_code`: The second word of the response line. (in this case `301`)

- `object_size`: Use the `Content-Length` response header field. Its value will be identical to the actual HTTP response payload size. If this field is not present, then this field should be `-1`. (In this case it is `219`).

In this case, the log should be:

```
google.com GET / HTTP/1.1 301 219
```

Hostname Matching

`host name` in an http rule can be either a domain name or a single IPv4 address (neither prefix nor `any`). Domain names (full/wildcard) are used in the same way as in Project 3. The following shows some examples of valid hostnames.

- `google.com`
  - only matches google.com
- `*.facebook.com`
  - matches foo.facebook.com and bar.baz.facebook.com, but not facebook.com
- `123.45.67.89`
  - matches 1) if the Host header field value is 123.45.67.89, or
    2) if the Host header field is not present and the external IP address is 123.45.67.89.
- `*`
  - matches every HTTP request.

Those host names are matched against the `Host` header field value in HTTP requests. If the header is not present, use the external IP address in the quad-dotted notation as a fallback.

If multiple http rules match, log the transaction only once.

Notes and Hints

Packets to Inspect

- The external port for HTTP requests should be the destination port, and the external port for HTTP responses should be the source port. (We assume you don't run a web server on your own VM.)
- You can assume that all TCP traffic on port 80 is actually HTTP.
- Because we only check port 80 for HTTP, this cannot match HTTPS (nor should it), whose default port number is 443.

Log File

- The name of log file is fixed: `http.log` (in the current directory)
- Your firewall should append to an existing log file, or create it if it does not exist.
  - `f = open('http.log', 'a')` will do this for you.
- **Flush the log file after each write, with `f.flush()`**
  - If you don't call `f.flush()` after each write, the your firewall process would keep buffering data and delay actual writes to the file for unpredictable amounts of time.

- This could mean the autograder sees nothing written by your firewall during grading.
- In short, `f.flush()` after each use!

TCP Reassembly:

- The communication between HTTP applications (web browser/server) with a byte stream for each direction. What your firewall sees is packets segmented by the TCP layer. To parse HTTP requests and responses, you should reassemble TCP segments into byte streams, based on the TCP sequence numbers. This process is similar to what the "Follow TCP Stream" feature does in Wireshark (see the supplementary document).
- The HTTP request/response header may span multiple packets.
- One common example: an HTTP request/response header is bigger than MSS, so it is broken into multiple TCP packets.
  - One extreme example: suppose an HTTP request is segmented into single-byte TCP packets, i.e., "G", "E", "T", " ", "/", " ", "H", "T", "T", "P", … Can your firewall handle this case?
  - Packets may be dropped/reordered arbitrarily. To make it easier to handle this we suggest you drop *out-of-order packets with a forward gap* in sequence numbers, on a per-connection basis, for each direction, so that both the endpoint and the firewall only process in-order packets. TCP RTO will ensure that packets are retransmitted. While this negatively impacts performance, it simplifies code and reduces the amount of state that your firewall needs to maintain.
  - Suppose you were expecting SEQ 4000 for the next packet. If you get a TCP packet with SEQ 5000, you drop the packet.
  - On the other hand, if you get a TCP packet with SEQ 3000, you should **pass** the packet, since it indicates retransmission of lost packets. If you drop the packet, the connection will be stuck.
- Note that TCP sequence numbers are 32-bit unsigned integers, so they can *wrap around*.
- Assume that HTTP is not pipelined (we disabled this feature in Firefox), so requests and responses always begin at the beginning of the payload of a TCP packet.

Dealing with Persistent HTTP Connections

- You should handle persistent HTTP connections. If a connection is persistent (refer to http://en.wikipedia.org/wiki/HTTP_persistent_connection), then message length will be specified via `Content-Length` (which we define as the HTTP payload size in bytes, excluding the header).
- Responses to `HEAD` request do not have HTTP payload, but they may have non-zero `Content-Length` values (Section 14.13 in the RFC document).
- Note that HTTP requests may have non-empty payload (e.g., `POST` method)

Others

- Your firewall must support concurrent HTTP connections. It is typical that a web browser opens tens of connections to access a web page.

    - Dropping out-of-order packets should be done on a per-connection basis.
- Your implementation must not be too slow. For the reference, it should not take more than 10 sec to download a 700 KB file when the bandwidth is not the bottleneck.
- Again, your firewall should not crash.

Examples

1. Consider "`log http *.berkeley.edu`". Opening http://www-inst.eecs.berkeley.edu/~cs168/fa15 in Firefox will produce the following log entries, after clearing the browser cache:

```
www-inst.eecs.berkeley.edu GET /~cs168/fa15 HTTP/1.1 301 254
www-inst.eecs.berkeley.edu GET /~cs168/fa15/ HTTP/1.1 200 273
www-inst.eecs.berkeley.edu GET /~cs168/fa15/content.html HTTP/1.1 200 1586
www-inst.eecs.berkeley.edu GET /~cs168/fa15/overview.html HTTP/1.1 200 2581
www.eecs.berkeley.edu GET /Includes/EECS-images/eecslogo.gif HTTP/1.1 200 828
www-inst.eecs.berkeley.edu GET /~cs168/fa15/images/Book.png HTTP/1.1 200 174
www-inst.eecs.berkeley.edu GET /~cs168/fa15/images/Keycard_A.png HTTP/1.1 200 324
www-inst.eecs.berkeley.edu GET /favicon.ico HTTP/1.1 200 0
```

Details (the ordering, fetched objects, or their size) may vary, due to various reasons.

2. Consider "`log http *`". In a terminal window, "`wget google.com`" will produce the following log entries (again, details may vary).

```
google.com GET / HTTP/1.1 301 219
www.google.com GET / HTTP/1.1 200 -1
```

Note that in Example 1, the request for favicon.ico explicitly specified `Content-Length` as 0, whereas in Example 2, the request to `www.google.com` for `/` did not specify a `Content-Length`, so we used the placeholder value of -1.

Logistics

Submission Instructions

You may submit `firewall.py` for testing once per day. The tests we run daily are similar to at least some of the tests we will run for the final grading, though we may not test all aspects of the project in the daily tests.

Your final grade will be `max(score_on_daily_tests * 0.7, score_on_final_tests)`. In other words, if you get a perfect score on your daily tests, you will not do worse than a 70% on the project. The one exception is that we do not run the full set of anti-cheating tests in the daily runs, so if you manage to get 100% while cheating on the daily tests, you may still get 0% on the final grading. You will use the provided OK tool to submit your work. See http://cs61a.org/articles/using-ok.html for some basic info on OK.

Please submit using ok as early as possible to see if there is any problem with your ok account. Also, please indicate your partner's email in okpy.org if you work in group.

Collaboration Policy

The project is designed to be solved independently, but you may work with at most one partner if you wish. Grading will remain the same whether you choose to work alone or in partners; both partners will receive the same grade regardless of the distribution of work between the two partners.

You may not share code with any classmates other than your partner. You may discuss the assignment requirements or your solutions (e.g., what data structures were used to store routing tables) -- away from a computer and without sharing code -- but you should not discuss the detailed nature of your solution (e.g., what algorithm was used to compute the routing table). Refer to the course webpage for more information. If you are not sure what may constitute cheating, consult the instructor or GSIs. Assignments suspected of cheating or forgery will be handled according to the Student Code of Conduct[1]. Apparently 23% of academic misconduct cases at a certain junior university are in Computer Science[2], but we expect you all to uphold high academic integrity and pride in doing your own work.

1 http://sa.berkeley.edu/code-of-conduct

2 http://www.pcworld.com/article/194486/Computer_Science_Students_Cheating.html

DOs and DON'Ts

DOs

- It is okay to use external libraries or applications to **test** your firewall.
- You can modify not only `firewall.py` but also other source code files, but only for debugging purposes.
    - Do remember that you only submit the `firewall.py` file, and it should work with the original code of other files.
- We encourage you to share test strategies with your fellow students on Piazza, but only at the high level (e.g., no test code).
- We recommend using Firefox in the VM, rather than installing other web browsers. The Firefox installed in the VM was specially configured to suppress some seemingly strange behaviors.

DON'Ts

- Do not create extra threads or processes.
- Do not use any libraries other than Python 2.7 standard modules to implement the firewall.
- Do not alter the network configurations of the VM. Do not install "Network Manager" package. The VM relies on manual/delicate configurations to provide the firewall functionalities. You may have to reinstall the VM if some configuration gets broken.
- **All parts of the solution code must be your own; copying someone else's code snippet (including from public repositories such as GitHub, Pastebin, etc.) is strictly prohibited.**