UC Berkeley CS 168, Fall 2014

CS168 Project 3a

(Version 1.3)

Due: 11:59:59 am (at noon), November 17th, 2014 (hard deadline)

Chang Lan Shoumik Palkar Sangjin Han

Overview

In this project, you will implement a basic firewall running at end hosts. A firewall is a "*security system that controls the incoming and outgoing network traffic by analyzing the data packets and determining whether they should be allowed through or not, based on a rule set*" [Wikipedia]. Unlike the previous projects of this course, where you worked in simulated environments, you will deal with real packets in a Linux-based virtual machine (VM) for this project.

This month-long project is divided into two parts, and each part has it own submission deadline. In the first part (3a), which is covered in this document, you are asked to build a stateless firewall on top of the given framework. In the second part (3b), you will be extending the functionality of your firewall to support stateful rules at the application layer. Note that your solution for Project 3a will also be used as a base for Project 3b. It is very important to keep your code readable and extensible.

Your task for Project 3a is to implement a firewall that filters out packets based on simple firewall (Protocol/IP/Port and DNS query) rules on a packet-by-packet basis. Upon successful completion of this project, you will be able to:

- Understand the basic functionalities of a firewall.
- Be familiar with the details of TCP/IP packet formats.
- Explore low-level packet processing.
- Utilize various tools for network testing.

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.

Changelog

v1.3 (11/5/2014)

- Removed anything that mentions timers.

v1.2 (10/30/2014)

- Changed the VM image link to http://bit.ly/cs168proj3

v1.1 (10/29/2014)

- Changed due time description in order to avoid confusion

v1.0 (10/27/2014)

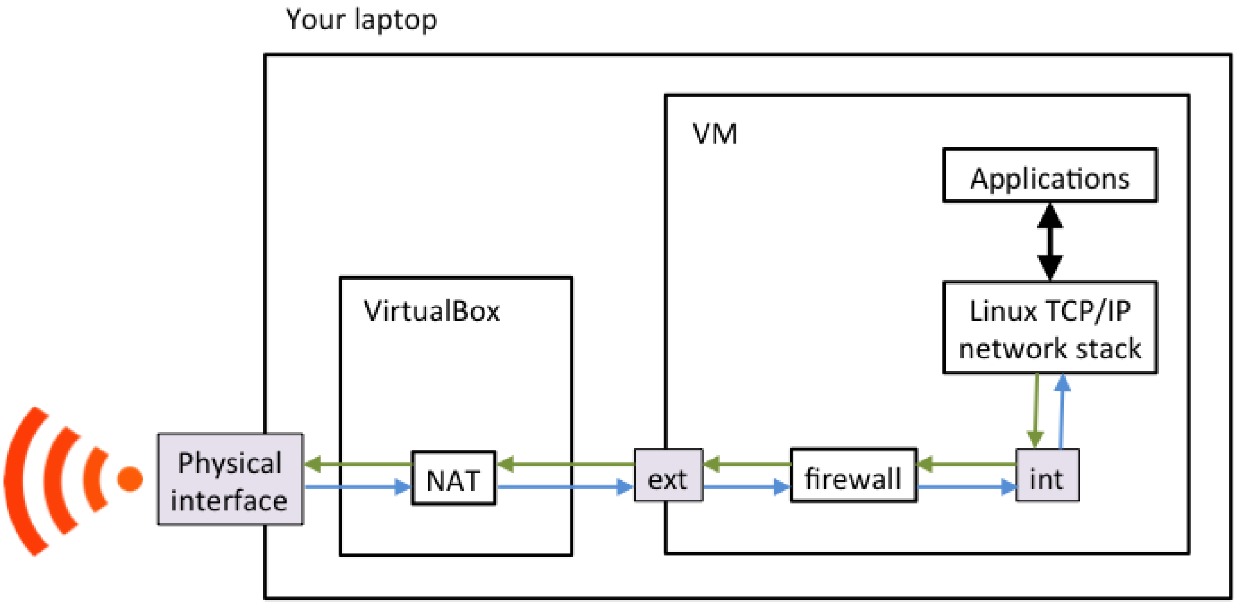
- First release

VM Setup

A personal firewall must have the ability to intercept incoming and outgoing network packets from network interfaces. Since this operation is security-critical and dependent on your operating system, we provide a VM image that is preconfigured to be readily used for the project. To run the VM, you will need to use your personal x86 laptop/desktop, rather than instructional machines. The VM runs on the Ubuntu Linux 14.04 Desktop edition.

Understanding the VM Network Configuration

The following figure illustrates how the VM network is configured.



The arrows in the diagram represent the flow of network traffic (packets). In the VM, there are two network interfaces, namely `ext` and `int` (short for "external" and "internal", respectively), and the firewall in between, as a bump-in-the-wire. All outgoing packets of the VM will be sent through the `int` interface. Similarly, all incoming packets will be received via the `ext` interface. Your firewall should inspect packets received from one interface and selectively pass them to the other interface. For example, when the firewall receives an incoming packet from `ext`, it determines whether to pass the packet through based on the firewall rules. If so, the firewall transmits the packet onto `int`, to be processed by the Linux TCP/IP stack. Similar things happen for outgoing packets as well.

When the firewall is not running, nothing relays packets between `ext` and `int`. **Thus the VM has no access to the outside network by default**. Don't be surprised if you cannot access to any websites with Firefox in the VM. Once you have correctly implemented your firewall, or after running "`sudo ./main.py --mode bypass`" (explained later), your applications in the VM will have access to the Internet.

Note: Due to some internal issues, **you will not see any packets on the `int` interface with tcpdump/wireshark, while the firewall is not running.**

For simplicity, we made the following design decisions for the project:

- `ext` and `int` are Ethernet interfaces, which carry Ethernet frames (with 14-byte Ethernet header followed by its payload, mostly an IP packet). However, your firewall will only need to care about IP packets; it receives and sends IP packets, and all Ethernet-related operations will be handled by the code we have provided.
- IPv6 was completely disabled in the VM. All you see in the firewall is IPv4 packets.
- Your firewall will not receive any fragmented IP packets.

Another thing worth mentioning is that there is a NAT[1] module in VirtualBox, which connects the VM to the outside network. While the details of NAT is out of the scope of this project, there are two things to remember:

- The IP address of the VM (statically set to 10.0.2.15) will not be seen from the outside network. This is because the NAT module translates it into the IP address of the host (your laptop) for every outgoing packet, and the other way around for every incoming packet.
- By default, the NAT module will not allow incoming connections destined for the VM (e.g., the NAT module will not forward incoming TCP SYN packets). In other words, you cannot run network server applications in the VM.

VM Installation

1. Download and install the latest version (**4.3 is recommended**) of VirtualBox, from https://www.virtualbox.org/wiki/Downloads
2. Download the VM image, from http://bit.ly/cs168proj3
   a. It may take a while to download the image, due to its large size (> 2.2 GB).
   b. Make sure you have enough disk space, as the compressed image gets bigger when it is imported. We recommend securing at least 10 GB of free space.
3. Launch VirtualBox, and import the VM.
   a. Select "File → Import Appliance"
   b. Click the "Open appliance..." button
   c. Choose "cs168proj3.ova"
4. Once imported, select "cs168proj3" on the left panel of the main window, and click the "Start" button to launch the VM.

Account

[1] Network Address Translation: http://en.wikipedia.org/wiki/Network_address_translation

The username is "`cs168`" and its password is "`F1rewa!!`" (number one instead of letter I and exclamation mark instead of letter L), without quotes. Feel free to change the password if your left hand fingers get tired.

Many applications (including the firewall itself) you will run on the VM need root permission, as they need to perform privileged network operations. Your account is granted sudo privileges without the password.

(Optional) Setting Host-Only Interface

You can skip this configuration if you do not mind working in the Ubuntu desktop environment.

Some of you may want to connect to the VM via SSH to work on the project, rather than directly using the GUI of the VM. For the connection between the host and the VM, you need to follow the procedure (based on Mac OS X, but it should be similar on other operating systems) below.

1. Turn off the VM.
2. Add a host-only interface for the host
    a. Select "VirtualBox → Preferences" in the menu.
    b. Choose the "Network" tab.
    c. Choose the "Host-only Networks" tab.
    d. Click the "+" button on the right.
    e. Click the screwdriver button on the right.
    f. On the "Adaptor" tab,
        i. Set the "IPv4 Address" to 172.16.122.1
        ii. Set the "IPv4 Network Mask" to 255.255.255.0
    g. On the "DHCP" tab,
        i. Unmark the "Enable Server" checkbox.
3. Add a host-only interface for the VM
    a. Select the "cs168proj3" VM on the left panel of the VirtualBox Manager window.
    b. Click the "Settings" button.
    c. Choose the "Network" tab.
    d. Choose the "Adaptor 2" tab.
    e. Mark the "Enable Network Adaptor" checkbox.
        i. Attached to: Host-only Adapter
        ii. Name: vboxnet0 (or anything else you created above)
    f. Click on the "Advanced"
    g. Set the "Adapter Type" to "PCnet-FAST III"
    h. Set the "MAC Address" to 080027107f8d (in hex)
    i. Mark the "Cable Connected" checkbox.
4. Check if everything is OK.
    a. Launch the VM
    b. Open a terminal window.

c. Run the command “`ifconfig`”.
d. You should be able to see the “`host`” interface.

The IP address of the host-only interface in the VM is 172.16.122.2. In the host (your laptop), you can make a SSH connection to the VM, with “`ssh cs168@172.16.122.2`” (on Linux or MAC) or your favorite SSH client (on Windows).

Network connections over this dedicated interface will not be affected by the firewall. All outgoing packets from the VM with a destination IP address in 172.16.122.0/24 will be sent through the `host` interface, instead of the `int` interface.

Project Specification

Provided Files

All files needed to do the project reside in the `/home/cs168/proj3` directory. Your task is to implement the project specification in the `firewall.py` file. Remember that this file is the only source code you submit for the project. All your modification must be done only in this file.

The `/home/cs168/backup_proj3` directory also contains the same files, just in case you need the original version.

main.py

This is the main executable file for your firewall. Modify this file only for debugging purposes. It contains some low-level code to intercept/inject packets from/to Linux network interfaces. Because of this, you will need root privileges to run this script.

```
sudo ./main.py [--mode <module name>] [--rule <rule file name>] [--opt1 arg1] ...
```

There are two predefined options:

- `--mode`: It specifies a Python module name that implements `Firewall` class. The default argument is “`firewall`”, which will run `firewall.py`.
- `--rule`: It specifies a rules file name. The default argument is “`rules.conf`”.

All other command-line options will be parsed and stored in the `config` dictionary, which is given to the constructor of `Firewall` class.

bypass.py

This is a dummy example that implements the bypass mode. In this mode, all incoming/outgoing packets will be passed regardless of the firewall rules. This mode can be useful when you want the VM to communicate without interference from the firewall, such as:

- Installing new applications
- Analyzing how network protocols work in normal conditions with tcpdump/wireshark
- Copying your source code to outside the VM for final submission
- … so on.

The following command will run in the bypass mode:

```
sudo ./main.py --mode bypass
```

The Firewall class implemented in this file will give you some basic ideas on how to implement your own firewall in firewall.py, such as how to pass packets with the .send_ip_packet() method.

firewall.py

This is the file containing the skeleton for the code you need to implement. This module currently does not do anything, and all packets between the int and ext interfaces will be dropped. Your task is to complete the Firewall class in the file so that packets can be filtered out based on the firewall rules file. The skeleton of the class looks as follows.

```
class Firewall:
    def __init__(self, config, iface_int, iface_ext):
        self.iface_int = iface_int
        self.iface_ext = iface_ext
        ...

    def handle_packet(self, pkt_dir, pkt):
        pass
```

__init__():

- You should load the rules file (the filename is given in config['rule']).
- Also read geoipdb.txt here.

handle_packet():

- Whenever a packet is captured, this handler will be invoked.
- pkt_dir indicates the direction of the packet. It can be either of the following two values:
    - PKT_DIR_INCOMING: The packet has been received from the ext interface. You should call self.iface_int.send_ip_packet() to pass this packet.
    - PKT_DIR_OUTGOING: The packet has been received from the int interface. You should call self.iface_ext.send_ip_packet() to pass this packet.
- pkt is a Python string that contains the actual IP packet, including the IPv4 header.
- To drop the packet, simply omit the call to .send_ip_packet().

Rules File Format

A firewall rule describes a type of packets that the firewall should pass/drop. A rules file contains firewall rules, each of which is written in a single line. `rules.conf` will be used by default, unless specified otherwise in the command line. The rules file decides whether to drop a packet or not, when the packet content/header matches one the defined rules. If multiple rules are matched, use the last one. There are two type of firewall rules: Protocol/IP/Port rules and DNS rules.

Note that the content of the rules file can be arbitrary. The provided `rules.conf` file in the `proj3` directory is just an example, and various rules files will be used to grade your solution. Your firewall should work correctly with any rule files that conform to the following format.

Protocol/IP/Port Rules

You apply Protocol/IP/Port rules for every packet that `Firewall.handle_packet()` takes. All protocol, external IP address, and external port fields must match to apply the verdict. "External" means "outside", thus it may represent either the source or destination IP address/port, depending on the packet direction. For example, if an incoming packet has a UDP/IP header 8.8.4.4:53 → 10.0.2.15:32154, the external IP is 8.8.4.4 and the external port is 53.

Format: <verdict> <protocol> <external IP address> <external port>

Field	**Possible values/formats**	**Description**
verdict	1. "`pass`" 2. "`drop`"	● `pass` means that a matched packet should be handed over to the interface on the other side, with the `send_ip_frame()` method (e.g., to `int` if received from `ext`). ● `drop` discards the packet
protocol	1. "`tcp`" 2. "`udp`" 3. "`icmp`"	● This field examines the Protocol field in the IPv4 header of packets.
external IP address	1. "`any`" 2. a 2-byte country code 3. a single IP address (e.g., `128.32.244.172`) 4. an IP prefix (e.g., `123.34.128.0/17`)	● For country codes, see the "GeoIP DB" section. ● `any` is identical to `0.0.0.0/0` ● `1.2.3.4` is identical to `1.2.3.4/32`
external port	1. "`any`" 2. a single value 3. a range (e.g., `2000-3000`)	● It specifies TCP/UDP port numbers. For ICMP packets, it is for the ICMP Type field. ● The range is inclusive (i.e., `2000-3000` includes 2000, 2001, …, and 3000)

DNS Rules

Format: <verdict> `dns` <domain name>

- verdict: “`pass`” or “`drop`”
- domain name: e.g., “bar.foo.com” (full domain name) or “*.gov” (wildcard domain name)
    - A full domain name is for **exact match**.
    - “`*.gov`” matches not only “`fda.gov`” but also “`www.fda.gov`”.
    - “`*.foo.com`” does not match “`foo.com`”
    - The asterisk (“*”) can only be used at the leftmost DNS label, alone.
        - Good syntax: “*”, “`*.net`”, “`*.google.com`”, ...
        - Bad syntax: “`mail.*.com`”, “`*.foo.*.org`”, “`www*.salary.com`”, ...

You apply DNS rules only for DNS query packets. More specifically, the packet should satisfy all of the following conditions to be considered for DNS rule matching.

- It is an outgoing UDP packet with destination port 53.
- It has exactly one DNS question entry.
    - There may be other non-empty sections (Answer, Authority, and Additional)
- The query type of the entry is either A or AAAA (QTYPE == 1 or QTYPE == 28), and
- The class of the entry is Internet (QCLASS == 1).

Example

Suppose that Starbucks wants to provide free WiFi, but with very restrictive rules as follows.

```
% allow “ping”, but no other types of ICMP packets
drop icmp any any
pass icmp any 0
pass icmp any 8

% allow DNS packets only to Google DNS servers
drop udp any any
pass udp 8.8.8.8 53
pass udp 8.8.4.4 53

% allow only HTTP(80), HTTPS(443), and Skype(1024-65535)
drop tcp any any
pass tcp any 80
pass tcp any 443
pass tcp any 1024-65535

% punish Italy (for not having Starbucks) and MIT (for the greedy /8 address block)
drop tcp it any
drop tcp 18.0.0.0/8 any

% ahem…
drop dns peets.com
drop dns *.peets.com
```

The rules implemented above do not allow any TCP/UDP/ICMP packets by default, but with some explicit exceptions.

Performance

Performance is one of the most important aspects of a firewall. Typically, commercial firewalls can process more than millions of packets per second. For this project, however, we will focus on its functionality and correctness, rather than performance. After all, the firewall is software-based, runs in a virtualized environment, and will be implemented in Python; it would be impossible to match the performance of commercial firewalls.

However, there will be minimum performance requirements. **If your firewall implementation underperforms by a factor of five when compared to our reference implementation with the same rule set, you may lose some points**. The performance will be measured in the number of processed packets, for a certain period of time (> 20 seconds, including startup time). Our reference implementation does not incorporate any sophisticated optimizations, so you do not need to worry much unless your implementation is very inefficient.

Notes

- Rule matching:
    - **If none of the rules match, just pass the packet.**
        - Thus you should always pass non-TCP/UDP/ICMP packets.
    - **If multiple rules match, apply the last one's verdict in the rules file.**
    - DNS request packets are inspected not only by DNS rules, but also by Protocol/IP/Port rules.
    - DNS rules may appear before Protocol/IP/Port rules.
    - For this project, we will not have more than 30 rules in the file, so linear scanning over the rules for every packet is perfectly fine.
    - The wildcard domain name matching described here does not conform to the standard (RFC 4592).
- Analyzing packets
    - `handle_packet()` will be called only for non-fragmented IPv4 packets.
    - Do not worry about packets with wrong TCP/IP checksum.
    - If a packet is so malformed that you cannot decode the packet for rule matching, simply drop the packet.
        - Also, if a DNS query packet (UDP with destination port 53) is too malformed to decode, you should drop the packet, even if there is no DNS rules.
    - If a DNS packet has non-question entries, you should consider the packet for DNS rules, as long as QDCOUNT==1 and the question entry conforms to the DNS rule matching conditions.
        - For example, `dig` will include one "Additional" entry in its DNS packets, but your firewall should apply DNS rules for those packets.
    - Your program should not crash.
    - Watch out for endianness. Most network protocols follow network order.
        - http://en.wikipedia.org/wiki/Endianness#Endianness_in_networking
- Parsing the rules file:
    - **All rules are case-insensitive, including domain names and country codes**. For example, "`tcp`", "`TCP`", and "`tCp`" must be all allowed.
    - You can assume that the syntax/format of the file is always correct. For example, we will not trick you with `128.50.132.20/24`, which has bad IP prefix syntax.
    - Ignore empty lines and comment lines (lines beginning with "%")

GeoIP DB

In the `proj3` directory, we provide the database file `geoipdb.txt` that has geolocation-mapping information of the IP address space. You should use this file to implement the country-based blocking in your firewall. In the file, each line represents an IPv4 address range and its corresponding country code, in the following format.

Format

<start IP address> <end IP address> <2-character country code>

- start IP address, end IP address: These are dot-separated IPv4 addresses. The given IP address range is "inclusive" in that it includes both start and end IP addresses.
- 2-character country code:
    - ISO standard country codes: http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2
    - There are also non-standard country codes: A1, A2, EU, AP.
    - We use the same country codes for the rules file (again, case-insensitive!)

The database file should be loaded into memory in `Firewall.__init__()`.

Notes

1. Note that we may use a different version of the database file for grading. However, the following two assumptions will always hold:
    a. All IP address ranges given in the file do not overlap each other. Hence, no longest prefix matching will be required.
    b. All IP address ranges are sorted in ascending order in the file.
2. Some IP addresses may not match any records in the database. It is normal, so don't worry.
3. Your lookup code for country matching should be "reasonably" fast, to meet the minimum performance requirements described above.
    a. No linear search over the entire database or a specific country for every packet!
    b. Instead, consider using more efficient algorithms and data structures, such as binary search, radix tree, etc.
    c. If loading of the database takes more than a few seconds, you are probably not on the right track.
4. For this project, we assume that the database is always correct and up-to-date.

The original database was retrieved from here: http://dev.maxmind.com/geoip/legacy/csv/

References

The following documents provide the detailed protocol specifications to accomplish this project.

- IPv4 header:
  - http://en.wikipedia.org/wiki/IPv4
  - http://tools.ietf.org/html/rfc791
  - IP protocol numbers: http://tools.ietf.org/html/rfc790
- TCP header:
  - http://en.wikipedia.org/wiki/Transmission_Control_Protocol
  - http://tools.ietf.org/html/rfc793
- ICMP header:
  - http://tools.ietf.org/html/rfc792
- DNS packet format:
  - http://tools.ietf.org/html/rfc1035

You will (and should!) spend more time for testing than coding. We list some of the most helpful network testing tools below, all of which are preinstalled in the VM. There are a lot of online tutorials you can find on the Internet.

- `tcpdump` / `Wireshark`
  - `tcpdump` is a command-line packet sniffer that captures and display packets on networks. It also decodes packet headers for various protocols, which is very useful to verify the correctness of your own packet decoder.
  - `Wireshark` (formerly known as `Ethereal`) provides similar features with a graphical user interface.
  - They can also be used to check the behavior of your firewall. For example, if an outgoing packet is seen at both `int` and `ext`, it implies that the firewall successfully relayed the packet between those interfaces.
  - A short YouTube clip on Wireshark: http://youtu.be/6X5TwvGXHP0
- `nslookup` / `dig`
  - `nslookup` and `dig` are command-line tools for querying the Domain Name System.
  - You can use these tools to generate DNS query packets.
- `wget` / `curl`
  - `wget` and `curl` are handy tools to generate HTTP requests without using browsers.
- `nc`
  - `nc` (short for netcat) is a popular tool for generating TCP/UDP connections.
  - Refer to this page for more packet crafters: http://sectools.org/tag/packet-crafters/

We will release an introductory document for these protocols/tools at the course webpage.

Logistics

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 (so choose a partner wisely!).

By the nature of this project, it could be hard to "split" the work between teammates. Instead, consider pair programming (http://en.wikipedia.org/wiki/Pair_programming), if you choose to work in partners.

Submission Instructions

Turn in a `.tar` file with both you and your partner's last names in the file name (For example, `project3a-han-lan.tar` or `project3-palkar.tar`) on an instructional machine. Your tar file should include:

- `firewall.py`: **remember, this file should include all your code.**
- `readme.txt`
  - Your (and your partner's) full name and login name
  - (optional) Any comments/concerns on the project. This will not affect grading. We will collect your opinions anonymously to design future projects better.

```
$ tar -cf project3a-han-lan.tar firewall.py readme.txt
$ submit project3a
```

Regrade & Late Policy

Your regrade request must be made within seven days after score release. Submit your new tar file on an instructional machine, then email clan@eecs.berkeley.edu.

- final score = max{old score, new score * (regrading penalty)} * (your original late penalty)
  - regrading penalty = 0.9 - (code difference in tokens) * 0.001
    - We will provide a script that measures code difference.
  - Of course, regrading penalty doesn't apply if it turns out be auto-grader's fault.
- You have only one opportunity for regrade request.

The late policy is simple; no slip dates. If submission is late, it is penalized as follows:

- < 24 hours late, you lose 10%
- < 48 hours, 20%

- < 72 hours, 40%
- More than three days late, you can no longer hand-in the assignment.

Cheating

Simply put, DO NOT EVER TRY IT. We will do our best to protect our students from losing the value of their honestly earned grades due to cheaters. We may perform manual inspection and use automated tools (do not underestimate them) to detect potential plagiarism over all submissions.

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[2]. Apparently 23% of academic misconduct cases at a certain junior university are in Computer Science[3], but we expect you all to uphold high academic integrity and pride in doing your own work.

[2] http://sa.berkeley.edu/code-of-conduct

[3] 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.
- Backup is always a good idea.
- Feel free to surf the Web for **general ideas**, such as Python language/library references, network protocol specifications, testing tool usages, and debugging tips.
- You can discuss with anyone about algorithms, approaches, and concepts without details, but stay away from your computer/code while doing so.
- 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.**
- Do not share your code with anyone, other than your project partner.
- Do not post your code on any public repositories like GitHub and Pastebin.
- Do not post any project-specific questions on Internet forums other than Piazza.
- The project is led by Chang Lan (main), Shoumik Palkar, and Sangjin Han. Avoid asking other GSIs about project-specific questions.

UC Berkeley CS 168, Fall 2014

CS168 Project 3b

(Version 1.0)

Due: 11:59:59am (at noon), December 3rd, 2014 (Hard deadline)

Anurag Khandelwal Shoumik Palkar Sangjin Han

Overview

In this project, you will implement a basic firewall running at end hosts. A firewall is a "*security system that controls the incoming and outgoing network traffic by analyzing the data packets and determining whether they should be allowed through or not, based on a rule set*" [Wikipedia]. Unlike the previous projects of this course, where you worked in simulated environments, you will deal with real packets in a Linux-based virtual machine (VM) for this project.

Recall that in 3a, 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 3b, you will be extending your solution for 3a to make a stateful active application-layer firewall. You will use the same framework, VM, test harnesses, tools, and as in 3a. 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.

You will likely find the supplementary document from 3a useful for this part, as well.
http://www-inst.eecs.berkeley.edu/~cs168/fa14/projects/project3a/supplement.pdf

Good luck, and 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 (11/17/2014)

- First release

Requirements

Overview

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

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 3a. For `host name`, see below.

Firewall behavior from Project 3a 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 3b, we will neither test firewall rules defined in 3a (`pass`/`drop` rules) nor country-based matching,** so your grade for 3a will be mostly decoupled from your grade for 3b. If your solution for 3a was incomplete, don’t worry too much.

For the sake of simplicity, you can make the following assumptions:

- The rules file has always correct syntax, as defined in Project 3a.
- 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.

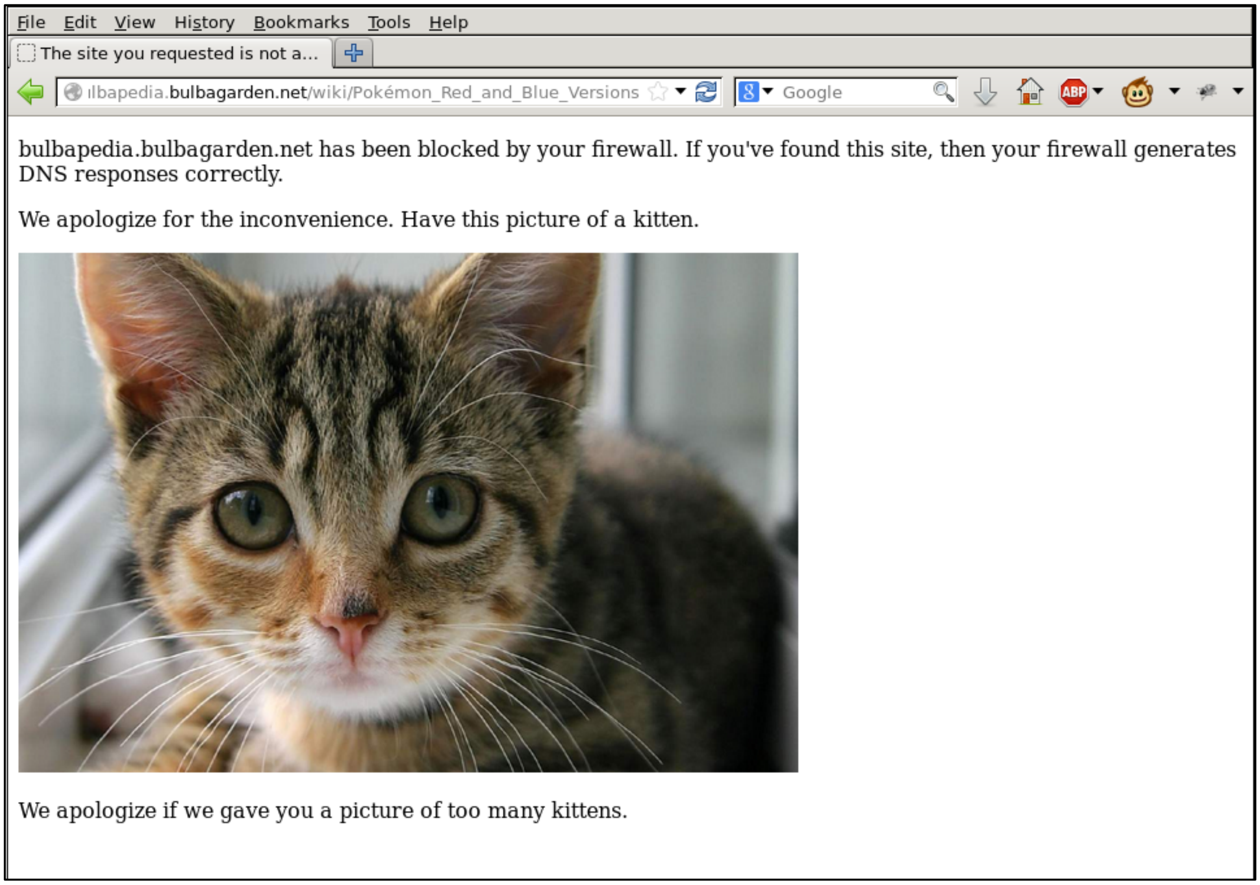
When generating the reset packet you must carefully compute both the TCP and IPv4 checksum; consult Wikipedia or the relevant RFCs for details. http://locklessinc.com/articles/tcp_checksum/ also has some C code (go through the `checksum1()` function) that might help you figure out how to implement these. 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 `54.173.224.150`. 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.


File
Edit
View
History
Bookmarks
Tools
Help
The site you requested is not a...
ılbapedia.bulbagarden.net/wiki/Pokémon_Red_and_Blue_Versions
Google
ABP

bulbapedia.bulbagarden.net has been blocked by your firewall. If you've found this site, then your firewall generates DNS responses correctly.

We apologize for the inconvenience. Have this picture of a kitten.

We apologize if we gave you a picture of too many kittens.

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

3. HTTP Log: `log http`

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**. 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., whitespace 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`). It will be mostly `GET`.
- `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).

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 3a. 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:

- Note that since we assume you are not running a web server on port 80, the external port for requests should be the destination port, and the external port for responses should be the source port.
- 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*, 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.
  - Also, dropping out-of-order packets should be done on a per-connection basis.
- Your implementation must not waste memory unnecessarily. For example, when you download a large file via HTTP, your firewall should not store the entire byte stream. You only need to buffer partial HTTP headers.
- Again, your firewall should not crash.

Examples

1. Consider “`log http *.berkeley.edu`”. Opening http://www-inst.eecs.berkeley.edu/~cs168/fa14 in Firefox will produce the following log entries, after clearing the browser cache:

```
www-inst.eecs.berkeley.edu GET /~cs168/fa14 HTTP/1.1 301 254
www-inst.eecs.berkeley.edu GET /~cs168/fa14/ HTTP/1.1 200 273
www-inst.eecs.berkeley.edu GET /~cs168/fa14/overview.html HTTP/1.1 200 2581
www-inst.eecs.berkeley.edu GET /~cs168/fa14/content.html HTTP/1.1 200 1569
www.eecs.berkeley.edu GET /Includes/EECS-images/eecslogo.gif HTTP/1.1 200 828
www-inst.eecs.berkeley.edu GET /~cs168/fa14/images/Keycard_A.png HTTP/1.1 200 324
www-inst.eecs.berkeley.edu GET /~cs168/fa14/images/Book.png HTTP/1.1 200 174
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

Collaboration Policy

The project is designed to be solved independently, but you may work with your partner from 3a, if you wish. **You may not work with someone new, unless both you and your new partner worked alone for 3a.** 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.

By the nature of this project, it could be hard to "split" the work between teammates. Instead, consider pair programming (http://en.wikipedia.org/wiki/Pair_programming), if you choose to work in partners.

Submission Instructions

Turn in a `.tar` file with both you and your partner's last names in the file name (For example, `project3b-khandelwal-palkar.tar` or `project3b-han.tar`) on an instructional machine. Your tar file should include:

- `firewall.py`: remember, this file should include all your code.
- `readme.txt`
  - Your (and your partner's) full name and login name.
  - (optional) Any comments/concerns on the project. This will not affect grading. We will collect your opinions anonymously to design future projects better.

```
$ tar --cf project3b-han.tar firewall.py readme.txt
$ submit project3b
```

Note: **We always use the latest submission.**

Regrade & Late Policy

Your regrade request must be made within seven days after score release. Submit your new tar file on an instructional machine, then email anuragk@eecs.berkeley.edu.

- final score = max{old score, new score * (regrading penalty)} * (your original late penalty)
  - regrading penalty = 0.9 - (code difference in bytes) * 0.001
    - We will provide a script that measures code difference.
  - Of course, regrading penalty doesn't apply if it turns out be auto-grader's fault.
- You have only one opportunity for regrade request.

The late policy is simple; no slip dates. If submission is late, it is penalized as follows:

- < 24 hours late, you lose 10%
- < 48 hours, 20%
- < 72 hours, 40%
- More than three days late, you can no longer hand-in the assignment.

Cheating

Simply put, DO NOT EVER TRY IT. We will do our best to protect our students from losing the value of their honestly earned grades due to cheaters. We may perform manual inspection and use automated tools (do not underestimate them) to detect potential plagiarism over all submissions.

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

DO’s and DON’Ts

DO’s

- 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.
- Backup is always a good idea.
- Feel free to surf the Web for **general ideas**, such as Python language/library references, network protocol specifications, testing tool usages, and debugging tips.
    - Admittedly, Google is faster and more accurate than GSIs in most cases.
      Also it works 24/7.
- You can discuss with anyone algorithms, approaches, and concepts without details, but stay away from your computer/code while doing so.
- 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 is strictly prohibited.**
- Do not share your code with anyone, other than your project partner.
- Do not post any project-specific questions on Internet forums other than Piazza.
- The project is led by Anurag Khandelwal (main), Shoumik Palkar, and Sangjin Han. Avoid asking other GSIs project-specific questions.