CSE 127: Computer Security (Fall 2018) **Due December 4, 10:00 P.M. PDT**

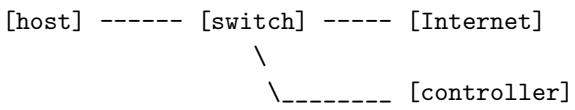
Assignment 5

100 pts

For this assignment, you will implement an in-path network attack similar to China's Great Cannon. The goal is to implement a man-in-the-middle attack that injects a specific iframe into certain targeted HTML pages. Unlike an arbitrarily-powerful in-path attacker, you will not be able to inject new packets, but you are able to decide whether to alter packets or allow them to pass through unmodified. You will write a Python function that inspects IP packets flying by on the wire and modifies them as it sees fit. Your solution is due on December 4, 2018 no later than 10:00 P.M. PDT. You may work with *one* other person in the class on this assignment; if you do so, you should only submit one solution for the two of you. You may not discuss your solution with anyone except your partner until seven days after the assignment deadline.

1 Overview

Here is the network topology:



Both the host and the switch are emulated by Mininet, a popular network emulator, in the VM we provide. Since the switch connects the host to the Internet, it will see all traffic from and to the host. The switch is controlled by a controller via OpenFlow protocol. That is, the switch will send every packet it receives to the controller and does whatever the controller tells it. The controller is written using POX, a Python framework for writing OpenFlow controllers.

In this project, you need to implement a function named `manipulate_packet` in class `Cannon` (it can be found at `/home/mininet/pox/pox/triton/cannon.py`), which is called by the controller. Specifically, the controller passes every IP packet it receives from the switch to this function. This function needs to decide if the packet should be allowed to pass through unmodified, modified, or dropped.

2 Requirements

You must implement a function named `manipulate_packet` in `/home/mininet/pox/pox/triton/cannon.py`. This function modifies certain HTTP replies to inject an iframe into the HTML of Web pages being loaded from the host. Which HTTP replies to modify and what iframe source to inject are provided in the `Cannon` class's constructor arguments:

1. `target_domain_re`: a compiled regular expression. This argument specifies the domains which are candidates for attack.
2. `url_path_re`: a compiled regular expression. This regex specifies the URL paths that are candidates for attack.
3. `iframe_url`: a string. If the Webpage load is subject to iframe injection, inject an iframe into its body tag with the src attribute equal to `iframe_url`.

The caller always passes an IP packet object (an instance of class `ipv4` in `pox.lib.packet.ipv4`) to `Cannon.manipulate_packet`, and it is supposed to either return an IP packet object (either a modified packet or the one passed in as an argument), or `None` (a Python None object). A None object tells the caller that the packet in question should be dropped.

If the Triton Cannon observes an HTTP response from a matching target domain with a matching URL path, then the cannon should inject an iframe as the last element before the `</body>` close tag. The iframe element source should take the following form:

```
<iframe src="%iframe_url%"></iframe>
```

where `%iframe_url%` is `IFRAME_URL` defined in `cannon.py`. Do not add any bytes before or after the opening and closing iframe tags. In other words, the length of the modified Webpage should increase only by 24 plus the length of the iframe URL's encoding, since 24 is the length of the string "`<iframe src=""></iframe>`".

Your code should not modify an HTTP reply if it does not belong to both the target domain and target URL path. If your code modifies a packet, its contents should only be based on the contents of packets observed up to that point; let's say you are processing packet n, you should make a decision about packet n based on previous packets (packet 1 to n). You cannot cache packet n first and then decide to modify it based on, say packet $n+1$. Also, you code should NOT serve as a proxy that downloads the requested Web page on its own (e.g., via `urllib`) and serves it back to the client, with or without modifications to the page.

Do *not* modify the name of the Cannon class or the signatures of the `Cannon.__init__` or `Cannon.manipulate_packet` methods. Other than that, feel free to modify `cannon.py` in any way you find useful, or add new files to the triton package (`/home/mininet/pox/pox/triton/`).

3 The Environment

3.1 POX

The controller used in this project is in `/home/mininet/pox/pox/forwarding/dummy.py`. When it sees an IP packet, it will pass it to `manipulate_packet`. Based on the return value of `manipulate_packet`, it may ask the switch to route the packet returned from `manipulate_packet`, or do nothing (drop the packet). Notice the regexes for the target domain and URL paths are defined in this file.

Since the code you will write is inside POX, and POX has many useful data structures and library functions (e.g parsing TCP), you may want to look at what POX can provide you for free.

3.2 Mininet

The network topology shown above above is emulated by Mininet. The network is defined in

`/home/mininet/mininet/examples/cannon_network.py`.

You should not change it. In `cannon_network.py`, the host is `h1`, the switch is `s1`, and the controller is `c0`. After you start the network emulation, a Mininet command line interface (CLI) will also show up. You can run commands Mininet provides. Here are just a few examples:

- `net`: show the network topology.
- `h1 ping s1`: The host pings the switch.

You can find links to related materials in Section 7. You can even run bash commands in emulated nodes in the network.

- `h1 su mininet -c "wget www.ucsd.edu"`: Fetch / from www.ucsd.edu in the host.

- `h1 su mininet -c chromium-browser`: Run chromium in the host.
- `h1 wireshark &`: Run Wireshark in the host.

3.3 VM

You can download the VirtualBox VM image configured for this assignment from:

https://drive.google.com/file/d/1Ze8EqoYdFT5fsk3Cno00KRr3RIXHqDAo/view?usp=sharing

The username and password to the VM are "mininet".

The VM is configured with SSH on port 2222. A simple GUI called LXDE is also installed on the machine, which you may find useful for running Wireshark, for example. You can run `startx` to launch it.

The files needed to begin the assignment are already on the VM. However, if you want to copy files from the VM to your computer, use the following command from your host computer (with a scp program installed):

```
scp -P 2222 mininet@127.0.0.1:/path/to/files/ /destination/path
```

3.4 Getting Started

After logging in to the VM, first run `startx` to start the LXDE GUI, which lets you use tools like Wireshark as well as open multiple terminal windows.

To start the emulated network, you need to first start the controller by running:

```
/home/mininet/pox/pox.py forwarding.dummy
```

Then you need to start Mininet by running the following command in another terminal

```
sudo /home/mininet/mininet/examples/cannon_network.py
```

Now the network is running. You can verify it by running the following command in Mininet's CLI.

```
h1 ping -c 3 www.google.com
```

If the host can ping Google, then the network is working.

4 Solution Format

You solution should consist of the following files:

- `cannon.py` with an implemented `Cannon` class
- A filled out `PID` file, with one partner's SID on line 1, and if applicable, followed by a space and then by the other partner's SID (all on the same line).
- A writeup (`writeup.txt`) explaining how your code works in detail and any interesting challenges you faced/overcame.

5 Submitting Your Solution

To submit, simply run the `hw5-turnin.sh` script from the `/home/mininet/pox/pox/triton/` directory. Make sure the `PID` file is correctly filled out, and all your necessary files are in that directory.

6 Hints

Wireshark is your friend here; learn how to use it to inspect traffic. Also, take advantage of POX's APIs. Additionally, be mindful of the length and checksum fields when modifying a packet. Finally, be aware that some websites use various compression schemes on their HTTP responses, so you need to find a way to avoid this.

When testing, take care to type the URL into the browser exactly the same as the target domain and URL path regexes (notice the extra slash in `blink.ucsd.edu/technology/security/` vs. `blink.ucsd.edu/technology/security`). Also, when possible, use Incognito Mode to test your function. For various reasons (caching etc.), the browser might not always send the HTTP request to the server, but Incognito Mode prevents this.

7 Reference material

The protocol specifications for IP, UDP, TCP, DNS, and HTTP are all maintained by the Internet Engineering Task Force (IETF) as "requests for comments" (RFCs). Some RFCs that you may find useful to consult:

- A TCP/IP Tutorial, RFC 1180
- Internet Protocol, RFC 791
- User Datagram Protocol, RFC 768
- Transmission Control Protocol, RFC 793
- Domain NamesConcepts and Facilities, RFC 1034
- Domain NamesImplementation and Specification, RFC 1035
- Hypertext Transfer ProtocolHTTP/1.1, RFC 2616 (Note: obsoleted by the more detailed RFC 723035 suite)
- Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, RFC 7230
- Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, RFC 7231

Note that RFC 791 and RFC 793 date to September, 1981, and RFC 768 to August, 1980!

For POX:

- The POX wiki page, with very useful documentation
- The POX github repository

For Mininet:

- The Mininet Walkthrough. You will need to understand only at the part bewtween Interact with Hosts and Switches and Part 2: Advanced Startup Options
- The Mininet FAQ.

8 Thanks

This project was originally developed for CSE 127 by Zhaomo Yang (using Mininet, OpenFlow, and POX), based on an earlier CSE 127 project by Wilson Lian (using iptables, NFQUEUE, and Scapy), which was in turn based on the network security project from the 2006 iteration of Stanford's CS 155 (using the Virtual Network System).

9 Frequently Asked Questions

The HTTP spec has lots of options! What sort of server responses do we need to handle?
You need to handle responses encoded using the `Content-Length` header.

What about chunked encoding? Isn't that very popular? Shouldn't we handle that, too?
It sure is! In previous iterations of this course, handling chunked encoding was a requirement. But it turns out that there are lots of corner cases to consider for robust handling of chunked encoding at the IP layer, and it was a pain for everyone. Since CSE 127 is a security class, not a networking class, we've decided to let chunked encoding go.

Do we need to be able to modify an HTTP response that has been compressed? That seems hard!
That would indeed be hard. You do not need to.

All the servers we test seem to compress responses. How can we modify any responses, or even test our code?
Servers compress responses because HTTP clients tell them they can. If you modify the HTTP `Accept-Encoding` request header to list only "`identity`", you will make the server believe that the client can't handle compression.

Okay, what about HTTP pipelining?
Nope, don't need to worry about that.

HTTPS? HTTP/2? SPDY?
Nope, nope, and nope.

HTTP/1.0?
Again, no; just HTTP/1.1. In particular, this means that you can assume that the `Host` header will be present in the client's request.

Do we need to do different things depending on the request verb? the response status code?
No. So long as the request matches the regular expressions you are given and the response has a body, you should modify the response. Note that some requests, such as GET with If-Modified-Since and HEAD, will generate responses with an empty body.

Can we assume that Web servers will always run on TCP port 80?
Web servers may run on any port, but for the purpose of this assignment you may assume that any target Web server runs on port 80, and that the `target_domain_re` regular expression does not specify a port.

Do we need to deal with out-of-order transmission of packets?
Yes, within bounds. Your code shouldn't assume that packets are always delivered in exactly sequence order. However, you don't need to handle a really unfortunate case like seeing the server headers after the body you would have needed to modify.

Can we assume that there will be just a single HTTP request outstanding at any time, or do we need to handle TCP packet interleaving between multiple HTTP requests?
You will need to deal with multiple simultaneous HTTP requests, even multiple simultaneous requests that match the given regular expressions and therefore need modifying. Note that you should be able to distinguish distinct requests by TCP four-tuple.

Do we need to handle TCP segmentation?
Yes, in the sense that you absolutely should expect that the server's response and probably the client's request will be too long to fit into a single IP packet, and that therefore TCP will generate multiple packets.

Do we need to handle IP fragmentation?

No, you do not need to consider IP fragmentation. In particular, we will set things up so that you can increase the size of a packet by 100 bytes or so without exceeding the MTU.

I am seeing a lot of IP fragmentation!

You shouldn't be. It is possible that the MTU on the interface connecting your VM to your host was not set correctly. Try running the following command and see if it fixes things: `VBoxManage modifyvm hw8vm --natsettings1 1300,0,0,0,0`

Can we hardcode a single client IP address?

No, your router will need to handle and modify requests from multiple client IPs.

Do we need to handle TCP four-tuple reuse?

For the purposes of this assignment, you may assume that a TCP four-tuple is never reused for a second TCP connection.

Is it okay if wget used as a client truncates the resource we modify, or hangs waiting for more content after all of it has been transmitted?

No, this likely indicates that your TCP sequence number manipulation isn't quite right.

Is it okay if, when our router modifies packets, it causes either the client or server to retransmit?

No, your modifications must be transparent to both client and server.

What can we assume about the HTML in the body of the response we modify? Is it well-formed?

Do not assume that HTML is well-formed. For the purpose of this assignment, though, it's okay to assume that the string `</body>` won't occur anywhere in the response except where you need to make your modification. (Even though it could, in general, also occur in a comment, in JavaScript, etc.)

Can the tag we are looking for be split between two TCP segments?

You may also assume that the `</body>` tag won't be split between TCP segments, though you shouldn't assume this about any other tag.

Can we install additional Python libraries and use them in our solution?

No, you should use only the libraries that are already installed on the VM.