

# CS166 Project 1: SynCity

Due on Thursday, February, 26 2009 at 11:59 pm EST

Project Out: February 10, 2009

## Prologue

It was a dark, rainy night. She walked into your office with the strut of a runway model and the eyes of a scared teenager, wet from the streets of Syn City<sup>1</sup> and straight into your heart. You knew from the start that it could only mean trouble, but then again, trouble was your business.

"*[insert login here]*, I need your help," she crooned in a voice softer than silk and more pathetic than bubblesort.

It was a messy affair, just like it always is. Two months later, she was gone. Sometimes, even when you know the only things you can trust are yourself, your drink, your revolver, and your NetApp filer, it's hard to keep a clear head. She left not only with your heart, but also with the knowledge of your personal server—gateway to the only thing more important than your heart: your cs166 homework.

You woke up on your office floor, surrounded in empty bottles and cigarette butts from your week-long bender, and you got yourself together. She'd been portscanning your server; there's no friend or lover who won't put a knife in your ribs for that extra dollar. It was only a matter of time before she wormed her way in. You wanted to make your box disappear, just like she did. So you turned the firewall up to the max, and watched the packets drop like so many Syn City mobsters.

But you needed a way to get in yourself, without letting anyone know your server was still up...

## Port Knocking: An Overview

From Wikipedia:

Port knocking is a method of externally opening ports on a firewall by generating a connection attempt on a set of prespecified closed ports. Once a correct sequence of connection attempts is received the firewall rules are dynamically modified to allow the host which sent the connection attempts to connect over specified port(s).

In essence, a server can have all of its ports drop incoming connection requests, thus effectively hiding its presence as a server. However, a port knocking daemon on the server is listening to incoming packets, keeping track of the ports requested by clients. If a client makes the correct

---

<sup>1</sup>Formerly known as Providence.

sequence of connection attempts within a specified timeout period, the server dynamically modifies the firewall rules to allow incoming connections for the client on a certain port or ports. This adds an extra layer of login authentication, albeit similar to requiring a plaintext password transmission that is common to all users. Security through obscurity isn't a great practice if it's your only security, but it can be a good complement to an existing system (eg. SSH).

## Your Task

Your task is as follows:

1. Write a port-knocking daemon that will run on your server, opening SSH once the correct knock sequence has been entered by a client.
2. Write a client wrapper for SSH that will allow you to easily connect to your protected server.
3. Briefly analyze the security of port-knocking by discussing possible attacks.

## The Server

Your server vm will initially have its firewall configured to drop incoming connection requests on all ports, accepting packets only from localhost connections and from already-established connections. You will write a port-knocking daemon which will listen for a five-connection-attempt sequence of your choice, and open port 22 (SSH) to incoming requests from clients who complete it correctly. It's a pretty basic principle, but there are some subtleties:

- The connection attempt must be initiated and completed within a specified timeout period, which can be set based on the expected network latency and congestion (for example, for a LAN, one second might be about the right timeout while for the internet, 30 seconds would be more reasonable). If a client fails to complete the sequence in the allotted time, its progress should be reset.
- If a client repeatedly fails to complete the sequence correctly (say, after three failed attempts) it should be "blacklisted" for some period of time. This is to prevent attackers from brute-forcing the sequence (though they would have a hard time of that, at  $65535^5$  combinations). Five seconds is probably enough for this project; in real life, a longer time might be preferable, but it's not as convenient for testing. You may want to consider a more permanent blocking for a client who knocks incorrectly too many times.
- Once a client knocks correctly, the firewall should be opened for on port 22 for their IP. However, it shouldn't be open to them forever, and it should be closed once they establish a connection. You want to close the port after five seconds or a successful connection, whichever comes first.

## The Client

For the client, you'll want to write a knocking program that performs the port knock, returning once port 22 has been opened for connection on the server. You could then use that as part of a script that connects via SSH, or perhaps just run SSH really quickly after running your knocking program.

Note that you *should* do your connection testing from your client VM rather than from the server to itself, as packets traveling within a single machine do not travel over the ethernet interface.

## Tools of the Trade

You'll be using two VMs for this project—one for the client and one for the server. For this, you should make two copies of `guidebian`. It's had the stuff you'll need for this assignment added. Copy them both into different directories and choose to create new VM identifiers for them when you open them (you should always do this when copying a new VM). The VM is located at

```
/course/cs166/asgn/syncity/vms/guidebian2
```

You guys only have access to VMPlayer which is plenty for our purposes. Launch it by accessing an I-Lab machine, either with `ssh` or in person, and typing `'vmplayer &'` at a console. Then you have to copy it to your `.project` folder and open it. Once in the VM, remember to save all your work to `/mnt/hgfs/TransferWork/`.

This points to `/home/svelagap/` by default but you should change it to be your `/u/username.project` folder, otherwise your files won't be written (hopefully you don't have write access to my folder)! Go to VMWare Player → Shared Folders → Properties to change the properties of the shared folder between the VM and the host.

Log in with username `root` and the password `"i<3166"`. You have access to a command line via `xterm`, where you can launch Eclipse for coding purposes by typing `"eclipse &"` in an `xterm` window.

One thing to remember is that the new `guidebian` has its firewall set to have all ports drop by default. This will happen on both the client and server you copy from here. In order to reset the firewall settings, or turn the firewall off, we've provided two handy scripts:

- `fwflush` disables the firewall
- `fwprep` resets the firewall rules to our initial settings described above

## Pcap, jpcap

Pcap is a packet capturing and monitoring API for Linux. The server VM will have `jpcap`, a Java interface to Pcap, installed. We have also installed the Perl, Python, Ruby, and C/C++ language and standard library distributions and the corresponding Pcap bindings.

You'll want to look at the javadocs for `jpcap`, but here's a basic rundown of what it takes to get going:

- **Start the Capture Engine:** You do this by instantiating a `PacketCapture` object.
- **Find and Open the Device:** First, use your instance of `PacketCapture` to `PacketCapture.findDevice()`. Then use `PacketCapture.open(device, false)`. The second parameter decides whether we want to operate in promiscuous mode or not. As we are not harlots, and also not interested in packets not destined for this machine, we do not need to be in promiscuous mode. Note that on machines with more than one network device, we'd need to list them all and pick the right one, but our VMs are set up with only one, so `findDevice` suffices.

- **Set a Filter:** You might not want to see every packet that our system receives (or you might). If you don't, you can use a filter string to specify which kinds of packets to capture. The format used is the same used by tcpdump, so `man tcpdump` for a fuller discourse on this (see the "expression" subsection). The jpcap command is `PacketCapture.setFilter(filter, true)`. The boolean here says to turn on optimization. As for filter, a good starting point might just be "tcp"—capture all TCP packets. You can also get it to do many more complex things, though, such as filtering based on host, net, port, etc. See the man page for more information.
- **Add a PacketListener:** The class you add here will get its `packetArrived` method called whenever a new packet arrives. You'll use `PacketCapture.addPacketListener(new YourPacketHandlerClass())`. *Note that your packet handler must implement the `PacketListener` interface!*
- **Write Your Packet Handler:** This goes with the step above. The key (and only) method to implement from `PacketListener` is `packetArrived(Packet packet)`. Note that while it is passed a generic `Packet`, if you filter for only TCP packets, and you should, you can safely cast this to a `TCPpacket`, which has a lot more useful information.
- **Start Capturing:** Invoke the capture method on your `PacketCapture` object.

Jpcap Javadocs: <http://jpcap.sourceforge.net/javadoc/index.html>.

## iptables

Iptables is the tool you'll be using to modify the netfilter rules of your server. It has a relatively simple command-line interface, which can be accessed from Java by running `Runtime.exec()` calls. Here's a basic formula for the type of iptables commands we'll be running:

```
iptables [command] [match] [target]
```

Command is the type of change we want to make to our rule set. In order to append a rule to the filter table, use iptables with the `-A` flag. Correspondingly, in order to delete a rule, use the `-D` flag. We need to specify that we're working on incoming packets, so we also add `INPUT` after this. ex:

```
iptables -A INPUT [match] [target]
```

Next, we need to figure out which packets we're matching. Iptables has a very comprehensive set of match conditions. A few you might be interested in are:

- Protocol: `-p [protocol]` to specify packets from a specific protocol. Example: `-p tcp`
- Destination Port: `--dport [port]`
- Source IP: `-s [ip]`

Example:

```
iptables -A INPUT -p tcp --dport 80 [target]
```

Finally, we need to choose a target. Mostly, we'll want to accept, drop, or reject packets. To do this, we'll use `-j [action]`, where [action] can be ACCEPT, DROP, or REJECT.

Example:

```
iptables -A INPUT -p tcp --dport 80 -j DROP
```

The above example would drop all incoming tcp packets to port 80 (HTTP).

This should be enough to get you up and running around firewall-land, but if you want more information, <http://www.faqs.org/docs/iptables/> has an excellent FAQ on iptables with examples. Chapter 6 is particularly relevant to this project. The iptables man page is also a useful but extensive reference.

## A Few Notes on Implementation, With Particular Focus on Java <sup>2</sup>

- When you are port-knocking, the firewall will be dropping your packets. This means that it will, barring network error, receive your SYN packets, but you won't get anything back. This means your connection attempts will time out. As you might not want to wait that long before sending your next SYN, look into the variant of Java's `Socket.connect(...)` which provides a timeout parameter.
- There are several situations in your firewall-manipulation in which you want to insert a time-sensitive clause. Some of them can be handled by simply remembering the time of the last action, but at some point you may wish to say something like "do this, in x seconds/minutes." For this, you should look into Java's `Timer` class. Something to keep in mind is that because it will run in a separate thread, our method should be `synchronized` or use one of Java's concurrent data structures (eg. `ConcurrentHashMap`).
- Finally, there will be many times when you will want to see what packets are actually getting sent or not sent while you're debugging your code. For this, the VM provides a few common packet sniffers including `Ethereal`. A user guide for `Ethereal` is available at <http://www.ethereal.com/distribution/docs/user-guide-us.pdf>. We also provide a simple but very useful command-line utility called `netcat` which can serve as a barebones TCP and UDP client or server, including the ability to pipe standard input into, or standard output out of, the connection. For information on how to use `netcat`, run `nc -h` or read its man page.

## Handing in

We will do hand ins the same way as was done for Sentinel unless otherwise noted. When finished, please take your code out of the VM, and use `cs166_handin SynCity` to hand in your code.

You will be required to have a `README` that documents everything you did and if there are any bugs in the project. The `README` should contain a section of about one page analyzing the security of port knocking (see item 3 of your task). You should also comment your code so that we can read it easily.

## Grading

We will be having interactive grading for this assignment (And most likely for all future assignments). We will be grading you based not just on functionality but also on the design, efficiency, commenting, and documentation.

## Internet Lab Rules

For this project, all your coding must be carried out in virtual machines running in the Internet Lab. You may use VMware shared folders to back up your code into your normal home directory or into your CS 166 project directory, as a safeguard against VMware crashes. Even still, coding for this project is to be done within the virtual machines.

---

<sup>2</sup>...but which generalize well to other languages.

The Internet Lab has around 16 individual machines, and it should be obvious that this class has significantly more students than machines. Keeping this in mind, you might want to plan your schedule so that none or very little of your work is compressed into the final moments before this project is due. If it becomes necessary we will institute a wait list for machines in the Internet lab. Our use of the Internet lab is contingent upon everyone respecting the computers in there, not carrying drinks in, and not abusing access to this lab. Doing any of those things will lead to 166 being thrown out of the Internet Lab, and consequently to a total existence failure for the course. This would not bode well for you, and you should therefore not engage in such actions.