Paxson
Spring 2017

CS 161 Computer Security — Project 3

Due: May 5, 2017, 11:59PM
(no late submissions accepted)

Version 1.0: April 17, 2017

Background

Your unsung-but-heroic efforts two months ago led to the miraculous defeat of the evil Calnet! Freedom reigns across Caltopia . . .

. . . *Or does it?*

Walking home from a spirited late-night victory celebration with your Birkland colleagues, you find your mind uneasy. In the wake of Calnet's destruction, Governor Vladivostok Getin has called for an election to determine Caltopia's future. Surely this is the right step forward, yet . . . Something is out of place. Something is wrong. Your security spider-sense tingles!

Your phone vibrates with an incoming text. But whoa, you have zero bars—how did that happen? You check out the short message: `Vital u act or with this election we lose all. Vlad-baby's gonna steal it. Access and leverage his comm w/ gov-of-caltopia.info. Cripple repressive sw + steal email cred. + expose fraud = profit. --Neo`

Your head spins with questions and possibilities. Could such a heinous plot really be under way? Couldn't you just have a break from all this hassle until the end of the semester?

And how should you even begin? Can you really go up against the very top of the government all alone by yourself?

Your phone vibrates again. `Also, suggest you work in team of 2. --Neo`

Getting Started

Haunted by the prospect of a stolen election, and daunted by the fate of Caltopia resting entirely in your hands, you find yourself sleepless and your mind racing. Throughout the night, texts come in, building out the picture and providing snippets of guidance.

First, Neo points you at an obscure file on The Pirate Bay that turns out to supply you with a VM tailored for enabling you to complete each step of the task ahead.

Software Setup

You can run and investigate the VM on your own computer. You will need the following software:

- VirtualBox,[1] the virtualization server.
- Your favorite SSH client.[2]

Neo placed the VM image at `https://inst.eecs.berkeley.edu/~cs161-tak/nethack-0.2.ova` (a 1.2GB download.) Download it and import it in VirtualBox via File → Import Appliance. NOTE: If you are using Windows you may need to disable the Serial Port. To do this click on Settings → Ports and uncheck "Enable Serial Port." You may need to do this for Port 1 and Port 2.

Make sure you have Internet connectivity when you first start the VM. Open a browser and check that you can load `http://example.com`. If not, you should go to Settings → Network → Advanced and enable Cable Connected. Restart the VM afterwards.

All of the network programs will run inside of this VM. The image is a bare-bones Xubuntu Linux desktop installation on a 32-bit Intel architecture.

Memory management. The Virtual Machine comes pre-configured with a 2GB memory limit. We recommend using as much memory as you can afford. To do this, go to Settings → System and adjust the Base memory slider. You can experience unexpected crashes if the VM runs out of memory. To conserve memory, when operating within your VM:

- don't surf the web (except where necessary);
- don't browse "heavy" websites;
- and definitely don't watch any videos.

Save early, save often. Follow instructions precisely!

SSL Tools

The OpenSSL Project develops the `libcrypto` and `libssl` libraries, and the accompanying `openssl` tool, which together provide full support for the latest SSL/TLS protocols. Important tools you should familiarize yourself with include:

- `openssl genrsa`: This tool allows you to generate public/private keypairs. This keypair can be used for both authentication and encryption. As indicated by its name, this tool generates keys for use with the RSA algorithm. While OpenSSL also supports other public-key algorithms, Neo indicates you shouldn't need those.

[1] VirtualBox is available at `https://www.virtualbox.org`, or from your package manager in Linux. The release notes on The Pirate Bay indicate that the VM has successfully worked with version 5.1.8.

[2] On Windows, Neo recommends PuTTY: `http://www.chiark.greenend.org.uk/~sgtatham/putty/`.

- `openssl rsa`: This tool allows you to inspect keys generated with the previous tool.
- `openssl req`: This tool allows you to generate and inspect Certificate Signing Requests (CSR's). A CSR is what you hand to a Certificate Authority (CA) for them to sign, and the CA will copy information from the CSR into a signed Certificate that they'll give you back.
- `openssl x509`: X.509 is the standard used for formatting certificates. This tool allows you to inspect and manipulate such certificates.
- `openssl s_client`: The netcat of SSL connections. You can use this to connect to SSL-enabled servers and send and receive data and view certificates and other connection metadata.

For more information about each tool, check out their man pages (e.g., `man genrsa`).

To inspect SSL-related files, you can use the following syntax:

```
  # RSA Private Key
openssl rsa -in filename -text

  # RSA Public key. You may need to use -pubin instead
  # of -RSAPublicKey_in depending on the input file.
openssl rsa -in filename -text -RSAPublicKey_in

  # Certificate Signing Request
openssl req -in filename -text

  # X.509 certificate
openssl x509 -in filename -text
```

Ethics

This project requires that you communicate and interact with **real** machines on the Internet. You **must not** attack the machines in any way other than via intercepting and altering communication that originates from your VM as part of this project. Do not ever attack the servers directly for any reason! Any such attack or attempted unauthorized access constitutes a violation of the **Berkeley Campus Code of Student Conduct** and could have direct consequences for you as a student. In addition, it may expose you to **legal jeopardy**. Finally, attacks on our servers or other university infrastructure may make it impossible for us to do a project like this in the future. Please don't ruin that opportunity for other students.

You should conduct all of your analysis and exploration of the VM environment from within the VM environment. Inspecting the VM externally is not allowed and as such constitutes cheating.

If you have any questions about whether an attack or other action is in scope, do not hesitate to post a private note on Piazza.

The Task

You may interact with the VM via the provided graphical desktop. You can start a shell in the graphical interface using the terminal desktop icon, or you can SSH in using the *username:password* `neo:neo`. The SSH command is `ssh -p 16103 neo@127.0.0.1`.

The first thing you should do after you get a shell is run `nethackctl`. The first time you do this, you have to enter your class accounts in the format `cs161-`$x_1x_2x_3$`,cs161-`$x_4x_5x_6$, where $x_1, \ldots, x_6$ are the letters of your class accounts. You need to list the accounts in alphabetical order, *with no spaces in between.* For example, if a student with class account `cs161-wei` teams with a student with class account `cs161-vvm`, then you would enter the string "`cs161-vvm,cs161-wei`".[3]

Then, you will be able to select which question you want to work on. You will need to interact with different network services for each question. The services for each question are only running when that question is 'active.' To switch questions, just run `nethackctl` again. While it's possible to do each question independently, we strongly encouraged you to go in order, as each question builds conceptually upon the prior.

You can also use `nethackctl` to reset the VM to a clean state, for example if you want to enter your partner's account name or if you accidentally deleted an important project file. Remember to backup your progress outside of the VM when you do this.

[3] If you want to do some initial exploration by yourself before you've finalized your team, you can start off using just your class account for this configuration step. Once you have your team in place, you'll need to start again with a clean VM. Any solution secrets you've learned for your private VM image will be different from your team's final secrets. This reconfiguration process should go quickly once you've developed solutions the first time.

Question 1 *Exploring the Territory* (30 points)

Neo knows that it could prove daunting to find yourself confronted with the formidable task ahead. To get started, he has informed you that Gov. Getin maintains a server, `vault.gov-of-caltopia.info`, which has been recently removed from the public Internet and is now running only in his local network. Why? What is it hiding? (What's in the box??)

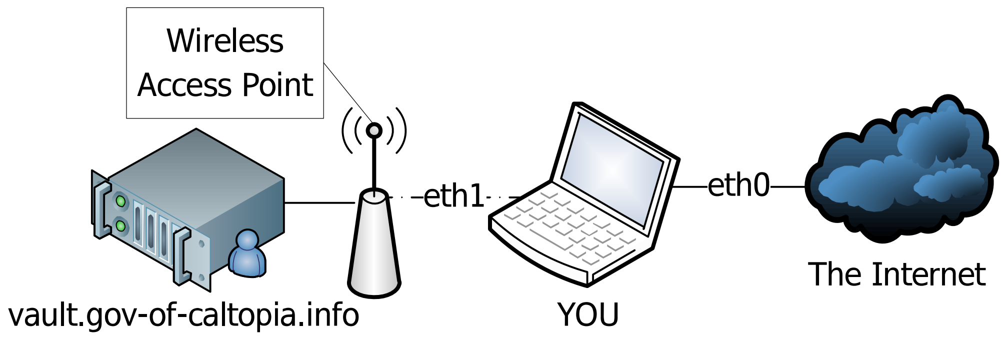
Neo has gotten you a foothold by placing your device in the same local network, attached on the `eth1` network interface. A little birdy tweeted that Gov. Getin's server is the only other host in the local network currently, which is no larger than a `/24` subnet. Additionally, Neo has installed the tools he suspects you'll need on your device, specifically `nmap` and `netcat`.

Nmap is a network port scanner which probes the ports of common services on a machine to test if those services are running and accessible. For example, Neo tells you he has a server at `52.52.137.246`. Try running `nmap 52.52.137.246` to see what popular services are available. Beyond scanning just a single machine, `nmap` can scan a whole IP address range using CIDR[4] notation (e.g., 52.52.137.246/26). Note the software takes some time to scan through many addresses.

Netcat is a tool that can make arbitrary TCP and UDP connections, and can be useful to help communicate with a service. For example, running `nc SERVER PORT` will establish a TCP connection with `SERVER` on port `PORT`, which passes the inputs from `stdin` to the server.

Neo urges you to familiarize yourself with these tools (take a look at their man pages), as one common procedure for attacking a system is to scan for available services and exploit a vulnerability in one of them. Often you can find a vulnerability in a service just by searching online! `vault.gov-of-caltopia.info` seems to be a very old server without any updates. Find and break into Gov. Getin's vault to find what is inside!

[4] "CIDR" refers to notation for describing ranges of IP addresses. IPv4 addresses are 32-bit values. Often we write an address as four numbers separated by dots (e.g., 172.16.254.1), where each number ranges from 0 to 255, thus representing 8 bits each. If we'd like to refer to a set of addresses that share a prefix though, we can use *CIDR* notation, which looks something like 172.16.254.1/X. In CIDR notation, X is a number indicating the length of the fixed prefix (in bits). Thus, 172.16.254.1/24 is the set of addresses where the first 24 bits (e.g., the first 3 IP address numbers) are fixed, leading to the set 172.16.254.* (where * is a wildcard). Frequently, the unspecified bytes are left off, so another way to write this address range would be 172.162.254/24 (no ".1"). If we refer to a network (or subnet) as a /X, it carries a similar definition, indicating a network assigned IP address that have the same X-bit prefixes.



The network diagram of Gov. Getin's local network.

Submit the following files:

`q1/secret`

This file contains a secret token from Gov. Getin's server. Put only the token in this file in the same format as you discovered it.

`q1/exploit`

This file is a shell script named `exploit` which should take an IP address as the first argument, attack the server at that IP address using your discovered attack, and print out the secret token. Our grading tool executes your script as `./exploit address` where address represents the IP address of the target host containing the secret, and checks whether the outputted secret token is correct. Do note that any network connection takes time, and **think carefully to ensure your code does not have any race conditions due to network latency**. Our grading environment may have different network latencies than in your testing VM. NOTE: **You must not hard code the secret or the IP address in your script.**

`q1/explanation.txt`

This file includes a description of the procedure you used to obtain the secret.

Question 2 **(80 points)**

Part A – *Can you hack it?* (20 points)

Neo is certain that he has obtained a packet capture of a secret conversation—but alas, one encrypted with TLS. `~/q2a/q2a.pcap` holds a copy.

Normally, the content of the conversation would be completely unknowable ... but in this case, the server's private key showed up on Pastebin, and Neo has provided a copy in `~/q2a/q2a_privkey.priv`. You must decrypt the conversation and obtain the secret within.

Neo has emphasized to you the importance of familiarizing yourself with tools for capturing and analyzing network traffic. The VM comes with two of these already installed: the graphical *Wireshark* utility (start via Menu → Internet → Wireshark), and the command-line tool *tcpdump*. *Wireshark* has the ability to decrypt TLS traffic if you know the private key. *tcpdump* does not.

After opening Wireshark, you can use the Files → Open option in Wireshark's main interface to load the packet capture. This will show the encrypted packets from the captured traffic. Look over the packets and the format of the connection between the client and the server.

To decrypt SSL traffic, you must install the private key into Wireshark. http://wiki.wireshark.org/SSL has detailed documentation about working with SSL in Wireshark, but in short, you'll need to:

- Open the SSL protocol preferences: Edit → Preferences → Protocols → SSL
- Add a new RSA private key, next to "RSA keys list" click Edit... → +, and fill in:
  - **IP address:** the server's IP, or the string `any`
  - **Port:** 443
  - **Protocol:** http
  - **Key File:** *browse to find private key*
  - **Password:** *leave empty*

You'll now be able to view the decrypted traffic from the packet capture.

Submit the following files:

`q2/a/secret`
: This file contains the secret from the conversation. This is **one** long random string (no spaces) that appears in the (English) plaintext. Put only the secret in this file in the same format as observed.

`q2/a/explanation.txt`
: This file includes:

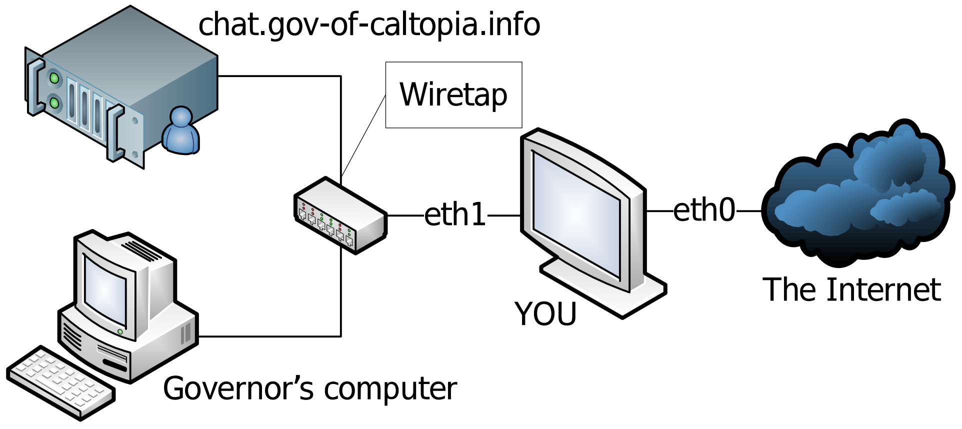
a) A description of the procedure you used to obtain the secret.

b) A list of cryptographic algorithms that were used for this TLS connection. For each algorithm, describe in one sentence its use in the protocol.

c) A discussion of whether there was any technical way by which you could have been prevented from decrypting the conversation even though you have obtained the private key. If so, briefly describe the technical approach. If not, explain why no such approach is feasible.

Part B – *Rumors Coming True* (60 points)

Neo informs you that there's a little surprise tucked away inside Gov. Getin's headquarters: a nifty device that captures Gov. Getin's network traffic. The device tunnels the traffic to your Virtual Machine, where it shows up at the Ethernet device `eth1`:



The network diagram for Neo's traffic-capture hack.

The buzz on the street is that Gov. Getin regularly checks his chat logs at `chat.gov-of-caltopia.info`. If only you could somehow figure out how to get a peek at those logs ... But, alas, they're encrypted using TLS.

You find a copy of the server's public key at `~/q2b/server_pubkey.pub`. But how is that helpful?

A late-night text from Neo indicates he managed to recover the code for the tool that the server at `chat.gov-of-caltopia.info` used to generate the public-private keypair, `~/q2b/generate_rsa_pair.c`, and a `Makefile` that compiles it in the same directory. Neo only had about 20 seconds to look it over before passing it along to you but it "`has luser stamped all over it`" ...

You must somehow leverage the code's poor quality to recover the contents of the encrypted conversation. Try to be as efficient as possible.

Neo senses that `chat.gov-of-caltopia.info` is set up to only accept traffic from Gov. Getin's computer. So don't bother trying to access it directly.

Submit the following files:

`q2/b/secret`

This file contains the secret from the conversation, which is again **one** long random string (no spaces). Put only the secret in this file in the same format as observed.

`q2/b/chat.priv`

This file contains the private key for `chat.gov-of-caltopia.info` in PEM format.

`q2/b/generate_rsa_pair.c`

This file contains the code you used to generate the private key. The autograder will compile this file using the same `Makefile` that shipped with your VM.

`q2/b/run`

A script that runs `generate_rsa_pair`, the compiled version of the source code you submitted, and prints the private key in PEM format to standard output. Do NOT call make or compile in this run script. **The script must not print anything else besides the key.**

NOTE: **Do not hard code key values or file paths in your solution,** beyond file paths already in the code or explicitly allowed. Your solution should take a public key from `server_pubkey.pub` in the current directory and print the corresponding private key. The public key is guaranteed to have been generated by the original `generate_rsa_pair.c`.

`q2/b/explanation.txt`

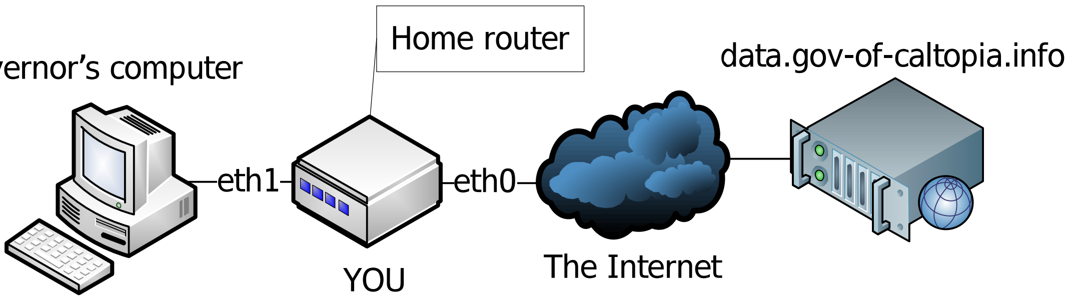
This file includes:

a) A description of the procedure you used to obtain the secret.

b) The line number of the line in the original `generate_rsa_pair.c` that doesn't do what the comment above the line states. Discuss whether or not (and **why**) fixing this line so it does what the comment says it should will make this key generation scheme be secure.

Question 3 *Dead Man Walking* (60 points)

The Gov. Getin's home router used a default password, which Neo leveraged to "upgrade" its firmware: hello, MITM! The hacked code again tunnels all of the traffic through your VM, with the home network attached to `eth1` and the Internet on `eth0`:



The network diagram for MITM fun.

Something related to the previous chat logs appears to connect to `data.gov-of-caltopia.info` every 30 seconds. You should use Wireshark to verify this. You need to **intercept and rewrite** these communications to foil Gov. Getin's plan.

Neo has obtained a snippet of code this software uses to verify the certificates. You can find it at `~/q3/client_fragment.c`. Neo also spotted that `data.gov-of-caltopia.info`'s TLS certificate is signed by the *Budget Certs R Us* Certification Authority.[5]

Courtesy of Neo, you can request a signed certificate for *any* CN ending in `.neocal.info` at https://budget-certs-r-us.biz. When providing Budget Certs R Us with a certificate signing request (CSR), you need to provide the contents of `~/IDENTIFICATION_SECRET` to identify yourself.[6] To create a new CSR, you'll want to use the `openssl req` tool. Neo has also provided you with an additional tool, `~/q3/rewrite_cn`, that given a CSR will rewrite the common name with **exactly** what you've given it in a text file.[7] This will come in handy.

The `sslsniff` tool (provided) is very useful for performing MITM attacks on TLS connections. Given a certificate/private key pair, it intercepts incoming connections and presents them with your certificate. It also connects as a client to the original incoming connection target, and subsequently relays any data between the two connections.

When doing so, sslsniff has access to the plaintext data (because of the use of your certificate, plus that it creates the actual connection to the original server). It logs the plaintext to the file `~/q3/sslsniff/sslsniff.log`.

[5]While the software in this question may trust *Budget Certs R Us*, your computer's operating system is more prudent, and does not. Expect certificate warnings if you attempt to visit `https://data.gov-of-caltopia.info` directly.

[6] **You remembered to log out and select question 3, right?**

[7]A word to the wise: it's prudent to inspect the text file given to rewrite_cn with `hexdump`.

You can install the certificate/private key pair using `~/q3/sslsniff/sslsniff_install_cert`, and start sslsniff with `~/q3/sslsniff/sslsniff`. Neo has modified sslsniff in this VM to pass all HTTP requests to the simple Python script `~/q3/sslsniff/rewriter.py` for modification. You can modify `rewriter.py` to change these requests on-the-fly. (**Warning:** be careful when altering application level protocols. You must adhere to the application specifications. Failure to do so can result in client crashes, or the server appearing to hang. For example, for HTTP the `Content-Length` header must *exactly* match the length of the request body!) Your computer **must** have a working Internet connection to solve this problem!

Neo has flagged for you that **there are two vulnerabilities to utilize for this problem**, and it's vital that you develop successful attacks for both of them. Doing so ensures that even if one of the vulnerabilities gets fixed, the other will still work for future intelligence-gathering.

IMPORTANT: one of the approaches *requires* `rewrite_cn` (i.e., there is no way to carry out the attack using the standard `openssl` tool). If you believe you have two different approaches neither of which requires using `rewrite_cn`, contact us privately before you develop them further so we can potentially help you avoid unnecessary work.

ALSO IMPORTANT: it is crucial that you check the success of your attack, and also know how to undo its effects. In order to do that, Neo has hacked into the data server, and, with his usual humor, set up a way for you to *break the 4th wall*[8] by going to `https://4thwall.neocal.info`.[9]

Submission *Important:* Leave `data.gov-of-caltopia.info` in the right state between your submission and the submission deadline (don't reset via the 4th wall after you carry out the attack for the final time).

Submit the following files:

`q3/secret`
: This file contains the secret from the communication. Put only the secret in this file in the same format as observed.

`q3/rewriter.py`
: This file contains your modified code that rewrites the network traffic. NOTE: **You must not hard code the rewritten traffic in your script.** While the traffic generated in the grading environment will be semantically the same (e.g., same HTTP headers, data fields, etc.) as in your VMs, you must not assume or hard code the ordering, position, or length/size of these.

`q3/data0.priv`

`q3/data0.req`

[8] `http://en.wikipedia.org/wiki/Fourth_wall`

[9] Neo laughs, "bet these lusers wouldn't even get the reference!"

`q3/data0.x509`

These files contain the private key, certificate signing request, and certificate respectively for the first approach you use to MITM the connection. The files must be in PEM format.

`q3/data1.priv`

`q3/data1.req`

`q3/data1.x509`

These files contain the private key, certificate signing request, and certificate respectively for the second approach you use to MITM the connection. The files must be in PEM format.

`q3/explanation.txt`

This file includes:

a) A description of the procedure you used to obtain the secret.

b) A discussion of what, if anything, `gov-of-caltopia.info` can do to protect against these attacks.

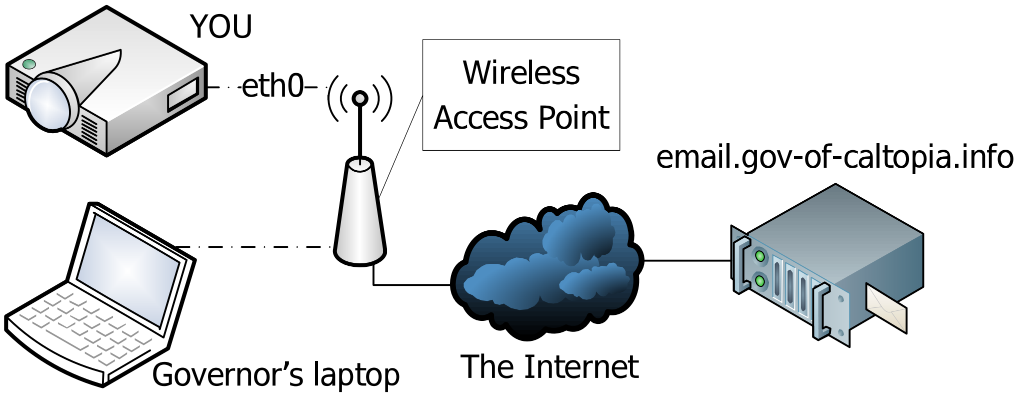
c) A discussion of what, if anything, Governor Vladivostok Getin can do to protect against these attacks.

d) A discussion of what, if anything, `budget-certs-r-us.biz` can do to protect against these attacks.

Question 4 ***Anything You Say Can And Will Be Used Against You*** **(60 points)**

The events of the last question have left Governor Vladivostok Getin feverishly attempting to log into his webmail, `https://email.gov-of-caltopia.info`, to delete any evidence. He can't believe his appalling luck when he finds the email server is currently undergoing maintenance! He can log in, but can't reach his email. In a haze of desperation he repeatedly attempts to log in over and over.

After suspecting Gov. Getin's malicious intentions, Neo arranged for a spy camera to appear in the Governor's office to obtain evidence, connected to the office wireless network, just like the Governor's laptop. The camera tunnels traffic to and from your Virtual Machine—you can view the Governor's browser in your VM using a handy shortcut at `~/q4/Governor's webcam`. The wireless network is attached to `eth0`:



The network diagram for the Governor's "closeup".

You have to somehow get Gov. Getin's webmail credentials in order to expose his nefarious freedom-squashing plot. Unfortunately, though, one of the Governor's lackeys set things up so he only ever logs into his webmail by opening a fully-patched Chromium web browser,[10] typing `https://email.gov-of-caltopia.info`, hitting Enter, then entering his credentials into a webform . . . and he'll only do this if everything about the email website **exactly matches** what he's expecting.

To carry out this attack, you need to somehow redirect the Governor to your own webserver—but while having his *fully-up-to-date* version of Chromium still saying he's visiting `https://email.gov-of-caltopia.info` **with no warnings**. If any warnings appear, the Governor will figure Wait A Sec That's Not Right, refrain from entering any information, restart Chromium, and begin again.

Neo has provided some tools to go after this vital task. `~/q4/digipwntar` contains some very interesting files from a Certificate Authority called DigiPwntar. DigiPwntar is trusted by the Governor's browser.[11] Neo has also given you a skeleton for using a Python packet capture library called `scapy`.[12] The skeleton can be found in

[10]Chromium is the open source browser that forms the basis for Google Chrome.

[11]And sensibly not trusted by your computer. Expect certificate warnings if visiting directly.

[12] `http://www.secdev.org/projects/scapy/doc/usage.html` has extensive documentation on how to use `scapy`. One hint: to access layers of a packet `pkt`, you can e.g. use `pkt.haslayer(TCP)` and `pkt[TCP]`.

`~/q4/pcap_tool/pcap_tool.py`.

Lastly, Neo has provided you with your own webserver at `~/q4/local_webserver/` that will do everything you need to do—should you somehow manage to get Gov. Getin's browser to visit it without any warnings. As before, your computer **must** have a working Internet connection to solve this problem.

NOTE: The attack needs to be as stealthy as possible! The attack should not disrupt communications to any other site Gov. Getin visits.

Submit the following files:

`q4/secret`
: This file contains the Governor's password. Put only the password in the format observed, and nothing else, in this file.

`q4/pcap_tool.py`
: This file contains your modified code that carries out the attack. NOTE: **You must not hard code IP addresses in your script, including your own host's IP address.** You should be able to set any relevant IP fields by reusing data already observed in captured packets or through the `scapy` API. Our grading environment will use different IP address for hosts!

`q4/email.priv`

`q4/email.req`

`q4/email.x509`
: These files contain the private key, certificate signing request, and certificate respectively that you used for your local webserver. The files must be in PEM format.

`q4/explanation.txt`
: This file includes:

  a) A description of the procedure you used to obtain the secret.

  b) A discussion of whether there are any mechanisms or protocols Governor Vladivostok Getin could have used to defend himself against your attack. If so, explain why your attack wouldn't work when using these. If not, discuss the implications of this attack for the use of TLS in the Internet today.

Feedback

We'd love to hear your feedback on this project. Please include a file `feedback.txt` if you are so inclined.

Submission Summary

Make sure you carefully followed all instructions in generating each file, such as the file content and format. Points will not be awarded for mistakes in content, format or following instructions, however minor.

Make sure you check that you submitted files correct. Any submission errors will also not be awarded points, however minor. You will not be awarded points or regrade requests for any submission errors such as submitting the wrong file version, submitting an empty file, or submitting only half of your file. It is your job to check that your submission is correct.

In summary, you must submit the following directory tree on your EECS instructional class account:

```
q1/secret
q1/exploit
q1/explanation.txt
q2/a/secret
q2/a/explanation.txt
q2/b/secret
q2/b/chat.priv
q2/b/generate_rsa_pair.c
q2/b/run
q2/b/explanation.txt
q3/secret
q3/rewriter.py
q3/data0.priv
q3/data0.req
q3/data0.x509
q3/data1.priv
q3/data1.req
q3/data1.x509
q3/explanation.txt
q4/secret
q4/pcap_tool.py
q4/email.priv
q4/email.req
q4/email.x509
q4/explanation.txt
feedback.txt (optional)
```