CS255: Cryptography and Computer Security **Winter 2016**

Project #2

Due: Wednesday, March 9, 11:59pm.

WARNING: This project will be much more time-consuming than Project 1. Please start it as early as possible so that you do not run out of time.

There is no starter code for this assignment. You may not use any libraries for the assignment apart from the ones we explicitly specify in this handout (you will have to write some code yourself that makes network connections and HTTP requests).

1 Introduction

Transport-layer security (TLS) is the protocol that protects the vast majority of encrypted traffic transiting the Internet. Although the principle behind TLS is simple—it's just a way to set up an encrypted tunnel over TCP—in practice it is surprisingly difficult to use correctly. The goal of this assignment is to expose you to some of the inner workings of TLS and to give you a taste of the subtleties of TLS. By the time you complete this assignment, you will have a much deeper understanding of TLS than you do now and, relative to the average programmer, you will be a TLS expert.

You should view this assignment as a mini-research project on TLS. To complete it successfully, you might have to read the pyOpenSSL documentation, parts of the pyOpenSSL source code, blog posts about TLS, and maybe even some RFCs.

2 Your Task

In this project, you will build a command-line utility called `scurl`[1] that implements a small subset of the functionality of `curl`, the standard Linux/UNIX/BSD utility. If you have not seen `curl` before, it is a nifty little tool for making HTTP requests.[2] For example, you can run:

```
$ curl https://www.stanford.edu/
```

and `curl` will print out the HTML of the Stanford homepage. The `curl` utility is installed on Stanford's Corn cluster, so you can play around with it there. Run

```
$ man curl
```

on the Corn shell to get a manual page describing `curl`.

Your task is to implement a new Linux utility called `scurl`. We should be able to invoke your `scurl` utility as:

```
$ ./scurl [options] URL
```

[1] `scurl` = "secure `curl`" or maybe "Stanford `curl`"

[2] The standard `curl` tool also speaks many other protocols, but we will ignore those in this project.

where `URL` is some URL with an `https` scheme, such as https://crypto.stanford.edu/. Your `scurl` should reject any URL that does not have an `https` scheme by printing a one-line message to `stderr` and exiting with a non-zero status code.

Your `scurl` implementation should support the following options:

- `--tlsv1.0`, `--tlsv1.1`, `--tlsv1.2`, `--sslv3`, `-3`. These options should do exactly what their `curl` counterparts do. See the `curl` manpage for details. If none of these options is passed in, your `scurl` should act as if the user passed in the option `--tlsv1.2`.

- `--ciphers`. See the `curl` manpage for an explanation of this option. Run "`openssl ciphers`" to get a list of all ciphersuites supported on your machine. To use the `--ciphers` flag, pass in a list of colon-separated ciphersuite names in your order of preference. For example:

  ```
  $ ./scurl --ciphers DHE-DSS-AES256-SHA:DH-RSA-AES256-SHA https://www.stanford.edu/
  ```

- `--crlfile`. See the `curl` manpage for an explanation of this option. When the user invokes `scurl` with this option, your `scurl` should look at the serial numbers of the certificates in the CRL file and refuse any certificate with one of the offending serial numbers.[3] You may assume that the entire CRL file fits easily in memory.

- `--cacert`. See the `curl` manpage for an explanation of this option. You may assume that there is only one CA certificate in the CA certificate file.

- `--allow-stale-certs N`. This option does not exist in `curl` but you will implement it in `scurl`. When a user invokes `scurl` with this option with an argument `N`, your implementation should accept a certificate C as valid if (a) C is an otherwise valid certificate that has expired and (b) C expired within the past `N` days. The argument `N` to this option must be a non-negative integer. If this option is used several times, the last one will be used.

- `--pinnedpublickey <filename>`. A variant of this option exists in recent versions of `curl` and you will implement a stripped-down version of it in `scurl`. This option takes a single argument, which is the path to a public key in PEM format. When a user passes this argument to `scurl`, the `scurl` TLS client will only connect to a server if the server's TLS certificate is exactly the one contained in the specified file. You must use the SHA-256 certificate fingerprint functionality built into pyOpenSSL to compare the server's certificate to the pinned public key.

  If the server sends the `scurl` client a certificate chain, you should only check that the leaf certificate matches the "pinned" public key certificate—ignore any CA certificates that the server sends.

  This option *overrides* the `--cacert` and `--crlfile` options. If this option is used several times, the last one will be used.

[3] In a real implementation, you would also compare the issuer of the CRL to the issuer of the certificate, but pyOpenSSL does not make this easy to do.

Important Requirements. Your implementation:

- must be written in Python 2 (version 2.7),
- must use `pyOpenSSL` as the TLS library (version 0.15),
- must run on the Corn/FarmShare machines *without* any modifications to the system libraries or packages,
- must free all system resources (open sockets, open files, etc.) before exiting normally,[4]
- must contain an executable Python file called `scurl` in the top-level directory of your submission (do *not* call your executable `scurl.py` or anything else)—we should be able to execute it as `./scurl {arguments}` on the command line,
- must return an exit code of `0` on success and something non-zero on error or failure,
- must print a **one-line error message to `stderr`** and exit immediately if any sort of error or failure occurs,
- must not print any error messages to `stdout` nor print more than one line of error information to `stderr`, and
- must behave like `curl` in how it handles errors and in how it interprets invalid or crazy combinations of options and arguments.

These last six points are important! We will grade your projects by script and if your `scurl` does not observe the expected input/output behavior, we will deduct points for each failed test case.

We will test your implementation on the Corn machines. You may develop your software locally but please test your code on the Corn machines to make sure that everything works on Corn as expected. If your main computer is a Windows machine, you should feel free to develop your software on the Corn machines.

Prohibitions. We will consider it a violation of the Honor Code if your implementation does any of the following prohibited things...

- You are ***not allowed to use*** the `requests` library, `urllib`, `urllib2`, `urllib3`, or `httplib`. You should not need an HTTP library for this project. Your implementation **must use pyOpenSSL for all of the TLS functionality**.

  You *may* use standard Python libraries (`sys`, `os`, `urlparse`, etc.) that have nothing to do with HTTP or SSL/TLS.

- Your implementation must not use external libraries, apart from pyOpenSSL. You should not need any other libraries to complete the assignment and you must not include source files from external libraries in your project submission.

[4] In the case of an error that causes the program to terminate prematurely, try to clean up as best you can but don't worry too much about it.

- Your implementation may not invoke any command-line utilities or other pre-existing binaries. You may not use `os.system()`, `Popen`, `subprocess` or any functionality of that sort.
- Your implementation may not make network connections, except for a single connection to the hostname in the URL passed to `scurl`.
- Your implementation must not read any files apart from those explicitly given as arguments to `scurl`.

Grading. We will grade your assignment based on style, functionality, and security.

- **Style (10%).** Your code should be easy to read, be split up into logical blocks, and use sensible variable and function names. You should explain subtle pieces of logic with comments. We will not require you to use any particular Python style guide, but you should feel free to use one if you wish; Google, Mozilla, and others publish them.
- **Functionality (30%).** Your `scurl` should allow us to make HTTPS requests to all of our favorite websites. Your `scurl` should correctly implement all of the command-line options described in this handout.
- **Security (60%).** Your `scurl` must behave as `curl` does when presented with crazy or invalid certificates, invalid arguments, or other malformed inputs. Test your code extensively!

3 Advice

The following pieces of advice may be helpful to you while completing the assignment.

- **We will stress test your code, so program defensively!** Your `scurl` should reject invalid certificates of all forms. Your `scurl` should handle invalid arguments gracefully. Your `scurl` should behave well even if the server does not.
- In Python, the name of the pyOpenSSL module is `OpenSSL`, so you import it with "`import OpenSSL`."
- To work with many modern websites (which is a requirement) your `scurl` client will need to support a TLS feature called SNI.
- Your `scurl` may make requests with the HTTP/1.0 protocol, if you wish.
- There is a very useful website called badssl.com that has links to many running TLS servers with bad and broken certificates.[5] This is a great resource for testing.
- For testing the `--pinnedpublickey` option, the instructions on "Public Key Extraction" on this page (`https://curl.haxx.se/libcurl/c/CURLOPT_PINNEDPUBLICKEY.html`) may be useful.
- Use standard `curl` as a reference implementation. You should run curl with crazy arguments and crazy combinations of arguments to see how it behaves. Your `scurl` should behave in exactly the same way that `curl` does when given the options that `curl` supports.

[5] The person behind the site (Lucas Garron) was a CS255 student, then was a CS255 TA, and is now a TLS ninja at Google. So I am not joking when I say that this class will turn you into an expert in TLS!

- You will need to write logic to validate certificates.
- To build a successful `scurl` client, you will have to test it extensively. The broader a set of test cases you can generate and the more automated you can make your testing regimen, the easier your life will be.

3.1 Submission Instructions

Research Study. In CS255 this year we are running a research study on cryptographic APIs. To indicate whether you do or do not wish to have your submitted code included in the study, please read this short information sheet (located at https://crypto.stanford.edu/cs255study/cs255-consent-form.txt) and fill out the three-question form at the bottom. Whether you participate will have no impact on your grade—you should participate only if you're interested. When you submit your project via Corn, include one completed form (in `txt` format) per student.

You should submit your project via Corn as you did with Project 1.

1. Collect your project source files (including one completed research form per team member) into a single directory on the Corn machines. **Please include one completed research form per team member. If you decline to participate in the study, just indicate that on the form.**
2. Run `cd` *`your_project_dir`*.
3. Execute `/usr/class/cs255/bin/submit`.