CSci 4271W (010 section) Project Instructions

Project 1 **Due: March 6th and March 18th, 2025**

1 Project: Badly Coded Bookmark Manager (BCBM)

Badly Coded, Inc., (BCI) is a software vendor responsible for keeping security students at UMN busily employed for many years. (Ask your friends! They're known for such incredible products as the "Badly Coded Versioning System (bcvs)" the "Badly Coded Compression System (bczip)", the "Badly Coded Print Daemon (bcpd)", and the "Badly Coded Calendar Alert System (bccal)" to name but a few...) The latest product under development at BCI is the Badly Coded Bookmark Manager (`bcbm`, for short), and the team responsible for this project has approached your group for a security assessment of (the pre-alpha version of) this code.

BCBM is intended to be a cross-browser and cross-device bookmark manager system: it can be run once on a user's device, and accessed from any browser on the device. Using the browser interface, users can add bookmarks, delete them, and order them in a "top 10" list of favorite bookmarks. BCBM can also synchronize a user's bookmarks across multiple devices, by uploading and downloading encrypted copies of the user's bookmarks file to a back-end BCBM cloud server. This synchronization between devices is managed through an email registration process: users enter an email address and password in the client that is submitted to the cloud server when uploading and downloading the bookmarks file. To set up an account, users visit a page on the cloud server, enter their email address, and are sent a confirmation link that can be used to set or reset the password.

Functionally, BCBM itself consists of two codebases:

- The BCBM "client", `bcbmc`, is actually a small HTTP application server program (built using the CrowCPP framework) that runs on a user's device. Users connect to the client from a web browser, and the client writes web pages that accept user inputs and contain the results of user actions, which are then displayed in the browser. The client also handles the client-side portion of synchronizing between multiple devices and communicates with...
- A cloud server, `bcbms` for facilitating synchronization between devices. The cloud server program has functionality to transfer encrypted bookmark files, verify user credentials, and register or update credentials using registration codes.

However, a number of other entities are potentially involved: `bcbmc` connects to third-party websites to download website icons and web page titles; the user's interactions are sent through one or more web browsers, and the browser (typically) displays pages from other web sites that could potentially try to issue commands to the client through included resources. As you progress through the stages of the project, you'll see more and more of the code for BCBM, but these other entities, and the communication with them, should also be considered as you think about the potential threats to the client and server.

2 Project 1: Local `bcbmc`

The project will consider the BCBM code in three stages. For project 1, we'll consider only the client code, restricted to the portions that manage the local copy of the bookmarks file. (In project 2, we'll add the portions of the client and server code that handle synchronization, and in project 3, we'll add the server code that handles authentication and registration)

To install `bcbmc` on your VM (please don't install this on a real machine — it's really buggy code) follow these steps:

1. SSH to your VM (from e.g. a VOLE session or CSE Labs workstation) using "X forwarding" to allow the VM to open GUI applications:

```
ssh -X student@csel-xsme-s25-csci4271-NNN
```

2. Install a graphical web browser on your VM. Our recommendation is that this the best way to keep the steps you do for the project separate from your normal web browsing.

```
sudo apt install firefox
export XAUTHORITY="$HOME/.Xauthority"
```

You can also use `chromium` or other browsers available in Ubuntu if you'd prefer. The `export` command setting an environment variable is a workaround you'll need to provide in every terminal your start a browser in, because of some details about how Ubuntu packages Firefox (and Chromium) using snaps.

2. Clone the source repo using:

```
git clone https://github.umn.edu/badly-coded-alpha/bcbmc.git
```

3. Build by changing directory into `bcbmc` and running `./install.sh`

This will build and install the `bcbmc` executable. To interact with `bcbmc`, type `bcbmc` on the command line, and then from another terminal **also on your VM** (also connected with X forwarding, so using the `-X` option as in step 1), open a browser (e.g., `firefox`) and navigate to `http://localhost:8888/`. (Note: `localhost` is a name for whatever computer you are currently using; it's a way to tell a program that expects to connect to other computers to connect instead to the computer the tool is running on. So the `localhost` URL will only work to connect to bcbmc when you run the browser on the VM.)

The code for bcbmc is split across several files:

- The main code for the application server is in `bcbmc.cpp`, which uses the CrowCPP framework. The framework itself (the code in `crow_all.h`) should be excluded from your analysis, but you should at least skim the documentation for CrowCPP to help you read and understand the code in `bcbmc.cpp`. The `main` function is in `bcbmc.cpp` and uses Crow to "route" requests to the other modules in the system. It also calls functions to set the six-digit "authentication" code that (attempts to?) prevent(s) other pages/programs from calling the bookmark management functions of `bcbmc`.
- The file `acode.c` contains the code for choosing a random authentication code.
- The file `manage.c` contains most of the code for responding to user requests. For most interactions, a function is called from `bcbmc.cpp` with the inputs provided by the user, and the function writes an html file with results to be displayed. `bcbmc.cpp` then returns this file to the browser for display.
- The file `sync.c` will eventually include all of the code for synchronizing bookmarks across devices. For now, all of the code involving the encryption and decryption of cloud bookmark files, and their upload and download from the server, is left out / stubbed in. What is present is the code for merging the local bookmarks file with the version obtained from the cloud, using the locally maintained "transactions" file (`bcbm.tx`).

Some other notes about interacting with/testing BCBM for Project 1:

- A possible threat that the developers clearly considered is tampering/spoofing of the user's commands by other websites displayed in the browser. This is a possibility because any website can tell the browser to load certain "resources," commonly images, from another website. A simple example of this is the file:

```
<html><img src="http://localhost:8888/666666/add/http%3A%2F%2Fevil.com%2F"></html>
```

which would cause the browser to send an "add" command for `http://evil.com/` to their local bcbmc client. (Thanks to the randomized authentication string, the command will almost certainly be rejected!) You can experiment with similar files as examples of this type of threat: local files can be loaded in the browser. For testing, you can also use the `curl` command line tool (on your VM) to send a request to bcbmc and observe the response:

```
% curl "http://localhost:8888/666666/add/http%3A%2F%2Fevil.com%2F"
```

(Since bcbmc itself uses `curl` you might benefit from spending some time reading its man pages)

- bcbmc also interacts with web pages when adding them as bookmarks. For testing purposes, you can place pages in your CSELabs user homepage and add/delete them as bookmarks as you see fit. A PoC exploit using a page from one of these directories should include the code for the page, and a curl command to call the corresponding action in the client.
- **NO SOCIAL ENGINEERING!** The purpose of this project is to find problems inherent in the code of bcbm; so attacks that involve asking the user to run `bcbmc` with odd arguments, or paste strange values into `bcbmc` are out of scope. (However, it is OK to assume that the user might bookmark an attacker-controlled page or load an attacker-controlled page in their browser.)
- While there are several Elevation of Privilege attacks planted in this code, and many more may be possible due to the fact that it is Badly Coded, don't forget to look for other possible STRIDE threats, such as potential spoofing, tampering, information disclosure and denial of service against the client server. Your final report, after all, should include at least two vulnerabilities that are not in the EoP category.

3 Assignment and Timeline

As also explained on the Project overview page, your assignment is to review and produce a security assessment of the `bcbmc` code so far. You should produce a report that includes (1) a system design section that answers the question "what are we building?", (2) a threat model using STRIDE that outlines the types of threats you evaluated the system against, and (3) a summary of dindings that documents and explains at least 2 vulnerabilities. You will also produce "proof of concept" programs showing that your vulnerabilities can be exploited, and a `README.md` file explaining how to run your proof of concept programs and what to expect. Finally, your report should have a separate section entitled "Addendum: Group Accountablity" which details which sections were primarily drafted by each group member and your overall plan to ensure that each group member will be the primary contributor to a design section, threat model section, and vulnerability description over the course of the semester. To demonstrate the separate efforts of each group member, you will need to individually submit drafts of a section you have been working on before each full group submission.

3.1 Grading

- As noted on the project information page,
  - 20 points are assigned for including the required elements. Check that you included: design section, DFD, all DFD elements, threat model section, listing of security goals in the threat model, STRIDE by each element in the DFD, summary of findings, 2 vulnerabilities, mitigation discussions for all vulnerabilities);
  - (For Project 1) 5 points are assigned for the per-student section draft;

  - (For Project 1) 5 points are assigned for the "group accountability" addendum;
  - 30 points are assigned for high-quality writing.
  - 16 points are split between the two vulnerabilities in your project report: for each vulnerability, did you clearly describe an actual vulnerability, correctly explain how it could be mitigated, and clearly distinguish it from the other vulnerabilities?
  - 8 points are available for the clarity and correctness of the `README.md` file in your proof of concept repo.
  - 8 points are available for the readability, reproducibility, and correctness of each of two proof-of-concept programs in your repo (16 points total).

- To re-emphasize an important point from the project information page: **Claiming a vulnerability without confirming your claims about it is academic and professional dishonesty and may result in a grade of 0 for the project**. (An example would be to claim that a stack buffer in the function `gen_auth_code` can be overflowed, resulting in a tampering attack, without giving any evidence that an input from an untrusted source, of the appropriate length, could actually reach the `gen_auth_code` function. If all you are sure of is that the code looks risky but you're not sure an attack is possible, you can get partial credit for saying that honestly.)

3.2 Timeline:

- **By 11:59pm, ~~Tuesday, March 4th~~ Thursday, March 6th**: Each member of a group will submit a PDF draft of one section of the report that they have been responsible for. This submission isn't for detailed grading or feedback, but to confirm that each member of the group has been working on part of the report.

- **By 11:59pm, Monday, March 17th**: If you are requesting an extension to Friday March 21st, for project 1 (thereby using your only extension request) you should send email to Prof. McCamant with title **"4271: Project 1 extension request"** and list your name and your group members, along with a copy of your current report draft.

- **By 11:59pm, Tuesday, March 18th**: You should submit your PDF report. The report should also include a clearly labeled reference to a private **UMN** github repo (so, on `github.umn.edu`) that is accessible to `smccaman` and `foiso001`, where we can find your two (or more) proof of concept programs and `"README.md"` file.

Happy Hacking!