

CSE 484 / M584 Lab 4:

Root-Cause Analysis and Patching Lab

1 Structure

Due Dates

- **Part A:** Friday May 30th, 11:59pm (Max of 2 late days!)
- **Part B:** Friday May 30th, 11:59pm (Max of 2 late days!)
- **Part C:** Friday June 6th, 11:59pm (NO late days!)

Handin

- All components on [Gradescope](#), see Deliverables.

Groups

- **All components:** Individual or partner

Grading

Points: 43 + EC

Note: unlike prior labs writeups may be partnered.

- **Part A:** 15 (11 patch, 4 writeup)
- **Part B:** 13 (one RCA writeup for exploit3 or exploit4)
- **Part C:** 15 (11 patch, 4 writeup: must be the same exploit as Part B's RCA)
- **Extra Credit:** 5.6 (For doing another combination of RCA+Patch)

2 Formatting Notes for this Handout

- Programs, files, or tools are stylized like `this`.
- Commands for programs are stylized like `this`.
- Code, servers, and strings are stylized like `this`.

Contents

1	Structure	1
2	Formatting Notes for this Handout	1
3	Before you start	2
4	Overview	3
5	Background	3
6	Grading and Guidelines	5
7	Getting Started	7
8	Testing / Turn-in process for patches	8
9	Using a Browser (recommended)	8
10	Notes and Hints	8
11	Extra Credit	9

3 Before you start

- Fork the repo: <https://gitlab.cs.washington.edu/dkohlbre/cse484-rcalab-25sp>
- Make your fork private!
- Work on umna.kcs.washington.edu
- Read the READMEs in the repository.
- Read the Background section, it is actually important!
- Take a look at the RCA template on Gradescope
- Get the filled out exploit1 RCAs (UWE-484-01) from the course webpage [RCA pdf](#)
- Make sure you've seen the lecture on lab4.

4 Overview

This lab is designed to give you some experience with performing root-cause analysis (RCA) on exploits and with patching them. Conceptually, this is similar to the process that might happen if your company discovered an exploit being used in-the-wild against an application you make or if your company's vulnerability disclosure program receives a proof-of-concept exploit for an application. You can see examples of these in Google Project Zero's (P0) writeups of exploits found by Google's Threat Analysis Group (TAG). We encourage reading a few of these as examples for your writeups. Also remember to look at the RCAs for exploit1!

You will need to take a working but unexplained exploit, determine what bugs in the application are used by this exploit, and propose an appropriate set of fixes for the application. All exploits and patches relate to material we've covered in the course, and require you to draw on material you've learned in lecture/homeworks/labs previously.

For **Part A**, you get RCAs for the bug (exploit1), and only need to patch it.

For **Part B**, you will turn in an RCA for one of the new issues (exploit3 or 4).

For **Part C**, you will turn in a patch for the issue you analyzed in Part B.

4.1 tinyserv

The application being exploited and needing patches is a small open-source HTTP-only web server written in C. It probably has vulnerabilities beyond the ones you need to explain. On the code side, you'll be given:

- The C HTTP server (tinyserv)
- Working exploits against that server: `exploit1.sh` – `exploit4.sh`
- A normal connection example to the server: `nonexploit.sh`
- You should run tinyserv with (exactly) `./tinyserv ./files/`
- Remember to read READMEs!

5 Background

5.1 Part A:

You work on an *open-source* project: tinyserv. It is not the best tiny HTTP server, but it works. Most of your users use it to host small, static web pages from their own private servers, and it even has a fancy admin log that shows all previous visitor's requests! The admin log and admin homepage are password protected by a completely random password when server starts, so only the administrators of the server can possibly access it.

But today you received a report of exploits being used against tinyserv in the wild. Your users are in danger! Thankfully a more experienced developer took the lead, and has performed a root-cause analysis (RCA) on the exploit sample to identify the underlying vulnerability.

As the junior developer on the project, this is a perfect opportunity to patch your first serious security bug. You'll need to use their RCA and the exploit sample to guide you as you develop a patch.

5.2 Part B:

Now that you've proven yourself on patching exploit1, your team is trusting you with both the root-cause analysis and the patching for any future bugs. Today, you got three new reports of exploits being used against tinyserv. All 3 exploits have found a way to access the admin log without knowing the password. How are they doing this?!

Your fellow maintainers (the TAs, in section) will handle exploit2, don't RCA/patch it. You need to choose one of exploit 3 or 4 and write up an RCA for it.

5.3 Part C:

It is time to write a patch for that exploit! Hopefully you will get feedback from your co-maintainers (your TAs) before your patch is done. (Don't wait for feedback to start though.)

A meta-note: you will write a better RCA and (maybe) a better patch if you role-play a bit here. Remember that you are a maintainer of the tinyserv open-source project and are doing your best to solve this problem!

5.4 Deliverables

Please note that we only require 1 of the 2 exploits to be 'solved' in Parts B/C. This is to provide you alternatives if one of the exploits particularly stumps you, and allow for extra credit!

We strongly encourage you to look at both; it can be deceiving at first look which is 'easiest' or 'hardest' to solve! All elements are turned in via Gradescope, turn them in as a pair if doing the project as a pair.

5.5 Part A

Submit to: Lab4 - PartA - Exploit1 Patch

- A patch for exploit1 named exploit1-patch.diff
- A short (1–3 paragraphs) description of your patch (description.txt)
 - This must contain a clearly labeled paragraph describing if any functionality changes were made. ("None" may be appropriate.)
 - This should be submitted at the same time as the patch: upload two files (a .txt text file and the .diff)

5.6 Part B

Submit to ONE of: Lab4 - PartB - Exploit3 RCA or Lab4 - PartB - Exploit4 RCA

- Fill out the RCA form on Gradescope.

5.7 Part C

Submit to ONE of: Lab4 - PartC - Exploit3 Patch or Lab4 - PartC - Exploit4 Patch

- A patch for exploit3 or 4 named exploit3-patch.diff or exploit4-patch.diff
- A short (1–3 paragraphs) description of your patch (description.txt)
 - This must contain a clearly labeled paragraph describing if any functionality changes were made. (“None” may be appropriate.)
 - This should be submitted at the same time as the patch: upload two files (a .txt text file and the .diff)
- You *must* submit to the same exploit as your Part B RCA.

Diff files All patch (.diff) files must be generated by the `handin.sh` script. Transfer them to your personal machine using `scp`. Do not copy-paste the text from these files. Do not modify them.

Writeup Your writeup **MUST** contain:

- A brief description of what your fix is supposed to achieve
- A brief description of the changes your patch made
- A clearly labeled paragraph stating if/what your patch changed anything about how users or admins use the server.
 - For optimal patches, the answer is usually “No changes”.
 - Some approaches to fixing problems will change minor functionality, and will lose points if that is not documented in the writeup here.

When submitting your writeup, we are expecting a plaintext file (e.g. not a word doc, not a pdf, etc.) Markdown syntax or other text-only formatting are fine. Gradescope allows uploading multiple files to a single assignment: please only upload ONE patch and ONE description to the assignment.

6 Grading and Guidelines

6.1 RCA Grading/Guidance

Most of the questions can be answered well in a few sentences. (The vulnerability details may need to be longer, more detail here is better.) *Remember the difference between the exploit and the vulnerability.* Carefully examine the bug(s) to determine all aspects of the vulnerability, not just what the exploit does with the vulnerability. Refer back to the starter RCA for a good example.

Technical accuracy matters in your RCAs; don’t make technical claims you haven’t seen evidence for (e.g., tinyserv doesn’t support HTTPS, so don’t claim something about HTTPS!)

6.2 Patch Grading/Guidance (11 points)

Patches will be graded approximately as follows:

- Does the patch attempt a good-faith fix of the bug? (If not, 0 points total for the patch, we won't grade patches that don't try to fix the correct problem.)
- Is the patch correctly formatted and only included relevant code changes? (1p)
- Does the patch pass all functionality tests? (2p) – Autograded
- Does the patch pass all exploitN tests? (2p) – Autograded
- Is this a generally good way to try and fix this specific bug? (2p)
- Is there any (important) functionality that is broken that we didn't test for? (2p)
- Corner cases or small aspects of the bug that weren't caught or were added? (1p)
- Will this patch make it hard to re-introduce the same bug in the future, or does it generally seem like a good long-term solution? (1p)

It is common that a short and simple patch will do well, but may miss 1–2 points. That is OK! A full credit patch is tricky to write and will require you to pay close attention to multiple aspects of the code and recall different parts of the course. We do not expect many students to earn perfects on their patches. **There are secret tests that you will not see the grades for, but will affect our manual grading of non-autograded parts.**

For patch quality, your fixes for each bug-exploit pair should completely fix the bug and prevent similar exploitation of that same bug – not just for the provided exploit – but not necessarily all classes of that bug. For example, if an exploit uses a buffer overflow vulnerability, then your patch should prevent further usage of the same buffer overflow. However, you would not need to fix every potential buffer overflow in the program. If we can change the length or characters of our exploit and retrigger an exploit using the same vulnerable code spot, then you have not patched the vulnerability. (Think about how the Lab2 XSS filters worked: a proper fix to that code would not simply change the filter from the one in XSS2 to the one in XSS4; the fix would prevent all XSS attempts via the `?url=` parameter. That might involve changing code in >1 place)

Other ways to lose points:

- Including patches for multiple bugs in one patch file
- Including large-scale reformatting in your patch file (e.g. changing all spaces to tabs)

6.3 Patch Writeup Grading/Guidance (4 points)

You should submit a 1–2 paragraph writeup describing your patch along with the patch itself. Unlike the patch plan in the RCA, you should write this after you have finished the patch. Your writeup should concisely describe what changes you made, why you made them, and what the expected results of your changes are. If you believe you accidentally introduced new bugs, or weren't able to fully fix the vulnerability, document that here. Think of this like the message you might have written to the other maintainers to explain this patch.

7 Getting Started

tinyserv is much larger than the other pieces of code we've looked at in this course, and it may not be obvious where to start! Remember that your goal is not to become the world expert on tinyserv, but to understand the basics of its operation and identify specific bugs.

7.1 Working on RCAs

We recommend that you not look at the RCA for exploit1 right away. Instead, spend some time trying to figure out why exploit1 works on your own (and/or pay attention to lecture!) Then if you get completely stuck or think you've solved it take a look at the filled out RCA.

If you are not sure where to start in an RCA, consider what must be true for the exploit to accomplish its goal, and read some of the code that seems to perform the relevant actions. You can also look at the difference between nonexploit.sh and the exploits to see if there are obvious differences in what is being sent to tinyserv.

7.2 Working on Patches

When designing a patch, come at this the same way you would with a bug in your own code. You have some functionality you want to preserve (tinyserv should serve pages, allow admin logins, support response caching, etc.) and you have a bug that you want to fix.

We strongly recommend using a web browser to interact with your patched tinyserv to make sure it behaves reasonably. Historically we've seen patches that work fine for a simple connection example (nonexploit.sh) but break normal web browser interactions with tinyserv.

7.3 Using git for tinyserv

You should start by making a fork of the gitlab repository for tinyserv. Then you can share your fork with your partner and sync work there.

Once you have cloned the repository to the server you will be working on, we strongly recommend using branches to manage each patch independently.

We have already created one git branch per-exploit (called "dev-exploitN") and everything relevant is under version control.

You can switch branches in git with `git checkout <branchname>`
For example, if you're switching back and forth between working on exploits 2/3 you might do:

- `git commit -am ``Message detailing progress so far on exploit2```
- `git checkout dev-exploit3`
- ...doing some work on exploit3 patching ...
- `git commit -am ``Message detailing progress so far in exploit3```
- `git checkout dev-exploit2`

8 Testing / Turn-in process for patches

Please carefully read the following and ask questions on Ed if anything is unclear. For each exploitN, we want one diff (exploitN-patch.diff) giving the changes needed for tinyserv.c. *Your patch for one exploit must not contain the patch(es) for any other exploit(s).* This lets us grade each patch independently.

To hand in a patch:

1. Run the `./handin.sh exploitN` script making sure to read all of the output.
2. Check that the file it created (.diff) looks like it matches your changes
3. Upload to gradescope, it will get run through an autograder
4. If any tests fail, it is not a very good patch.
5. If all tests pass, it might be a good patch. We can't test everything automatically!

handin.sh will put files that you'll need to turn in in the turnins directory (again, like Lab1). It will also tell you the generated patch files and any backup files. All handin.sh does is use git to generate a list of changes you made in tinyserv.c.

When you are ready, use scp to get these files to your personal machine and upload to gradescope. Do not copy-paste the text from these files. Do not modify these files.

Make sure you also upload a description of your patch as a text file.

9 Using a Browser (recommended)

While all of the exploits can be run from the command line, you can also browse (and even exploit) the site using a browser.

Like WebLab, you'll need to add a cookie to your browser's cookie store. Each group has a unique `LAB_GROUP_SECRET_KEY` cookie that is randomly generated to avoid anyone accidentally talking to any other group's tinyserv. This key is not part of the lab/exploits, and is not intended to have any bugs!

If you visit your server's page without a cookie set, there will be a page with a box to set your cookie. Make sure you copy it in without quotes around it! Alternatively after opening the browser console you can type: `document.cookie = "LAB_GROUP_SECRET_KEY=your group secret";`

10 Notes and Hints

- You can build and run this on umnak.cs.washington.edu or on a local Linux setup (MacOS may be possible to get working but will require unsupported changes.)
- Check the READMEs and make sure you ran setup.sh
- You only need to modify tinyserv.c. You do not need to modify the exploit files or write any exploit code. (You are welcome to do so for helping understanding/debugging though.)

- You cannot make changes that require adding new libraries. Any patch that does not build on the autograder will not get most of its credit.
- Port numbers have been automatically assigned based on your group number and are automatically inserted into tinyserv and your exploits. Please don't change any of the port number related code, it should all 'just work'.
- You can find your port number in `rcalab_port`
- You can find your group secret in `rcalab_group_secret` (note the extra quotes in it that you need to remove when adding the cookie to your browser!)
- Don't leave tinyserv running when you aren't using it; it is quite vulnerable! (It will self-terminate after 3 hours automatically.)
- Assume that everything under `target/` (e.g. the README and the `target/tinyserv`) is hosted on the public repository for tinyserv
 - This is the repo that your users would clone from github or whatever and then use as-is to run their web servers.
 - They will be modifying stuff in files/ and adding/removing content there. They will not be modifying anything else.
- You can access the admin login page by visiting either of the admin links on the main page.
- To reiterate, everything related to the `rcalab_group_secret` is not part of the lab. You should not edit any of the code dealing with it or really worry about it at all. (Bugs found in it are worth EC though. Report ASAP if you find one.)
- `nonexploit.sh`, `exploit1.sh`, `exploit2.sh`, `exploit3.sh`, and `exploit4.sh` are shell scripts using the curl tool. Exploit3 also requires building `exploit3_cookiegen`. You can run them like this: `./exploit2.sh`
- If you want more control over how your HTTP requests are sent, you can use nc or telnet.
- None of the bugs have to do with curl. curl is just a useful tool for generating http requests.
- We've talked about critical elements for each of the bugs in different parts of the course. You may want to rewatch lectures or review your notes for the relevant parts!

11 Extra Credit

You can earn extra credit on this assignment by turning in RCAs, patches, and patch write-ups for one additional exploit.

Extra credit must be submitted to the "Lab 4 - EC - Exploit3" or "Lab 4 - EC - Exploit4" Gradescope assignment.

Extra credit components are worth 20% of their normal value.

You can also submit any security critical bugs you find to David for (minimal) bonus points.