

# Exploit Analysis and Patching

**Deadline:** Tuesday, December 12, 11:59pm (**no late days**)

**Turn in:** Gradescope assignments, see Deliverables

**Individual or group:** Groups of up to 3

**Points: 43**

- Part A: 15 (11 patch, 4 writeup)
- Part B: 13 (one RCA writeup for splot2, splot3, or splot4)
- Part C: 15 (one patch, corresponding to Part B's RCA)
- Extra Credit: 25% of Part B+C for *one additional* splot (from splot2, splot3, splot4)

## Before you start

- As in Lab 1, **the following has to be done first, individually:**
  - Sign up on Canvas for a FinalProject group (even if you're working alone)
  - Fill out this Google form with your ssh public key:  
<https://forms.gle/8zVjNnsUoJLvTs849>
  - You might want to revisit the [SSH and SCP Guide](#)
  - **Once setup is done, you can access the lab via ssh to `cse484.cs.washington.edu` with username `cse484-au23-fp-[groupnumber]`**
- *Read the Background section below, it is actually important!*
- Download the filled RCA form for splot1 (UWE-484-01): [\[pdf\]](#)
- Download the RCA template form from the course webpage: [\[docx\]](#), [\[pdf\]](#)
- Make sure you've seen the lecture from Monday, November 27

## Table of Contents

[Overview](#)

[Background](#)

[Deliverables](#)

[Grading and Guidelines](#)

[Getting started](#)

[Testing / Turn-in process for patches](#)

[Using a Browser](#)

[Notes](#)

[Extra Credit](#)

# 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 [RCA form \(UWE-484-1\) 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 a writeup for the bug (splot1), and only need to patch it. For Part B you will turn in a writeup for **one** of a set of new bugs (splot2-4) using the template on the course page. For Part C, you will turn in a separate patch for the bug you analyzed in Part B.

## tinyserv

The application being exploited and needing patches is a small 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: `splot1.sh`, `splot2.sh`, `splot3.c` and `splot4.sh`
- A normal connection example to the server: `nonsplot.sh`

# Background

## 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 serve small, static web pages from their own private servers, and it even has a fancy admin page that shows all previous visitor's requests! Handily, the admin page is password protected by a completely random password each time the 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.

## Part B/C:

Now that you've proven yourself on patching sploit1, 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 page *without knowing the password*. How are they doing this?!

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 developer of tinyserv and are doing your best to solve this problem!

# Deliverables

Please note that we only require 1 of the 3 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! All elements are turned in to Gradescope.

## Part A (Starter Patch): Submit to Gradescope "Final Project Part A"

- A patch for exploit1 named `exploit1-patch.diff`
- A short (1-2 paragraphs) description of your patch (PDF)

## Part B (RCA): Submit to Gradescope "Final Project Part B+C SploitN Version" where N depends on which exploit you do

- A completed RCA (using the template) for one of `exploit{2,3,4}`

## Part C (Patch): Submit to Gradescope "Final Project Part B+C SploitN Version" where N depends on which exploit you do

- A patch for one of `exploit{2,3,4}` named `exploitN-patch.diff`
- A short (1-2 paragraphs) description of your patch (PDF)
- *(You must patch the same exploit in Part B that you did the RCA for in Part B)*

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 these files.

# Grading and Guidelines

## RCA Grading/Guidance

RCAs are graded based on the point values stated on the RCA form. 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!)

## 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**)
- Is the patch correctly formatted and only included relevant code changes? (1p)
- Does the patch pass all functionality tests? (2p)
- Does the patch pass all sploitN tests? (2p)
- 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.

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.)

Other ways to lose points:

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

## 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.

# Getting started

tinyserv is much larger than the other pieces of code we've looked at in this course, and starting can be intimidating! 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.

## Working on RCAs

We recommend that you *not* look at the RCA for `spl0it1` right away. Instead, spend some time trying to figure out why `spl0it1` works on your own. 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 `nonspl0it.sh` and the `spl0its` to see if there are obvious differences in what is being sent to tinyserv.

## 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 (`nonspl0it.sh`) but break normal web browser interactions with tinyserv.

## Using git for tinyservlab

We have already created one git branch per-spl0it (called "dev-spl0itN") and everything relevant in your homedir is under version control. We highly recommend you use these branches to work on each spl0it independently.

You can switch branches in git with `git checkout <branchname>`.

For example, if you're switching back and forth between working on spl0its 2/3 you might do:

```
git commit -am "Message detailing progress so far on spl0it2"
git checkout dev-spl0it3
... [doing some work on spl0it3 patching] ...
git commit -am "Message detailing progress so far in spl0it3"
git checkout dev-spl0it2
```

# Testing / Turn-in process for patches

**The turn-in process is different from the past labs, so please carefully read the following and ask questions on Ed if anything is unclear.** For each `spl0itN`, we want one diff (`spl0itN-patch.diff`) against the original `main.c` file you are given with the `main.c` file after you are done patching the `spl0it`. *Your patch for one `spl0it` 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 <spl0itN>` script making sure to read all of the output.
- 2) Check that the file it created (`.diff`) looks OK
- 3) Run the `./test-patch.sh N` where `N` is the `spl0it` number you want to test
- 4) If any tests fail, it is not a perfect patch (and it will tell you why, but only after RCAs are due!)
- 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 `main.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.

## Using a Browser

While all of the `spl0its` can be run from the command line on the `cse484` machine, you can also browse (and even exploit) the site using a browser. To do this, you'll need to add your group key cookie to your browser's cookie store. Each group has a unique `LAB_GROUP_SECRET_KEY` that is randomly generated to avoid anyone accidentally talking to any other group. This key is *not part of the lab, 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, for most browsers, after opening the inspector, you can go to the Console tab and type:

```
document.cookie = "LAB_GROUP_SECRET_KEY=<your group secret>";
```

# Notes

- You can build and run this on `cse484.cs.washington.edu`, and we recommend this since we will use the `cse484` machine to grade this lab.
- By default, your `homedir` is already a git repository with one branch per-spoit. This is useful for managing multiple patches.
- You only need to modify `main.c`. You do not need to modify the `sploit` files or write any exploit code. (You are welcome to do so for helping understanding/debugging though.)
- 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 `~/lab3_port`
- You can find your group secret in `~/lab3_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! (We will automatically kill any `tinyserv` process that is running for >3 hours)
- You can access the admin login page by visiting either of the admin links on the main page.
- To reiterate, everything related to the `lab3_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.)
- `nonspoit.sh`, `sploit1.sh`, `sploit2.sh`, and `sploit4.sh` are shell scripts using the `curl` tool while `sploit3` is a C file (a Makefile for building it is included). You can run them like this: `./sploit2.sh` and `./sploit3`
- If you are doing the extra credit and want more control over how your HTTP requests are sent, you can use `nc` or `telnet`. For example, on the `cse484` machine, if your port number is `50007`: `echo "GET /index.html HTTP/1.1\r\nCookie: LAB_GROUP_SECRET_KEY=mykey\r\n\r\n" | nc 127.0.0.1 50007`
- Please do not attempt to exploit the lab infrastructure (e.g. patch testing, other groups, etc.) If you believe you found a bug, report it and don't exploit it.
- None of the bugs have to do with `curl`, `curl` is just a useful tool for generating http requests.



## Extra Credit

You can earn extra credit on this assignment by turning in writeups and patches for **one additional exploits for Part B and/or C** (i.e., sploit2, sploit3, or sploit4, as long as it is one you did not do for the required component of the project).

The 2nd RCA and patch will count for 25% of their point value.

**Submit to Gradescope “Final Project Part B+C SploitN Version” where N depends on which sploit you do for extra credit.**