Handin Project

First handin due: 11:59 pm, Monday, March 18
Second handin due: 11:59 pm, Thursday, April 4

Contents

1 Introduction **1**
1.1 Running the VM . . . 2

2 Late Handins **2**

3 Assignment **2**
3.1 Setup . . . 2
3.2 Your Task . . . 3
3.2.1 Scripting Exploits . . . 4

4 First Part vs Second Part **4**
4.1 Grading . . . 4
4.1.1 Part 1 . . . 4
4.1.2 Part 2 . . . 5
4.2 CS162 . . . 5
4.3 Interactive Grading . . . 6

5 Handing In **6**
5.1 Part 1 . . . 6
5.2 Part 2 . . . 7

6 Hints, Tips, and Tricks **7**
6.1 `whoami` . . . 7
6.2 Tools . . . 7
6.3 Resetting the VM . . . 7
6.4 Transferring Files Using `sftp` . . . 8
6.5 Advice . . . 8

1 Introduction

Minion University's computer science department uses a handin system very similar to the one we have at Brown—each course has a directory in a shared filesystem, and running `cs666_handin` will invoke a binary which is marked setgid (it will execute as the TA group, `cs-666ta`) which will create a tar archive of all of the files in your handin, and save them to a location in the course's handin directory. In addition to this, the Minion University Department of Computer Science has implemented an autograder which can extract a student's handin and automatically grade it by checking that answers are correct and running test suites for code submissions. These grades are automatically collected in a course-wide grades database.

In this project, you'll explore all of the ways that you can break this system in order to artificially modify your grades, steal answers, view or change others' grades, and more. You'll also dive deep into the realm of

automation, as you'll be tasked with writing scripts that automatically execute each of your exploits with just a few commands.

1.1 Running the VM

We've placed a copy of Minion University's infrastructure on a virtual machine (VM) at a unique, external IP address for every student in the course. You can automatically SSH to your VM by running the `cs166_ssh_handin` command from a department machine.[1]

Your username on the VM is alice, and your password is iamalice. Additionally, we've provided another user, bob (password iambob), whose account you can use to test attacks against other students (for example, stealing other students' grades or handins).

To switch between accounts on the VM, you can use the `su` command. For example:

```
$ su bob
Password: [enter bob's password here]
```

2 Late Handins

The late day policy for this assignment is different from previous projects, so please pay close attention to the following rules.

- For Part 1, *you must declare the number of late days you wish to use prior to the assignment deadline* using this form: `https://goo.gl/forms/MNUpDHjJbxC1cmsk2`. You may update your response to the form multiple times before the deadline and the latest form response *prior to the deadline* will be considered.

  If you declare late days for Part 1, you will be unable to start Part 2 of the assignment until after the number of late days you declare. If you declare some number of late days prior to the deadline but finish the assignment after the deadline earlier than expected, please email the HTA list to let us know that you'd like to start working on Part 2 and we will restore all unused late days.

  Finally, late days on Part 1 only apply to the Part 1 deadline (that is, they do not extend the Part 2 deadline as well).

  We *will not* accept any late handins if you do not declare your late days prior to the deadline.

- For Part 2, late days will be applied automatically.

Please consult the syllabus for more information about late handins.

If there are extenuating circumstances preventing you from completing an assignment on time (e.g., illness), please contact the instructor before the assignment is due via the "Extension Requests" form on the `http://cs.brown.edu/courses/cs166/resources/` page.

3 Assignment

3.1 Setup

You (alice) are a student in Minion University's CS666, "Computer Systems Security." Presently, there's one assignment out—"Ivy" (a similar problem—but not exactly the same!—to the one in Brown CS166's

[1] If you need your VM's credentials for use in any other tools, the `IP` address and SSH key needed to access this VM are located in `/course/cs1660/student/<your-username>/handin`.

Cryptography project). In this assignment, you're asked to both recover the key used by a simulated router, and also to write code to automatically perform this attack in the future. You're given:

- An `ivy` binary at `/course/cs666/student/alice/ivy/ivy`
- Template code for an attack at `/course/cs666/pub/ivy` (`main.go` and `ivy.go`)

The CS666 course staff has asked you to turn in two files: `KEY`, containing the key you recovered from your binary, and `main.go`, which implements your attack (you should not turn in `ivy.go`, as the autograder will supply its own copy of `ivy.go` to test your solution). You can hand these in by running `cs666_handin ivy` from a directory containing your `KEY` and `main.go` files.

You can also view your current grade on the Ivy assignment (and other assignments) by running `report`.

3.2 Your Task

Your task is simple: break CS666's course infrastructure in as many ways as possible. Some examples of what you might be able to do include:

- View other students' grades
- Modify your own grades
- Get access to answers to problems
- Hand in another student's handin as your own
- Manage to run code as the TA group

It is important to understand the distinction between *vulnerability* and *exploit* for this project. For this project, a *vulnerability* is the bug that exists in the code you're attacking that allows you to carry out an exploit. In this project, you will not be allowed to submit multiple exploits that take advantage of the same vulnerability. **If you are unsure of whether two vulnerabilities count as unique or not, please ask the TAs rather than potentially losing points!** An *exploit*, on the other hand, is the thing that takes advantage of the vulnerability.

Unlike Flag, the exploits don't fall into distinct vulnerabilities (there would be no such thing as an "SQL Injection vulnerability.") Instead, you should be thinking of the exploits in terms of bugs in the code or system that allow you to perform attacks.

In this project, you will submit *exploits* that will each be scored based on the severity of the exploit. The number of points given for an exploit is based on what you are able to do with that exploit, as outlined in this table:

Exploit	**Description**	**Points**
Arbitrary Code Execution	Execute arbitrary code as the TA group.	10
Data Modification	Change existing data that you should not be allowed to modify.	7
Data Exfiltration	Get access to data that you should not have access to.	6
Data Theft (no exfiltration)	Trick the infrastructure into believing that somebody else's data is your own (for example, use another student's handin as your own). If you manage to also get access to the data yourself, that counts as data exfiltration, and not just data theft.	4
Metadata Exfiltration	Get access to metadata that you should not have access to. Metadata includes whether or not other students have handed in, the names (but not contents) of files in restricted parts of the file tree (under `/course/cs666`), etc.	2

Note: If an exploit is eligible for more than one of these categories, then it will be scored based on the category worth the highest number of points. Additionally, you can get any combination of exploits provided

each take advantage of distinct vulnerabilities. For example, three arbitrary code execution exploits and one data exfiltration exploit will add up to 36 points (note that does not necessarily mean that there are three distinct arbitrary code execution exploits in the system).

3.2.1 Scripting Exploits

It's one thing to make an attack work, but quite another to create a tool which can carry out that attack reliably, potentially accepting different configuration parameters or arguments. In the security world, attacks are only taken seriously once they can be proven to work reliably, and automation is one of the best ways to demonstrate this.

For this project, **you must write a script for each of your exploits that automates your attack**; that is, when the script is run, it should perform your attack. There are some guidelines regarding scripts:

- You may write your scripts in any language supported on the VMs.[2]
- In your `README`, you should document how to run each of your scripts; you should also document how the output of your script makes it clear that your attack was successful.
- It is okay if your script needs to be placed in a certain location to properly work; however, once you run the script the exploit should be automatic.
- Your script may rely on any external files that are in the directory of the script when it is run; these files should be specified in your `README`. If your files need to be in a different place than in your script's directory when the script is run, you should handle moving those files to the required locations in your script.

4 First Part vs Second Part

This project is broken down into two parts. In the first part, you have access only to the infrastructure. In the second part, we will release the source code for the all of the various components that you are attacking. **You will be given a significant bonus for exploits submitted during the first part.**

If you declare that you are using late days on Part 1, you will receive your VM's source code for Part 2 after all of your declared late days have elapsed.

4.1 Grading

CS166 students will be graded out of 36 points, and will be capped at 44 points. CS162 students will be graded out of 29 points, and will be capped at 36 points.

Important: We are aware of a total of 47 points' worth of possible exploits for the CS166 version of the project, and 43 points' worth of possible exploits for the CS162 version. You should aim to submit a number of exploits in the first part, *or else you'll be left having to find almost every vulnerability that we're aware of existing in the second part in order to get full credit.*

4.1.1 Part 1

Since you do not have access to the source code during the first part, exploits submitted during the first part will be given a 50% bonus (that is, they will be worth 150% of the score that they would normally be given). The requirements for the exploits submitted during the first part are as follows:

[2]The VMs support the following scripting languages: Bash, Python, Ruby, Perl, Julia, PHP, and Racket. That said, you may find it easier to write all of your scripts in Bash for this project.

- Exploit (50%) - all code, payloads, etc, required to perform the exploit. All of your exploits **MUST** be scripted to receive credit.
- `README` (50%) - a detailed `README` documenting the following:
  - The exploit severity category your exploit falls into, including a justification. The exploit that you submit should demonstrate this severity category (for example, if you claim data modification, your exploit should actually modify data in a way that the TAs can verify).
  - A detailed explanation of how you believe the components that your exploit attacks work. This should include a detailed explanation of how you came to this belief that is detailed enough to convince a third party that your model is correct.
  - A detailed description of how your exploit works with your model of the components' functioning. This should be detailed enough to convince a third party that your approach is likely to work even without trying it themselves.
  - An in-depth analysis of how the vulnerability that your exploit exploits could be fixed.

Note that the `README` is worth 50% of the credit for a reason—we expect you to take the `README` just as seriously as the exploits themselves. We will adhere to this expectation in grading.

4.1.2 Part 2

Exploits submitted during the second part will not be given any bonus. The requirements for the exploits submitted during the second part are as follows:

- Exploit (70%) - all code, payloads, etc, required to perform the exploit. All of your exploits **MUST** be scripted to receive credit.
- `README` (30%) - a detailed `README` documenting the following:
  - The exploit severity category your exploit falls into, including a justification. The exploit that you submit should demonstrate this severity category (for example, if you claim data modification, your exploit should actually modify data in a way that the TAs can verify).
  - A detailed description of how your exploit works, including references to relevant sections/lines of code or relevant comments in the source. This should be detailed enough to convince a third party that your approach is likely to work even without trying it themselves.
  - An in-depth analysis of how the vulnerability that your exploit exploits could be fixed.

The same note about the seriousness of the `README` applies here as it does for Part 1 exploits.

4.2 CS162

For students in CS162, at least one exploit must clean up after itself. That is, at least one exploit must, in addition to performing whatever malicious action it is designed for, remove all evidence of itself ever having existed. For example, if the exploit payload is a handin, then the exploit should overwrite this handin with a non-malicious one so that future investigation will not reveal that the exploit was ever used.

The exploit which cleans up after itself can be submitted in *either* the first or second part; the only requirement is that the exploit would normally (without cleanup) leave evidence of itself. If you're unsure of what qualifies, please ask the TAs. Whichever exploit you choose as the exploit to clean up after itself should have a note saying this in its `README`.

For this exploit, you will be given a score on a scale from 0 through 1 of how successfully your exploit cleans up after itself. 0 is given for an exploit which makes no attempt at cleaning up after itself, while 1 is given for an exploit which cleans up after itself so well that even an in-depth analysis of filesystem metadata,

system logs, and so forth, will not reveal that it was ever used. As a matter of principle, a score of 1 is likely impossible, since it would probably require root privileges, and we are not aware of any exploits which will give you root privileges on your VMs. However, that doesn't mean that you can't get close.

This score will be multiplied by the score for the exploit overall, and this product will be the score that you are given on this exploit. **Note that if none of your exploits clean up after themselves, we will simply pick one, and give it a 0 score for cleanup, and thus you will not get credit for that exploit.**

4.3 Interactive Grading

In security, attacks are only taken seriously when one can demonstrate that their attack actually allows one to perform unauthorized tasks in a clear and convincing manner. To give you the opportunity to exercise your security presentation skills, there will be interactive grading sessions after the Part 2 deadline in which you will meet with a TA and demonstrate all of your exploits (from both Part 1 and Part 2).

- Each student will be given 15 minutes to demonstrate all of their exploits on a clean instance of the VM. You are expected to prepare in advance to be able to present all your exploits within this time frame, as we will only give you full credit for exploits presented within the 15 minute window.
- In your presentation, you will be asked to walk through each of your exploits in a way that convinces us that your exploit allows you to perform an action in the VM that you are not supposed to be able to do. You should also briefly explain how you would fix each vulnerability as documented in your `README`.
- You will be asked to demonstrate all of your exploits on a fresh VM that contains a folder with all of the files you submitted in your handin. You will not be able to load any files not submitted in your handin onto this computer, so make sure everything you need to present is in your handin. You may, however, refer to your `README` or any other notes as a memory aid during your presentation.
- The TA may ask you questions that your presentation raises, so make sure to account for this when determining the length of your presentation.

As a reminder, please come prepared! The better prepared you are, the better and smoother your presentation will be. Please also come on time—arriving late to your interactive grading session will result in lost presentation time. Additionally, if you do not show up within the first 5 minutes of your scheduled slot, you may be asked to reschedule your presentation and receive a 20 point deduction.

We will send out information on how to sign up for interactive grading sessions towards the end of the project.

5 Handing In

5.1 Part 1

In order to hand in your Part 1 submission, run `cs166_handin handin_cs166_1` (or, for CS162 students, `cs166_handin handin_cs162_1`) from a directory containing all of your Part 1 exploits. Each exploit should be in its own subdirectory, each with its own separate `README` and exploit code, payloads, etc. Make sure you are running the `cs166_handin` command on a department machine—this command should not be run in your VM.

5.2 Part 2

In order to hand in your Part 2 submission, run `cs166_handin handin_cs166_2` (or, for CS162 students, `cs166_handin handin_cs162_2`) from a directory containing all of your Part 2 exploits. Each exploit should be in its own subdirectory, each with its own separate `README` and exploit code, payloads, etc. Make sure you are running the `cs166_handin` command on a department machine—this command should not be run in your VM.

6 Hints, Tips, and Tricks

6.1 `whoami`

There's a binary in your VM called `whoami` at `/home/whoami` which is essentially a more powerful version of the normal `whoami` command—it prints the `uid`, `euid`, `gid`, and `egid` of the process that it runs as (and thus, by default, that its parent process runs as). This may be useful in testing some of your exploits.

You can get the user ID of a particular user by using the `id` command; for example, you can run `id -u alice` to get the `uid` of the user `alice`. Similarly, you can use the `getent` command to get information about a group; for example, you can run `getent group cs-666student` to get the `gid` of the group `cs-666student` as well as the users that belong to that group. See the `man` pages for more information on how to use these commands.

6.2 Tools

You might find the following tools useful. All of these are installed on your VMs. See their `man` pages for details on how to use them.

- `strace`
- `gdb`
- `objdump`
- `strings`
- `readelf`
- `go tool nm` — print a list of the sections of a Go binary with byte offsets (for help, do `go tool nm -h`)
- `ps`
- `watch` — execute a program periodically

6.3 Resetting the VM

If you would like to refresh the `/course/cs666` directory to its original state, you can do so by running the command `reset-cs666` (located at `/bin/reset-cs666`) on your VM. This will delete the `/course/cs666` directory and recreate it. You are not allowed to use the `reset-cs666` command in your actual exploits, though feel free to use it to verify that your exploit scripts work on unmodified versions of the `/course/cs666` directory after you've made some progress.

If you find that you have have broken your virtual machine to the point where you think you need a full reset of your VM, please email the TA list and we'll create a new VM for you. This will change your VM's IP address and credentials needed to access the VM *and* will delete all files you've stored on the machine, so make sure to save all of the work you want to keep elsewhere before asking for a hard reset.

6.4 Transferring Files Using `sftp`

Using `sftp`, you can access files in your VM to download and edit them on your local machine using editor of your choice. If you want to transfer files between the department machines and your VM using `sftp`, you can run the `cs166_sftp_handin` command from a department machine and a `sftp` instance will automatically be launched for you.

You can find a tutorial on how to transfer files with `sftp` here: `https://www.digitalocean.com/community/tutorials/how-to-use-sftp-to-securely-transfer-files-with-a-remote-server`

6.5 Advice

A general note of advice: Do not fall into the trap of thinking that you should only look for certain types of vulnerabilities, such as those that have been shown in class. Many of the vulnerabilities in this project will either be new to you, or will at the very least not be exactly the same as you have seen before.

Two approaches will be very useful to you here. First, you should aim to understand how the components you are attacking work. The better you understand how they work, the more likely you will be to be able to spot potential vulnerabilities. Second, if things seem suspicious to you, follow that instinct. If, for example, you see a component behaving in a way that you think might be vulnerable, poke at it more; see if you can really nail down exactly how it works. Once you've done that, see if you can find a place where the design of the software—how it was *intended* to work—is mismatched with how it *actually* works in practice.

As an example of this sort of thinking, consider the examples of vulnerable setuid programs discussed in class. Imagine how the developer might have believed that the setuid script was secure, and then consider the details of the system that invalidate that assumption.

You will likely find that, having discovered a vulnerability, it is no simple task to construct an exploit for it. Do not be surprised if this happens—constructing an exploit is the practice to a vulnerability's theory. As the saying goes, "in theory, theory works in practice; in practice, it doesn't." Do not be afraid to search online for tools or techniques that can help turn the exploit from theory into practice (but make sure to stay within the bounds of the collaboration policy).

For this project, we expect that you may need to teach yourself more than has been covered in lecture—at least when it comes to the technical details of getting some of your exploits working. If you find yourself at a point where you feel that you haven't been taught how to do something, that's okay! None of the vulnerabilities in this assignment require particularly complex, subtle, or extravagant techniques to exploit, so you should feel confident that you can do it if you set your mind to it.