SEAMS - Secure Evaluation and Automation of Malicious Student Code

Nicolaas Kaashoek

1 Introduction

Many courses at MIT require students to complete programming assignments that need to be graded by course staff. This process can be painful for both students and staff, as it may take a while before students receive a grade, and may also require many hours from course staff to actually grade the assignment. While some classes like 6.009 do use automated grading schemes, the code written for classes like 6.858 is sufficiently complicated that many of these out of the box solutions are incapable of grading them. For instance, the code may spin off many processes or require the configuration of system files (e.g. for a web server). Additionally, many existing auto-graders provide poor security guarantees (see section 3). SEAMS aims to address these concerns.

SEAMS leverages containerization to allow for the automated execution of complex code submissions. It enables students to submit their code and have their grade reported soon after without any oversight from course staff, streamlining the grading process. SEAMS also provides strong security guarantees in the form of strict isolation of student code, protection of important staff resources, and evaluates the authenticity of the results reported by the student submissions. As SEAMS uses containers, it scales well; for example, classes that have resource-intensive tests (6.824, for example) could spin up many containers and run them in the cloud.

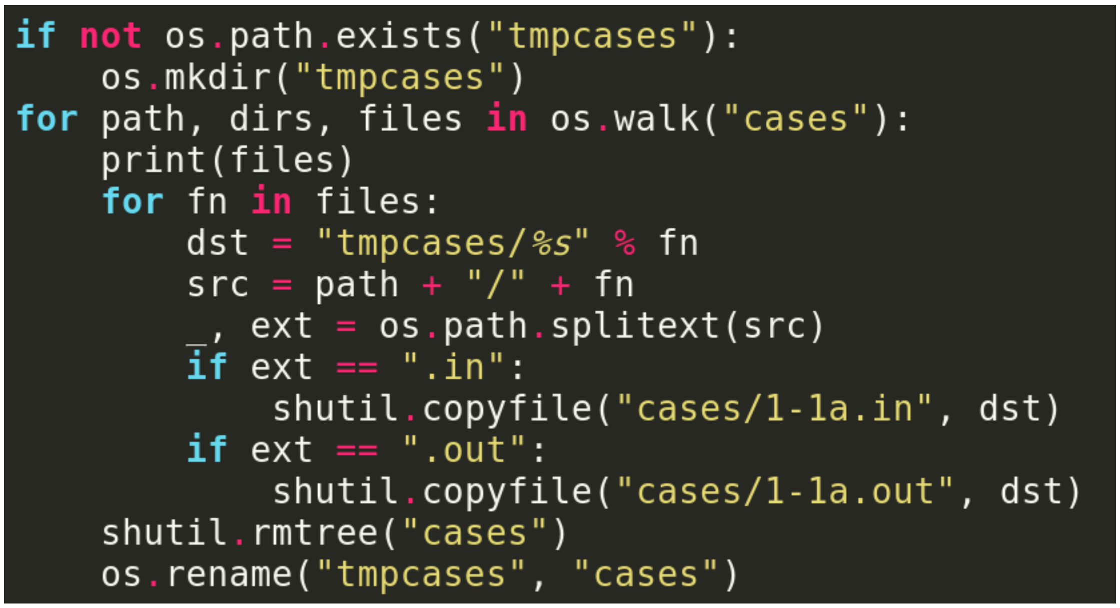
2 Threat Model

There are two approaches that students may take in trying to attack an automated grading system. First, students may try to gain access to resources they aren't permitted to access. This could entail trying to compromise the grading machine, trying to modify the grades file, or trying to access code belonging to other students. Students may also attempt to write code that appears to pass the provided staff test cases while not actually doing so. For example, in 6.858's third lab, student's could just print the expected answers to stdout without actually doing any concolic execution.

With this in mind, the threat model is as follows. Students are able to submit any piece of code that can be triggered from some staff provided grading script. Their code executes with whatever privileges afforded to it by the script, but other than that is unrestricted in its operation. It is assumed that students have full knowledge of SEAMS' system design.

Because some assignments require internet access, SEAMS allows student code to talk to the internet. This means that the submission can attempt to collude with some external server. SEAMS does not, however, allow student code to talk to any other containers or the host.

3 Case Study 6.009



```python
if not os.path.exists("tmpcases"):
    os.mkdir("tmpcases")
for path, dirs, files in os.walk("cases"):
    print(files)
    for fn in files:
        dst = "tmpcases/%s" % fn
        src = path + "/" + fn
        _, ext = os.path.splitext(src)
        if ext == ".in":
            shutil.copyfile("cases/1-1a.in", dst)
        if ext == ".out":
            shutil.copyfile("cases/1-1a.out", dst)
    shutil.rmtree("cases")
    os.rename("tmpcases", "cases")
```

Figure 1: 6.009 Attack

As an example of poor security in existing autograders, I've included the core of a simple attack on the 6.009 autograder. In this lab, students were tested by comparing the output of their game state to an expected set of states. The attack replaces all the input and output files with those of the most simple test case. At that point, students need only solve that test case, or just hard code the answers. This attack was tested and received full points when submitted to the 6.009 grading website.

4 System Overview

SEAMS is a collection of scripts that control the setup and management of a secure grading environment. SEAMS requires that staff specify a grading script that actually runs the student code and outputs the results in some way. The script for scoring the results of this grading script must also be provided by the staff. These components as well as other configurable information are provided by staff in a configuration file. An example configuration file is included in Appendix A. The other components of SEAMS are described in the following sections.

4.1 Isolation

To ensure that student code is unable to gain access to resources outside of the scope of the assignment, SEAMS isolates the running code in an LXC container[1]. LXC containers are available on most Linux distributions, including those used by the 6.858 course staff. SEAMS sets up unprivileged containers, which ensure that no user on the container can ever obtain root access on the host machine. I chose LXC for this project as it is widely supported on most linux distributions, is easy to clean up and configure, and is more lightweight than some other options like Docker.

The SEAMS configuration file informs the system about files that should be moved to the container. Once these files and the student code are placed into the container, the grading script can be run within the it. These containers can be spun up on the fly as clones of a pre-configured base container whenever a student submits code.

While LXC containers ensure that student code cannot access the host machine, they do provide network access. Although it is possible to cut off containers' internet access, some assignments such as 6.858's lab 4 require that the student code can connect to the network. While IPtables' firewall rules do make it possible to restrict the network access given to a container, I found that ensuring normal execution while implementing fine grained network control was challenging; some 6.858 labs need to download files that may be hosted at multiple locations on the internet. For instance, PhantomJS in lab 4 has multiple sites which serve it, and it is impossible to predict which one will be accessed by the container. For this reason, SEAMS simply ensures that two containers are unable to talk to each other or to the host machine. They are, however, able to communicate with the wider internet. This provides the necessary isolation between container and host while also ensuring that two separate SEAMS instances can't communicate with each other.

4.2 Protection

Most courses will have some files provided by the staff that are important to the test suite. For instance, 6.858's lab 4 relies on reference images which are compared to images generated by the student code. To ensure that the student code can't modify these files, two users are created within the container. The student user is given ownership of the student code, while the staff user is given control of any files belonging to the staff. This declaration is done by the course staff in the configuration file. SEAMS also provides support for extra users based on the configuration file.

For labs where the student needs to be given write access to important files such as logs, or cases where the students need to be given root access, SEAMS also allows staff to declare a set of protected files. These protected files are marked as immutable (or append only for directories) by using ext4's extended file attributes[2], which prevents even the root user on the container from deleting, modifying, or renaming them. This is possible because the root user on an unprivileged LXC container maps to a non-root user on the host, which means they can't modify the i or a bits on files. This is particularly valuable in cases like lab 4,

[1]`https://linuxcontainers.org/`

[2]`https://linux.die.net/man/1/chattr`

where students need to put some images in the same directory as the reference ones, as it is important that student code cannot move or delete files in that directory.

4.3 Authenticity

Verifying the authenticity of student results is necessary but challenging. In order to comprehensively do so, course staff are assumed to be capable of writing good test cases that have verifiable side effects, rather than just reading from stdout. If reading from stdout is necessary, SEAMS relies on course staff to randomize the input or to include hidden test cases.

SEAMS monitors the resource usage of the grading scripts using rusage[3] and compares those results with the results reported by a staff solution. Large discrepancies between the two may signal some bad behavior on the part of the student code, and SEAMS will alert the staff in these cases.

4.4 Status and Future Work

SEAMS is fully capable of setting up and managing the LXC containers in which student code executes. It correctly configures the file permissions, firewall rules, and container users to ensure proper isolation and protection of staff resources. It monitors the resource usage of the grading scripts, and correctly captures, scores, and reports the results of grading. I've tested this with a variety of smaller python labs from 6.009 and 6.034, as well as labs 3 and 4 from 6.858 which are more complex (multiple processes, network access, many programming languages, system configuration changes, etc). SEAMS also successfully prevents the attack from section 3. Getting the labs to work with SEAMS required little to no change in the actual grading scripts. SEAMS consists of around 600 lines of python scripts along with some extra configuration files. Developing SEAMS required a large amount of time devoted to learning about the inner workings of lxc/ipfw, examining the grading schemes used by different classes, and ensuring compatibility with these schemes.

There are a number of things I would like to implement to improve SEAMS. First is a streamlining of the configuration process, as at the moment it requires running a number of different python and bash scripts. Second, using LXD on top of LXC to improve the network configuration process, hopefully allowing for more fine-grained network control. Ideally, course staff would be able to declare a set of allowed URLs and the containers would only be able to communicate with addresses from this set. Finally, I would like to explore a more comprehensive solution to the grading authenticity problem. Some ideas include injecting some kind of secret into the student code to verify that it runs to completion, or writing the output of the tests to a more secure location than stdout.

5 Acknowlegements

Special thanks to Jon Gjengset for his advice and guidance on this project.

[3] `http://man7.org/linux/man-pages/man2/getrusage.2.html`

6 Appendix A: Sample Configuration SEAMS Configuration File

```
{
    "users": {
        "student": ["student", false],
        "coursestaff": ["coursestaff", true]
    },
    "staff_user": "coursestaff",
    "student_user": "student",

    "code_dir": "858-4",

    "student_files": [
        "http.c",
        "http.h",
        "index.html",
        "zook.conf",
        "zookd.c",
        "zookfs.c",
        "zookld.c",
        "answer-1.js",
        "answer-2.js",
        "answer-3.txt",
        "answer-4.txt",
        "answer-5.txt",
        "answer-6.html",
        "answer-7.html",
        "answer-8.html",
        "answer-9.html",
        "answer-10.html",
        "answer-11.html",
        "answer-12.html",
        "answer-13.html",
        "answer-14.txt"
    ],

    "grading_command": "make check",

    "setup": "setup.sh",

    "executables": [
        "check-lab4.sh",
        "clean-env.sh",
        "get-phantomjs.sh",
        "zookld",
        "zookfs",
        "zookd"
    ],

    "protected_files": [
        "check-lab4.sh",
        "clean-env.sh",
        "favicon.ico",
        "Makefile",
        "z_client.py",
        "LICENSE",
        "zoobar/",
        "get-phantomjs.sh",
        "lab4-tests/",
        "zookld",
        "zookfs",
        "zookd"
    ]
}
```