# Security Analysis of CAT-SOOP

Sooraj Boominathan, Saroja Erabelli, Alap Sahoo, Samyu Yagati
Massachusetts Institute of Technology

May 16, 2018

**Abstract**

CAT-SOOP is an educational platform used for many introductory computer science classes at MIT. The platform stores a great deal of confidential information, including homework solutions and student grades, so the integrity of several MIT courses depends on CAT-SOOP being a secure system. In this paper, we discuss CAT-SOOP's security policy and explore a variety of attacks aimed at finding vulnerabilities in the system. We provide an analysis of the secure and insecure elements of CAT-SOOP with respect to the system's confidentiality, integrity, and availability and provide recommendations for how CAT-SOOP can improve its security in these three areas.

## 1 Introduction

CAT-SOOP is an educational software tool for automatically collecting and grading student work [1]. It was originally developed by Adam Hartz for use in MIT's 6.01, but has since been utilized by over a dozen computer science and engineering classes at MIT and Olin College. CAT-SOOP has a wide array of functionalities and is used for multiple purposes by different courses: storing student grades and records, hosting tutorials and exercises, grading code submissions, etc. As a result, CAT-SOOP as a platform is accessed by a range of different users (e.g. professors, TAs/LAs, graders, students) and stores significant amounts of sensitive data. Furthermore, as a tool for assigning student grades, CAT-SOOP is responsible for maintaining the academic integrity of the courses that use it. Finally, CAT-SOOP should be readily available to students, since it is often the centralized location of both learning and grading; students should not have to worry about the site going down an hour before they have to submit an assignment. Therefore, the security of CAT-SOOP should be an important aspect of its design.

The goal of this project is to test the security of the CAT-SOOP platform and identify any vulnerabilities present in the system. In particular, we want to examine the confidentiality, integrity, and availability of the system. Our approach focuses on gaining access to unauthorized information, such as instructor solutions, manipulating problem set and test scores either directly or through fraudulent submissions, and testing CATSOOP's capacity to tolerate long-running submissions or a large number of requests. We hope to find any vulnerabilities that may exist in CAT-SOOP and provide recommendations on patching them, so that CAT-SOOP can continue to be a trustworthy academic software.

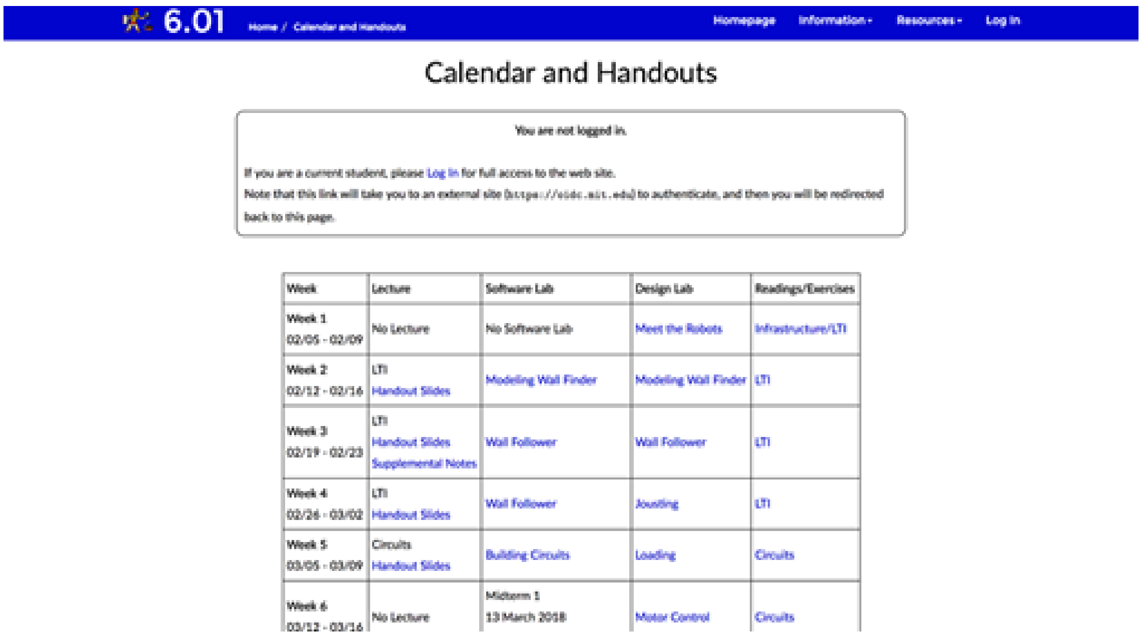
## 2 Responsible Disclosure

Before attempting any attacks on CAT-SOOP, we contacted Adam Hartz to receive his permission to look for vulnerabilities in the system. He kindly gave us permission to do so. We agreed to only test our attacks on a test CAT-SOOP instance, and did not attempt to attack any of the CAT-SOOP instances that were running courses in the Spring 2018 semester. We also did not attempt to carry out any attacks that would allow us to gain access to actual confidential student data for current students.

We also agreed to immediately report any sensitive vulnerabilities that we found to Adam so that he could rectify them and preserve the integrity of grading and assignment submission in classes that currently use CAT-SOOP. We have notified Adam of all the vulnerabilities that we describe in the report below to allow him to patch the issues as he deems necessary.

## 3 Platform Overview

CAT-SOOP primarily serves as an educational and evaluative tool, and its design reflects this purpose. The primary student facing components are focused on course content - syllabi, schedules, lecture slides, grades, and assignments. Access is dictated by a user login (at MIT, usually via our Kerberos certificates), and individual classes have a lot of flexibility to dictate which aspects of the CAT-SOOP interface they utilize. For example, 6.01, the class CAT-SOOP was originally used for, uses the site to host all aspects of the course, from test scores to administrative details to homework submissions, shown in Figure 1, while 6.006 only utilizes CAT-SOOP's Python code-checker, and hosts the bulk of its course material elsewhere.



Calendar and Handouts

You are not logged in.

If you are a current student, please Log In for full access to the web site.
Note that this link will take you to an external site (https://oidc.mit.edu) to authenticate, and then you will be redirected back to this page.

| Week | Lecture | Software Lab | Design Lab | Readings/Exercises |
|---|---|---|---|---|
| Week 1 02/05 - 02/09 | No Lecture | No Software Lab | Meet the Robots | Infrastructure/LTI |
| Week 2 02/12 - 02/16 | LTI Handout Slides | Modeling Wall Finder | Modeling Wall Finder | LTI |
| Week 3 02/19 - 02/23 | LTI Handout Slides Supplemental Notes | Wall Follower | Wall Follower | LTI |
| Week 4 02/26 - 03/02 | LTI Handout Slides | Wall Follower | Jousting | LTI |
| Week 5 03/05 - 03/09 | Circuits Handout Slides | Building Circuits | Loading | Circuits |
| Week 6 03/12 - 03/16 | No Lecture | Midterm 1 13 March 2018 | Motor Control | Circuits |

Figure 1: The 6.01 website contains administrative details and homework submissions.

Each CAT-SOOP instance is hosted by a server instance owned and used by the user. The instance responds to user requests and stores all user and course data. A unique feature of CAT-SOOP is the ability to run and test Python code submissions. Classes can choose to utilize their own code submissions, or redirect them to be run on the default CAT-SOOP server.

The most important aspects of the architecture, from a security perspective, are user information, potentially classified class content, the problem grading tools, and website availability. A particularly vulnerable aspect of the CAT-SOOP system is the Python auto-grader, since it essentially allows students to run their own code in a shared interface with course materials, even if through a sandbox. We also examined security vulnerabilities tied to how CAT-SOOP was used, in particular emphasizing efforts to overload the web server and code checker, as well as attempting to compromise CAT-SOOP's login system, since much of the website's security is due to segregation of permissions.

## 4 Security Policy

In this section, we present an overview of CAT-SOOP's security policy, including the principals involved in this system and the permissions given to each group. We then provide a high-level description of how an adversary might design attacks to find vulnerabilities in the system and violate the security policy.

### 4.1 Principals and Permissions

There are two main groups that interact with CAT-SOOP: students and instructors. Within the group of instructors, there may be several levels of hierarchy with differing levels of authority, including TA's, LA's, and graders. The permissions given to each of these groups are determined by 8 bits that control the following:

- "View" bit - allows a user to view the course materials after a specified release date
- "View Always" bit - allows a user to permanently view course materials, regardless of when the material was scheduled to be released
- "Submit" bit - allows a user to submit assignments within the specified submission limit and due dates
- "Submit Always" bit - allows user to submit assignments without any constraints
- "Grade" bit - allows a user to impersonate other users and edit the scores of other users

These permission bits encompass the types of actions that a CAT-SOOP user can take and provides a well-defined basis for its security policy. In a standard setting, students would only have permission to "view" and "submit" assignments and should not be allowed to take any of the actions allowed by the other bits. The course's primary instructors (i.e, the professors) should have all of these permission bits set and have full control over their particular CAT-SOOP instance. The other members of the course staff may have differing levels of authority that vary on a class-by-class basis. In some classes, TA's may be allowed to edit materials, while in others, they are limited to only grading and viewing the existing course materials. For our project, we used the default settings for our personal CAT-SOOP instance.

### 4.2 Attack Model

We focus on attacking the three fundamental aspects of CAT-SOOP's security: confidentiality, integrity, and availability. While outsiders may stand to gain from accessing student information or course content, students themselves have far more incentive to attempt to compromise the security of CAT-SOOP, since they stand to gain significant academic advantage from subverting the system. Therefore, our attack model emphasizes attacks that would be advantageous to a student and undetectable as cheating to an instructor.

CAT-SOOP stores sensitive data like student grades, contact information, unreleased class content, and solutions, and a well-implemented confidentiality policy is vital in ensuring that the information is not misused. Our attacks primarily address acquiring unreleased class content and solutions, as these would be the most useful to students enrolled in a CAT-SOOP course. In addition, we focused on viewing data that was hidden from our student test accounts, rather than attempting to directly modify records; cheating with modified records would be detected by a diligent instructor, while the simply accessing hidden materials would be hard to identify externally as cheating.

Next, we consider data integrity. As mentioned above, we did not focus on directly modifying records. By data integrity, we mean that student submissions are (1) as the student intended, and (2) an accurate reflection of the student's own work. For example, students should not be able to submit on another student's behalf. They should also not be able to view, copy, and submit instructor solutions, a goal that overlaps with confidentiality. These guarantees are important, especially since many students, and therefore many submissions, are generated for any given problem, and many CAT-SOOP courses have strong restrictions surrounding collaboration.

Finally, we address availability. CAT-SOOP is primarily used for auto-grading and assignment submissions. In addition, these submissions can be time-sensitive, as many assignments have open windows ranging from fifteen minutes (e.g. quizzes) to one week (e.g. labs). Therefore, we focused on the availability of the grading and submission system, as those are the most heavily-used parts of the system. They are also the most likely to be exploited by students, given that they are directly related to recorded grades: a student might, for example, want to delay a test for the whole class by preventing any individual from submitting.

## 5 Summary of Security Attacks

### 5.1 Confidentiality: Python Code Submission Vulnerabilities

In this section, we outline our attempts to find vulnerabilities in the confidentiality aspects of CAT-SOOP's security policy. We focus primarily on using Python code submission problems to carry out various exploits on the system. We first provide some background information about the Python code submission framework in CAT-SOOP, then describe ways in which CAT-SOOP effectively defends against potential attacks via code submission. We then describe two significant vulnerabilities that we found in the system that would provide students with access to unauthorized materials: hidden test cases and problem solutions.

#### 5.1.1 CAT-SOOP's Code Submission Framework

One of CAT-SOOP's most useful functionalities is its ability to automate the evaluation of student-written computer programs. CAT-SOOP is currently only equipped to handle programs written in the Python programming language, which is the language of choice in almost all introductory computer science classes at MIT. Instructors can allow students to submit their code to CAT-SOOP by either directly uploading a Python file or by entering the code into a in-browser coding environment, such as those in Figure 2.

When a student submits a code file to a particular problem, the file is sent to the server, where a new directory containing only that file is created. The code is then executed and evaluated for correctness using a suite of test cases that a course instructor specifies for the problem. Instructors can also provide a block of code that should be executed before the student's code is evaluated if specific functions and variables need to be defined before their submission can be run.

CAT-SOOP is designed to only provide very simple feedback to students about the result of their code submission. In general, it displays an output console that contains the output of any 'print' statements that a student includes in their code. It also indicates whether or not a student passed each test case for the problem.

CAT-SOOP's design does acknowledge the fact that allowing students to submit arbitrary Python code could potentially be very dangerous and takes some measures to ensure that malicious code will not be executed. First, CAT-SOOP defines a "blacklist" containing statements that are prohibited in student code. CAT-SOOP checks for these statements after stripping away all special characters from the student submission, so a student cannot use clever formatting tricks to insert these blacklisted statements. Any code containing these blacklisted statements is sent to the course administrators with information regarding the student who submitted the malicious code.

CAT-SOOP also provides an option for users to sandbox Python code submissions [2]. By default, sandboxed Python code is sent to a central server at `catsoop.mit.edu`; users also have the option to set up a local sandbox on their server for convenience. Ideally, this sandboxing would prevent students from gaining access to any part of the filesystem where the code is being executed.

CAT-SOOP relies on the fact that these two features are sufficient to prevent users from doing anything malicious to the system using code submission problems.

### 5.1.2 Secure Elements

Our general approach to attacks with code submission problems involved using Python's `os` module to perform actions on the filesystem where a student's code submission is compiled and executed. We wanted to see if CAT-SOOP prevents students from performing any unauthorized actions the server. We found that CAT-SOOP does a good job of preventing students from executing certain dangerous commands. For instance, the `os.system()` function can be used to pass arbitrary commands to the server's shell for execution. However, CAT-SOOP ensures that such function calls return a failure code of -1, preventing students from accessing the shell and directly performing malicious actions such as deleting files from the server or gaining access to directories that might contain confidential student data.

CAT-SOOP also ensures that students cannot write to any files on the server. For instance, we attempted to overwrite parts of the file that contains the student's code submission, but were met with a `File Too Large` error. This ensures that a student cannot overwrite required test cases and trick the server into accepting an incorrect piece of code as a correct submission.

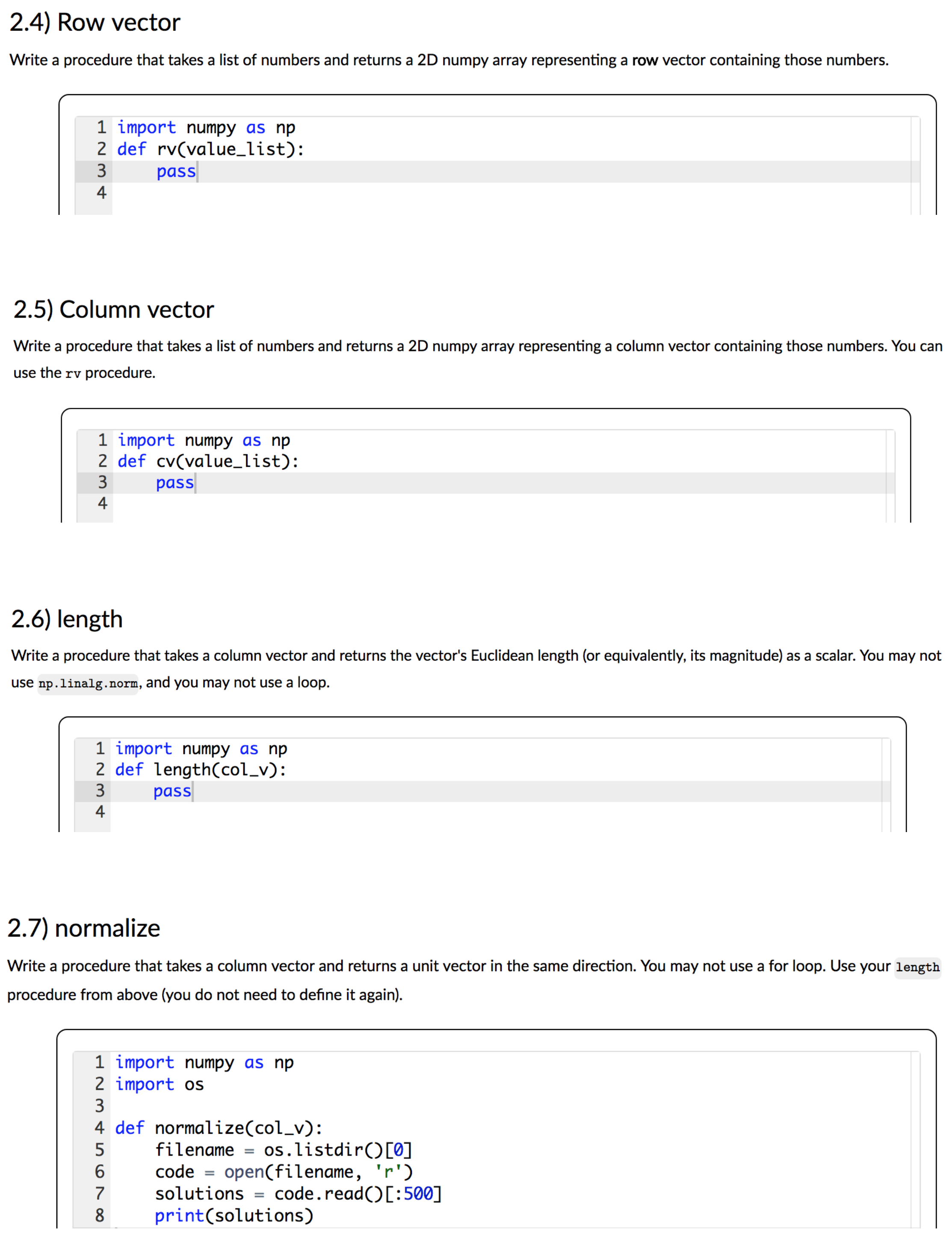
Finally, CAT-SOOP ensures that a user cannot even read the contents of higher-level directories in the server, as this may provide some crucial information about where confidential student records are stored on the server. In particular, we attempted to use the `os.chdir()` and `os.listdir()` function on the root directory of our test server to view the overall layout of the filesystem. However, we were denied permission to execute these commands. Overall, CAT-SOOP does a reasonably good job of ensuring that a student cannot interact with many items on the filesystem other than the directory containing their submitted code.

### 5.1.3 Vulnerabilities

Despite these security measures, we were able to find a few issues that violate the desired confidentiality standards that CAT-SOOP should provide. First, we found that we were able to create new files outside the directory on the server containing the student's code submission file. We were unable to write anything to these files, but it is undesirable to allow a student to have any form of write access outside of the submission directory - there is no reason why a student would need to directly create a file on other parts of the CAT-SOOP server. We were also able to call `os.listdir()` inside the directory containing the student's code submission and were able to successfully find the name of the submission file. We were able to use the ability to execute these commands to construct the two exploits described below.

***Accessing staff solutions.*** The first vulnerability allows students to gain unauthorized access to staff solutions for a particular problem type that commonly occurs in CAT-SOOP based courses. As described in section 5.1.1, instructors can provide code that should be executed on the server before the student's code submission is evaluated. In some cases, this code includes functions that students should have implemented in earlier parts of the assignment. This is a particularly common pattern in many CAT-SOOP assignments, as students progress from implementing simpler functions to more complex functions that make calls to the simpler functions. Students can use the vulnerability we discovered to directly acquire solutions to the earlier functions and successfully submit correct answers to these problems.

We present an example of this type of problem structure, the code used to acquire the solutions, and the resulting output in Figures 2 and 3. The exploit first reads in the name of the file containing the student's code submission on the server using a call to `os.listdir()`. This file is then opened and read, then the first 500 lines of the file are printed out to the student output console. None of these operations are prohibited by CAT-SOOP. A student can then see the first 500 lines of the file of the modified code that he submitted. In this particular problem, this section of the code file contains solutions to three previous problems in the assignment.



## 2.4) Row vector

Write a procedure that takes a list of numbers and returns a 2D numpy array representing a **row** vector containing those numbers.

```python
import numpy as np
def rv(value_list):
    pass
```

## 2.5) Column vector

Write a procedure that takes a list of numbers and returns a 2D numpy array representing a column vector containing those numbers. You can use the `rv` procedure.

```python
import numpy as np
def cv(value_list):
    pass
```

## 2.6) length

Write a procedure that takes a column vector and returns the vector's Euclidean length (or equivalently, its magnitude) as a scalar. You may not use `np.linalg.norm`, and you may not use a loop.

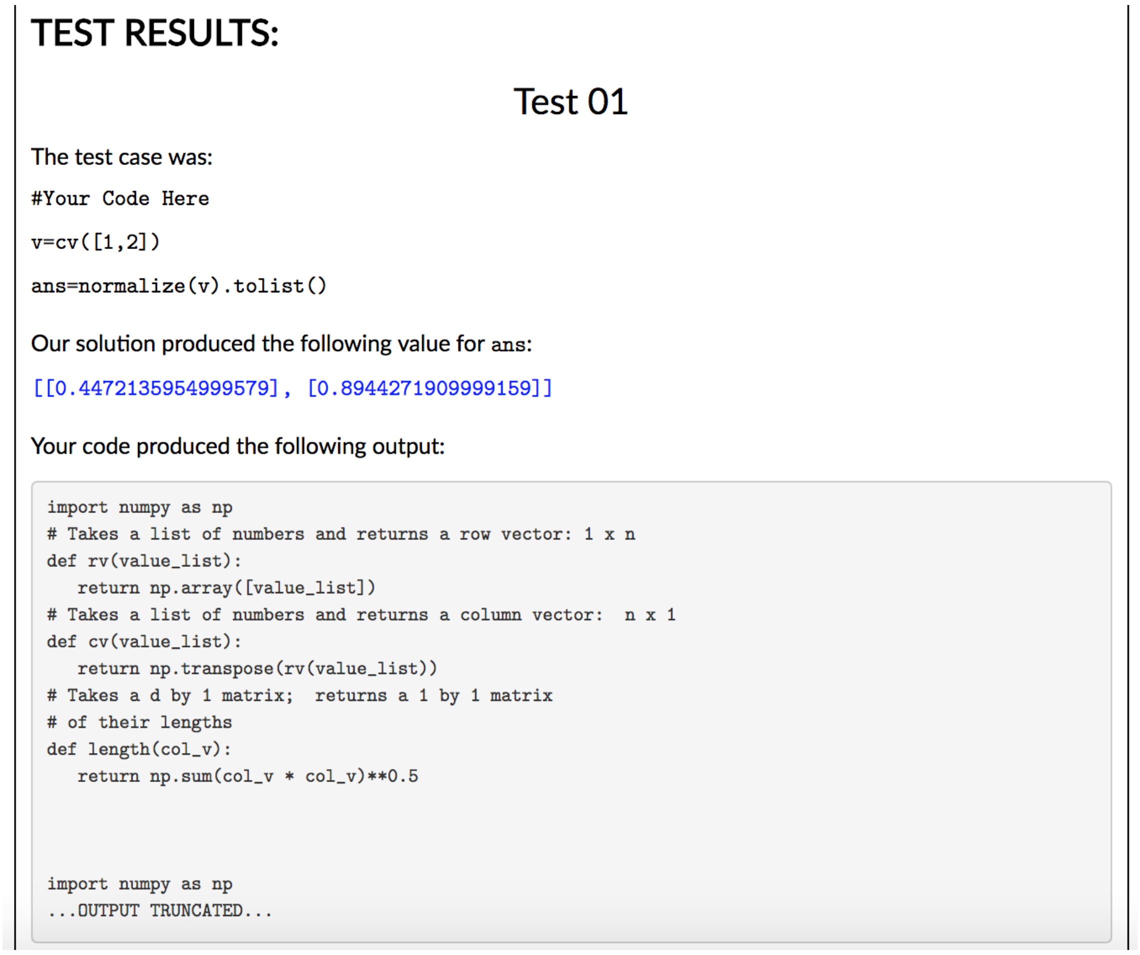
```python
import numpy as np
def length(col_v):
    pass
```

## 2.7) normalize

Write a procedure that takes a column vector and returns a unit vector in the same direction. You may not use a for loop. Use your `length` procedure from above (you do not need to define it again).

```python
import numpy as np
import os

def normalize(col_v):
    filename = os.listdir()[0]
    code = open(filename, 'r')
    solutions = code.read()[:500]
    print(solutions)
```

Figure 2: In this exploit, we acquire solutions to the first three problems shown in this figure using a submission to the last problem.If a student was to implement the final problem correctly, they would make use of the functions implemented in the previous problems


TEST RESULTS:

Test 01

The test case was:

```
#Your Code Here

v=cv([1,2])

ans=normalize(v).tolist()
```

Our solution produced the following value for `ans`:

```
[[0.4472135954999579], [0.8944271909999159]]
```

Your code produced the following output:

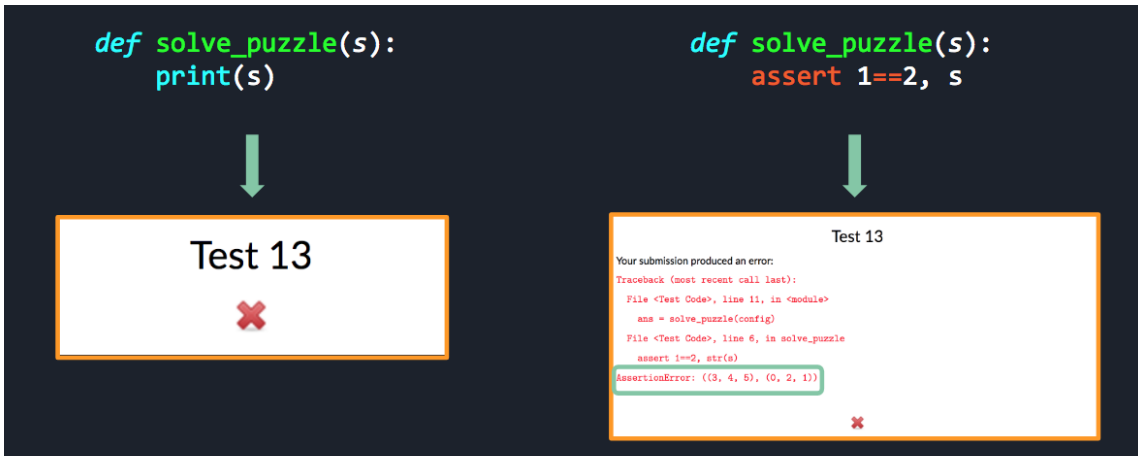
```
import numpy as np
# Takes a list of numbers and returns a row vector: 1 x n
def rv(value_list):
   return np.array([value_list])
# Takes a list of numbers and returns a column vector:  n x 1
def cv(value_list):
   return np.transpose(rv(value_list))
# Takes a d by 1 matrix;  returns a 1 by 1 matrix
# of their lengths
def length(col_v):
   return np.sum(col_v * col_v)**0.5



import numpy as np
...OUTPUT TRUNCATED...
```

Figure 3: This is the result of submitting the code from the last image in Figure 2. The output on the student console contains the staff implementations of the `rv`, `cv`, and `length` functions.

In general, a student can simply search through the code file to find the lines containing the name of a function that the current problem depends on and print out the surrounding lines to obtain the staff implementation for that function. Students should not be allowed to access these solutions, so this is a clear violation of CAT-SOOP's confidentiality policy.

***Accessing hidden test cases.*** The second vulnerability allows students to print out hidden test cases for a given problem. In many classes, instructors may not want students to view test cases so they cannot hard-code correct outputs to each test case and receive full credit for a problem. In many cases, the input to the function that the student should implement is the test case. Instructors can prevent students from viewing the value of this input by suppressing all `print` statements, so that nothing can be displayed on the student output console (unlike in the previous vulnerability).

However, we found that students can still obtain the input value by using an `assert` statement instead of `print`, as CAT-SOOP does not suppress a submission's error output. If the student asserts an expression that is guaranteed to be false, they are able to print out the value of the function's input. Figure 4 provides an example of this vulnerability. Both of these vulnerabilities are relatively significant and provide students with access to information that instructors would not want students to see.



```
def solve_puzzle(s):
    print(s)
```

Test 13

```
def solve_puzzle(s):
    assert 1==2, s
```

Test 13

Your submission produced an error:

```
Traceback (most recent call last):
  File <Test Code>, line 11, in <module>
    ans = solve_puzzle(config)
  File <Test Code>, line 6, in solve_puzzle
    assert 1==2, str(s)
AssertionError: ((3, 4, 5), (0, 2, 1))
```

Figure 4: The left image shows an unsuccessful attempt to directly print a test case to the student output console. The right image shows a successful acquisition of a hidden test case using an assert statement that is guaranteed to fail.

## 5.2 Integrity: Authentication Attacks

In this section, we detail our efforts to compromise CAT-SOOP's integrity policy via attacks on its authentication mechanisms. First, we provide an overview of CAT-SOOP's authentication procedure. Next, we discuss the elements of CAT-SOOP's authentication procedures that defend against possible attacks. Last, we outline two successful attacks that would allow an attacker to force user submissions.

### 5.2.1 CAT-SOOP's Authentication Procedure

Like many websites, CAT-SOOP uses a username/password combination to authenticate (or "log in") users. It also allows users to log in with certificates corresponding to their usernames, but the attacks in this section do not rely on this login method. Furthermore, most modern websites use cookies to preserve information about a session, where a session generally begins when a given user logs in and ends when that user logs out. Using cookies allows the user to conduct multiple actions without authenticating for each one. CAT-SOOP, however, does not use cookies. Instead, it preserves a session ID, and associates sessions with API tokens. Website backends usually generate these tokens on a per-user, per-session basis and use them for authentication in lieu of a username/password pair.

### 5.2.2 Secure Elements

The following attacks were conducted unsuccessfully on CAT-SOOP, suggesting that the system is properly secured against them.

***Code injection attacks.*** Code injection attacks comprise a set of attacks that allow an attacker to control some some portion of the server-side application by inserting code into a webpage (e.g. by modifying the URL, adding text to form responses, or directly modifying the HTML) [3]. We attempted a few basic code injection attacks. First, we checked that HTML variables are correctly escaped by entering faulty parameter values directly in the URL. Next, we verified that CAT-SOOP does not use any cookies with a manual inspection of the code. We were unable to successfully insert Javascript into the page's HTML.

***Compromised password database.*** CAT-SOOP follows current best practices in regards to its password database, as it stores only the hash of the salted password. Because it uses a salt, CAT-SOOP is not vulnerable to a rainbow table attack.

***Clickjacking attack.*** In a clickjacking attack, the attacker embeds the user's view of a page within a transparent iframe, allowing him to seamlessly overlay his own buttons and links over the legitimate ones on the page [4]. One common defense against such attacks is the use of the X-Frame-Options HTTP header, which is honored by most modern browsers. If the X-Frame-Options header for a page is set to `DENY`, then the browser will not display that page in a frame [5]. During our testing, we sent a variety of HTTP requests to the CAT-SOOP instance and the headers for all of these requests had X-Frame-Options set to `DENY`. Therefore, CAT-SOOP users are unlikely to be victims of clickjacking.

### 5.2.3 Vulnerabilities

While Hartz certainly considered many aspects of authentication in designing CAT-SOOP, the implementation of API tokens left some vulnerabilities that we were able to exploit. In particular, we were able to trick a logged-in user into submitting unintentionally. In CAT-SOOP, many questions have a limited number of submissions, and a large subset of these allow only a single submission (e.g. on quizzes). However, instructors can grant additional submissions. A student who has used up his submissions could send this phishing email to many classmates in an effort to force them to use up submissions, potentially causing the instructor to reset submission limits for the whole class. In addition, given an API token corresponding to a particular user session, we were able to make requests for the same session even after the user logged out. This is less dangerous, because the attacker must first acquire individual API tokens as opposed to mass-emailing, but nonetheless poses a potential issue.

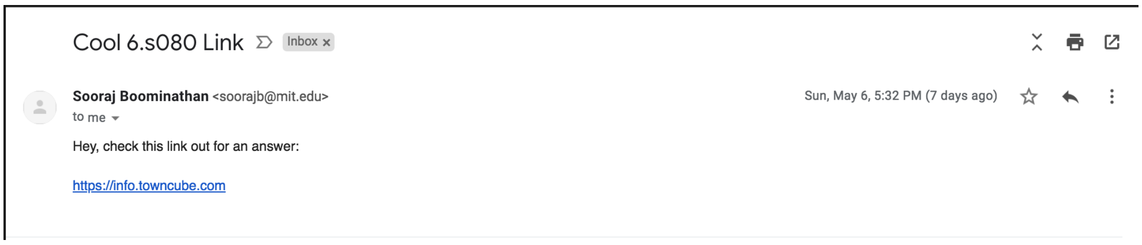
***Submitting for a logged-in user.*** The first attack can be carried out in two steps. First, the attacker devises a URL of his choosing. CAT-SOOP URLs explicitly contain the assignment, question number, action (e.g. submit), and data (e.g. the content of the actual submission), so it is simple to craft a URL corresponding to a particular submission (Figure 5). In the example shown in Figure 5, visiting the URL will submit Question 1 for Assignment 1 in course 6.s080, and the submission content is `{"q000001":  "import os\n def fourth_powr(x):\n "}`. Next, the attacker tricks the victim into clicking this URL. Our attack used a phishing-style email (shown in Figure 6). Many CAT-SOOP users do not log out of their accounts when they are not actively using them, largely for convenience, and it is likely that the user would be logged in when clicking on the provided link.

This attack works because, although CAT-SOOP maintains an API token for authentication, it does not actually check the token while the user's session is active. Since the API token is never checked, we were able to insert any string in place of the API token in our URL (see Figure 5) without causing any errors. Furthermore, because CAT-SOOP does not check the API token, it relies on the session information to get the user's identity, and associates all actions during a user session with that user. As a result, a user clicking on the link shown in Figure 5 while logged in would result in CAT-SOOP logging that that user submitted to Question 1 for Assignment 1.

***Submitting with an API token.*** In the second attack, we assume that we have acquired a user's API token. For the purposes of testing, we used API tokens from our own user sessions. While not logged in, we submitted HTTP requests to the instance via the command line, hardcoding the API token into the request. Rather than rejecting a request that it received while the user was not logged in, the server verified the credentials in the API token and successfully completed the action, allowing us to artificially extend the session.

```
https://towncube.mit.edu/6.s080/assignment1.0/square?
action=submit&names=%5B%22q000001%22%5D&api_token=dummy&
data=%7B%22q000001%22%3A%22import+os%5Cndef+fourth_powr(
x)%3A%5Cn++++%22%7D
```

Figure 5: A placeholder for the API token can be directly inserted into a submission URL, in this example a short code snippet for Question 1 in Assignment 1.


Cool 6.s080 Link

Sooraj Boominathan <soorajb@mit.edu>
to me

Sun, May 6, 5:32 PM (7 days ago)

Hey, check this link out for an answer:

https://info.towncube.com

Figure 6: An example "phishing" email that could be used to trick a user into submitting unintentionally. The link in this email actually goes to the link shown in Figure 5.

## 5.3 Availability Attacks

In this section, we describe our attempts to compromise CAT-SOOP's availability. Because one of CAT-SOOP's primary functions is as an automated grader, we focused on the availability of its code submission framework, rather than the availability of individual pages. We first provide some background on the submission queue, and then describe the security measures taken by CAT-SOOP. Finally, we outline two successful attacks that would affect the availability of the grading system.

### 5.3.1 CAT-SOOP's Submission Queue

As explained in section 5.1.1, CAT-SOOP's automatic grader is one of its most heavily-used functionalities. Currently, each CAT-SOOP instance maintains a queue of submissions, and executes the submissions in queued order in the framework detailed in 5.1.1. It is important to note that each instance maintains a single queue to which all problem submissions are appended, so submissions from different questions can delay each other's execution.

### 5.3.2 Secure Elements

Since the submission queue is shared by all users of a given CAT-SOOP instance, it is important that no single code submission takes a very long time to process. One example of long-running submissions is code containing an infinite loop. Since CAT-SOOP instances are used for courses taught for beginning programmers, infinite loop submissions are not necessarily malicious. However, students could potentially submit infinite loop scripts to delay processing other's submissions, for example to prevent submissions to timed quizzes. CAT-SOOP's design does include a timeout feature that cuts execution off after around two minutes, preventing accidental or one-off blocking of the submission queue.

### 5.3.3 Vulnerabilities

However, the timeout alone does not prevent malicious denial-of-service (DoS) attacks, as proven by the two attacks described in this section. We first developed an attack in which a single adversary could prevent all students in a class from receiving feedback in a timely manner to any problem submission. We were able to write a script using the Selenium Python package that would automatically open a browser session, navigate to the desired CAT-SOOP instance, and repeatedly upload a file containing an infinite loop to a selected problem on the site. The code for this attack is included in the Appendix.

***Single adversary blocking queue.*** We found that we were able to successfully block the queue for most users. If a user enters the queue when there are multiple submissions of the malicious file already in the queue, they will have to wait several minutes before receiving feedback on their submission. Moreover, the single submission queue architecture prevents students from submitting to any problem on the site in a timely manner, not just the one that the attacker is targeting. This attack renders the site nearly unusable for most students.

***Tricking multiple users into blocking queue.*** We were also able to develop a second attack based on the phishing attack described in section 5.2.3, in which the attacker tricked a CAT-SOOP user into submitting code to a problem. The code for the submission is included in the URL provided to the victim, and the attacker could easily make this code an infinite loop (e.g. `while True:\n print()`). By emailing the link to a group of users of an instance at a critical time, such as the beginning of a timed quiz, an attacker could cause multiple users to submit two-minute-running submissions to the queue and prevent receiving timely feedback to the quiz. While CAT-SOOP does not allow students to view the enrollment for a class, students can find email addresses of other students in their section. Since quizzes administered on CAT-SOOP are often fifteen minutes long, seven or eight clicks could pose a serious problem. Overall, we found that CAT-SOOP should take more stringent measures to ensure that the availability of a course site is more robust against these types of attacks.

## 6 Recommendations

### 6.1 Confidentiality

We conducted two successful attacks that breached CAT-SOOP's confidentiality, namely accessing hidden test cases and accessing hidden instructor solutions. In exercises with hidden test cases, normal printing to console is already disabled. We suggest suppressing error output as well in those exercises in order to prevent the illustrated misuse of Python's `assert` statement. Since students are expected to debug and test locally, rather than on the CAT-SOOP instance, we do not believe that suppressing error output would impede the site's functionality.

To prevent accessing hidden instructor solutions via the use of Python's `os` module, we recommend preventing imports of the `os` module. While it can be difficult to create an in-depth defense against using undesirable modules, there are some workarounds. The simplest way is to manually override the import in the sandbox code, for example with `sys.modules['os']=None`, or to use an import blacklist, which is provided by the Python sandbox that CAT-SOOP uses.

### 6.2 Integrity

While CAT-SOOP does take many basic precautions to prevent breaches of its integrity policy, its particular use of API tokens leaves it vulnerable to attack. In section 5.2.3, we presented two successful attacks on CAT-SOOP's authentication system. The first was possible because the system does not currently check the API token when the user is logged in, and the simplest solution would be to check both the logged-in state and that the logged-in user's credentials match the API token.

The second attack relied on API tokens extending beyond a particular session. Again, our recommended fix is simple: API tokens should expire as soon as a session ends, and be refreshed upon beginning a new session. While CAT-SOOP currently refreshes tokens periodically, it does not do so in coordination with sessions, which caused the vulnerability described in 5.2.3.

### 6.3 Availability

There are a few different mechanisms that CAT-SOOP could implement in order to prevent the availability attacks described in Section 5.3.3. The first attack presented in Section 5.3.3 involves a single malicious user issuing a very high number of submission requests from a single computer. If CAT-SOOP were to track the IP address for each code submission, it would be possible to detect unusually high submission rates from a single individual and limit their ability to submit problems to the site.

We also recommend that CAT-SOOP should add multiple code submission queues for every course and provide some form of load-balancing between these queues. It is still possible that a single adversary would still have enough resources to clog up all the submission queues, but it would make it much more difficult for them to render the site entirely unusable.

However, neither of these defenses would be particularly effective against the second availability attack that we presented. If the adversary tricks numerous students in the class into making a malicious code submission, then all the requests would arrive from different IP addresses and not exhibit any unusually high submission rates that CAT-SOOP could detect. It is also more likely that simultaneous malicious submissions from several users would block multiple code submission queues. However, if CAT-SOOP resolves its authentication vulnerabilities with the recommendations we present in Section 6.2, this attack is no longer a viable option.

## 7 Conclusion

Our analysis of CAT-SOOP found that the system's security has a few significant vulnerabilities that could negatively impact the integrity of the classes it is used in. In particular, we believe that the confidentiality exploits discussed in Section 5.1.3 could be easily used by students to artificially improve their grade in a class without truly understanding the material. Since one of CAT-SOOP's primary goals is to provide an interface that assists students in gaining a meaningful understanding of course material, resolving these vulnerabilities is crucial to helping CAT-SOOP continue to achieve its purpose.

We also found issues with CAT-SOOP's API token system that could allow a malicious actor to submit requests to a course website on behalf of other students and potentially have an adverse effect on their class performance. CAT-SOOP also does not protect against the possibility of a single student uploading malicious scripts that prevents other students from submitting their assignments and receiving feedback in a timely manner. Both of these issues are less likely to be exploited by students compared to those described in the previous paragraph, but they still represent potential threats to the system's security.

Despite these issues, CAT-SOOP does a good job of protecting against a wide range of other dangerous attacks, such as arbitrary remote code execution, clickjacking, cross-site scripting (XSS) attacks, and Javascript code injection. We were not able to mount an attack that would allow a student to gain complete control over a CAT-SOOP instance or gain access to confidential student data. We conclude that CAT-SOOP does have a few security issues that it needs to resolve, but is still reasonably secure and a very effective platform for use in MIT classes.

## 8 Acknowledgements

We would like to thank Adam Hartz for allowing us to conduct a security analysis of CAT-SOOP, helping us set up an environment where we could carry out all our attacks, and clarifying any questions that we had about CAT-SOOP in general. Without his help, this project would not have been possible. Some of our attack strategies were also partly inspired by a security analysis of Gradescope conducted by a 6.857 project team from 2016 [6].

We would also like to thank the entire 6.857 course staff for a great semester, especially our TA Jonathan Frankle for providing guidance and feedback on our project and Profs. Rivest and Kalai for so many entertaining and informative lectures.

## References

[1] A. Hartz, "Cat-soop: A tool for automatic collection and assessment of homework exercises," 2012.

[2] A. Hartz, "Cat-soop v11.x documentation." `https://catsoop.mit.edu/website/docs`.

[3] "Code injection." `https://www.owasp.org/index.php/Code_Injection`, 2017.

[4] "Clickjacking." `https://www.owasp.org/index.php/Clickjacking`, 2017.

[5] "X-frame-options." `https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options`, 2018.

[6] D. W. D. Z. Steven Hao, Edward Park, "Security analysis of gradescope," 2016.

## Appendix

### Code for Obtaining Instructor Solutions

```
import os
def problem(input):
    filename = os.listdir()[0]
    with open(filename, 'r') as f:
        solutions = f.read()[:500]
        print(solutions)
```

### Script for Automated Python Code Submission

```
from selenium import webdriver
import time

driver = webdriver.Chrome('/Users/SoorajBoominathan/Downloads/chromedriver')
driver.get('https://towncube.mit.edu/6.s080/')
continue_link = driver.find_element_by_link_text('Log In')
continue_link.click()

# Login to the website
username_input = driver.find_element_by_id("login_uname")
pwd_input = driver.find_element_by_id("login_passwd")
login_submit = driver.find_element_by_id("loginform_submitter")

username_input.send_keys("soorajb")
pwd_input.send_keys("6857test")
login_submit.click()

# Locate the desired question to submit to
assignment_link = driver.find_element_by_partial_link_text("Primitive Types")
assignment_link.click()

problem_link = driver.find_element_by_partial_link_text("Zeller")
problem_link.click()

# Upload the same file 100 times with a timeout of 120 seconds between each upload.
for i in range(100):
    file_upload = driver.find_element_by_id("q000000")
    file_upload.send_keys("/Users/SoorajBoominathan/6.857/Final Project/catsooptest.py")

    file_submit = driver.find_element_by_id("q000000_submit")
    file_submit.click()
    time.sleep(120)

time.sleep(25)
driver.quit()
```