Security Analysis of CAT-SOOP: Codifying Security Practices for Executing Remotely-Generated Code

Alejandro Velez Jason Villanueva Rami Manna

Abstract

CAT-SOOP is a tool for automatic collection and assessment of online exercises. To assess the system's security, our team focused on attacks following a malicious-student-centered threat model executed by submitting malicious code disguised as assignment submissions to an anonymized instance of CAT-SOOP developed for 6.S080: A Brief Introduction to Programming in Python (IAP 2018). The instance proved resilient to many attacks attempted by our team, suggesting it to be a secure system. Our team believes the system has been designed following strong security practices allowing for the relatively safe execution of remotely-generated untrusted code. Codifying these practices will allow for other developers of tools executing remotely-generated code, for educational technology or other purposes, to have a reference for security practices. Our contribution in this report consists of analyzing and codifying these practices. We also suggest a security user study as a next step for fully analyzing and perhaps improving CAT-SOOP security.

1 Introduction

CAT-SOOP is a Python-based tool for automatic collection and assessment of online exercises, originally developed by Adam Hartz[1] primarily for use in Introduction to Electrical Engineering and Computer Science (6.01) course at Massachusetts Institute of Technology (MIT). Now CAT-SOOP is used by computer science courses at MIT as the major assignment submission webpage. On a given instance of a CAT-SOOP website, each personnel is provided appropriate access ranging from submitting online exercise answers to setting the students' grades. Depending on the need of each course, additional functionalities such as code test, help queue system, or logging (e.g. late day counts) are implemented by the instructors.

Several aspects of CAT-SOOP's purpose and implementation make it an interesting case study in cyber security. For starters, the tool's purpose as a assessment tool likely to be used in academic settings lent itself for us to design a malicious-student-centered threat model for analysis. CAT-SOOP sites can also accept student code submissions and must therefore be prepared to securely execute remotely-generated untrusted code. Lastly, the tool is also Python-based and is, therefore, exposed to the security vulnerabilities of the Python programming language.

Our team rigorously tested the CAT-SOOP online exercise tool, following a student-centered threat model in order to examine potential vulnerabilities exploitable by current and future students of courses using CAT-SOOP software. Our three main areas of focus were data security, service reliability, and admin privilege escalation. Penetration testing along these three areas was guided by the OWASP Testing Guide (available here: `https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents`) and an exploration of the CAT-SOOP source code (tool website: `https://catsoop.mit.edu/website`). Our team attempted attacks on an instance of CAT-SOOP provided and setup by the tool's developer, Adam Hartz (also lecturer for MIT's department of Electrical Engineering and Computer Science, website: `https://hz.mit.edu`). The instance was originally developed for use in the MIT course 6.S080: A

[1] MIT Lecturer, hz@mit.edu

Brief Introduction to Programming in Python. The instance contained several accounts with anonymized student data from the IAP 2018 offering of the course. Attacks were executed through accounts created by the team under student permissions following our student-centered threat model.

Following this model, our team focused its attacks on the submission of malicious code to the course assignment graders as we believe this to be the vector students would most likely follow if they were to attempt an attack on a CAT-SOOP instance. This project analyzes the security of CAT-SOOP and codifies the perceived best-practices implemented by the software's creator.

2 Significance

CAT-SOOP is used by 11 courses at MIT and 3 courses at Olin College. Better security can promote CAT-SOOP to be used by more organizations. Currently, CAT-SOOP is small enough that it is unlikely that hackers are actively trying to attack any of its instances. However, as it gains traction, it is important that we ensure CAT-SOOP is secure, both from students and the public, so that courses can run reliably and without inconsistencies. Furthermore, understanding CAT-SOOP's security strengths and weaknesses could improve understanding of security for similar systems.

3 Responsibility Disclosure

Our team received permission from CAT-SOOP's developer, Adam Hartz, to work on this project. We maintained communication with him and agreed to update him regarding any security issues discovered by the team. A directory traversal attack was successfully executed on the instance but the vulnerability turned out to be due to a misconfiguration on the specific instance and is not indicative of a larger security flaw in CAT-SOOP. This attack was communicated to Adam, and the configuration fixed by him, prior to this report. This report has also been shared with Adam Hartz.

4 Related Work

To our knowledge, there have been no other attempts at penetration testing or otherwise analyzing the security of the CAT-SOOP system. We are aware another student team has also worked on analyzing CAT-SOOP's security this year and hope our reports are complementary. Furthermore, sandboxing vulnerabilities, especially in Python, and Python privilege escalation are not particularly standardized domains and literature is fairly sparse. This work can serve as a guide to mitigating risks associated with executing remotely-generated code as well as potential sandboxing and privilege escalation vulnerabilities.

5 Security Policies and Threat Model

Though no security policy is officially listed, the purposes of the software allow for stating clear security goals which can shape such a policy. Specifically, the principals for a security policy of the system (students and course staff) and actions which should not be permissible (students being able to view others' submissions) are readily identifiable and allow for the design of a threat model the system can be evaluated against. As goals, CAT-SOOP should be able to protect information (such as student-submitted code or instructor-submitted grades) from improper viewing or modification. CAT-SOOP should also be resilient as a service and maintain availability.

5.1 Threat Model

To evaluate the CATSOOP system's ability to satisfy its security goals, our team acted under a malicious-student-centered threat model, acting as attackers under student privileges. Our threat model focused attacks on data security, service reliability, and admin privilege escalation. Under these goals, possible general attacks are described below.

5.1.1 Denial-of-Service

Malicious students could aim to consume site resources such that the service is brought down. An incentive for such an attack would be to force course staff to approve an extension on an upcoming assignment deadline.

5.1.2 Modifying grades and/or viewing others' assignment submissions

Malicious students are incentivized to adjust their grades to be higher than the grade they have achieved. If a malicious student is struggling with an assignment, they may attempt to view the submission of another student which has satisfied the assignment's requirements.

5.1.3 Privilege Escalation

A malicious student who has attained admin privileges could achieve all of the malicious results described above.

Our team explored many attack vectors aimed at achieving the malicious results described above. These attacks are detailed in the following section.

6 System Overview and Security Practices

CAT-SOOP's trust model states that any of the source code content files are completely trusted by the system and read, written, or executed, by someone who has total control of the file system. All foreign submitted code is not to be trusted by CAT-SOOP and is instead executed in its own sandbox environment. In addition to CAT-SOOP's sandbox practices, the system focuses on three other security measures meant to thwart cybersecurity attacks.

6.1 Hashing

CAT-SOOP utilizes pseudorandom number generators (such as UUID4 in python's UUID library) as well as a third-party library: fast_sha256. This hashing subsystem is used to salt user names, hash passwords with the pbkdf2 algorithm for 100000 iterations, and generate API tokens that are paired with the hashed/salted password/username combinations. This encrypted information is stored in a CAT-SOOP specified database for subsequent calls to the API for user authentication. Much of CAT-SOOP's data security relies on this user authentication; for instance, when a user logs into the CAT-SOOP server URL, submits an answer to any of the various question types (multiple choice, expressions, code, numerical, circuit problems, etc...), or simply needs to postload a webpage that requires authorization such as viewing a problem set before its release date, or the user's grades.

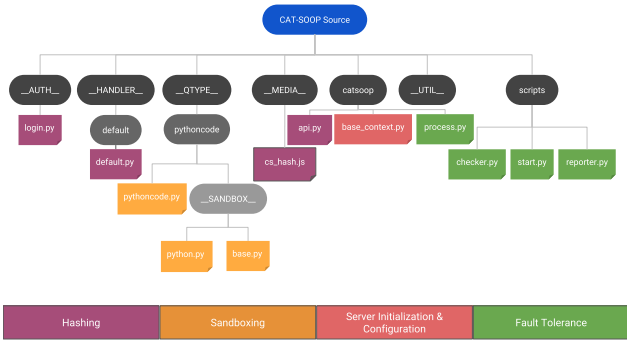

Figure 1: CAT-SOOP source code structure highlighting important sections related to cyber security defense mechanisms.

6.2 Sandboxing

The CAT-SOOP Python Sandbox uses a combination of OS-level isolation and a locked-down Python interpreter. The Python environment uses a custom whitelisting/blacklisting scheme to prevent access to undesirable builtins, modules, functions etc. The OS-based isolation offers extra protections as it forces a malicious student to have to break out of the sandbox in order to view the application's file structure.

Hard limits are placed on the resources (CPU time, memory, and file size) which can be consumed by the process running student code submissions. Fur-

thermore a temporary directory is created and a subprocess (consisting of a simple Python interpreter) is instantiated to run student code. The directory and process are removed/terminated upon completing execution (naturally or as forced by the limits) of student code. Each new submission is given a new directory and process, limiting the damage an attacker can cause even if successful in breaking out of the sandbox or running malware.

6.3 Server Initialization & Configuration

The CAT-SOOP server instance is configured with a select few global variable and functions in mind. For instance, the global variables within the base_context.py file control the base url for the site, the file system's root, and the database where data logs and CAT-SOOP user info is to be stored. Since CAT-SOOP's trust model indicates that these global variable are only accessible to the source code content files, it is of the utmost importance that the sandbox features protect against reading, or writing to the content files and their aforementioned global variables.

6.4 Fault Tolerance

When CAT-SOOP receives a student's code submission, CAT-SOOP spawns a child process in the server that is subject to resource constraint such as: CPU time, memory, and initial file size. Each of these individually attempts to timeout, or throw an error, should a student's code run longer than it should for malicious, or non-malicious intent. For example, a student that hasn't learned control flow may have infinite loops in their code that cause the server to consume all of its resources. This in turn would result in a denial of service for other innocent, or otherwise knowledgeable, students. These child processes are run serially, but also asynchronously in Python's threading module. Though the intent is to protect students from faulty server issues, there exists a current issue that results in child processes not dying and lingering on the server.

7 Attacks on CAT-SOOP

This section details the methodology for some attacks attempted on the CAT-SOOP instance and the features of the system which thwarted these efforts.

7.1 Malicious, resource-consuming, code

To attempt denial-of-service on the CATSOOP instance, our team submitted code containing infinite loops and performing expensive computations simultaneously via multiple accounts. The enforced timeouts on student-submitted code and the queueing of student submissions (such that only a subset of the submitted code instances would run at any given point) made the CATSOOP instance resilient to such attacks.

Furthermore, CATSOOP's sandboxing mechanism, which destroys the temporary locked-down python interpreter running student code, thwarted efforts to setup malicious web servers.

7.2 Directory Traversal Attack

Our team attempted to break out of the sandbox executing student-generated code and gain undesired information by traversing the file structure of the course server. Our first attempt at breaking out of the sandbox was successful by using Python's os library and traversing to the server's root directory. In doing so, the team was able to read configuration files in some of the server's directories. However, our team was unable to modify, write, or execute files in any directory. Our team was also unable to access 'lethal' files (such as those containing user passwords and user submissions) due to them being protected by further privileges (our team's attempts at privilege escalation were unsuccessful).

Additionally, our ability to breakout of the python sandbox was due to a configuration error causing the machine to use the system's Python for student submissions, as opposed to the sandboxed Python. Upon discussing the vulnerability with Adam Hartz, he noted the misconfiguration and, upon his reconfiguring of the machine, our team was unable to replicate

our attack. It should be noted that the possibility exists that our team could have been able to read testing scripts or other files on the instance server before reconfiguration.

7.3 Privilege Escalation and Modifying Files

After the reconfiguration of the CAT-SOOP instance, our team attempted to escalate file permissions on the UNIX server with the shell 'mv' command. As 'mv' preserves the file permissions of the source file, the idea behind this attack was to use the file execute permissions of the student-code submission in the sandboxed python and 'mv' a malicious file in place of the student code to execute on. An example file worth executing, is the shell command binary for 'chmod'. Executing this binary would allow the user within the server to execute a shell command the user might not have the correct permissions for, and grant permission read, write, or execute permissions for files and directories that were once unauthorized. This attack was thwarted as the sandbox disallowed writing new files to disk with the 'os' library; however, empty files are able to be created. The sandbox permissions also thwarted other attempts at privilege escalation such as hijacking python modules or imported files.

8 Next Step - Security User Study

While the system is deemed fairly secure, it is important to point out misconfiguration of a particular CAT-SOOP instance can create vulnerabilities which would not otherwise exist. This was the case with the original CAT-SOOP instance tested by our team. In practice, a large portion of security incidents can be traced back to problems in usability causing a user to make a poor decision. We believe fully assessing, and perhaps improving, CAT-SOOP's security requires a strong security user study. A next step along these lines would be to perform a walkthrough of configuration controls. Exploring how access control and other security mechanisms are configured on the instructor-side and places in this process in which instructors are likely to make mistakes is essential to properly securing CAT-SOOP.

9 Conclusion

The reconfigured CAT-SOOP instance tested by our team proved resilient against a malicious-student-centered threat model. Our team tested several attack vectors including denial of service, privilege escalation, and directory traversal. Though some security flaws were found in the initial instance of CAT-SOOP provided by Adam Hartz, which allowed students to execute their code using the system version of Python, proper reconfiguration of the instance seemed to patch these vulnerabilities. However, this event illustrates the need for a proper security user study focused on instructor-side configuration controls. Such a study would help assess the security of CAT-SOOP beyond theoretical security.

Our team has codified the practices allowing CAT-SOOP to securely execute remotely-generated untrusted code. These include:

1. the use of a hashing subsystem to salt usernames, encrypt passwords, and generate secure API tokens

2. a sandbox configured to provide OS-level isolation and a locked-down interpreter without access to system files

3. the creation of resource-constrained processes for the running of student-submitted and the termination of such processes upon completion of student code

We believe these practices should be followed when implementing tools executing remotely-generated code.

10 Acknowledgments

We thank professors Yael Kalai and Ronald Rivest as well as the full staff of MIT 6.857: Computer and Network Security for their assistance through the

course in our completion of this project. We give special thanks to our lead TA for the project, Jonathan Frankle, who served as a mentor throughout various stages of the project. Lastly, we thank CAT-SOOP developer Adam Harts for creating this software, empowering several courses at MIT. We also thank him for his collaboration throughout the project.

11 Resources and Bibliography

CAT-SOOP Git: `https://catsoop.mit.edu/git/cat-soop/cat-soop`

CAT-SOOP website: `https://catsoop.mit.edu/website`

OWASP Top 10 Security Vulnerabilities List: `https://www.owasp.org/images/7/72/OWASP_Top_10-2017`

OWASP Testing Guide: `https://www.owasp.org/index.php/Testing_Guide_Introduction`

OWASP Application Security Verification Standard (ASVS) Project: `https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project`

Erickson J., Hacking: The Art of Exploitation, Second Edition