Summary

Assignment 1, Part 2

Application Security CS-GY 4753

Prof. Justin Cappos

Author: Casey McGinley

Overview

Inside my project, there is a folder for each user's sandbox code. For all but Prof. Cappos, these folders are listed as the GitHub username. In each of these folders is the pertinent code for that user's original sandbox, as well as my exploit code and my README files to explain my exploit code. Note that all READMEs and PDFs that were included with each user's code originally have been removed to avoid confusion. I've used the following naming convention to ensure clarity as to what code is mine:

- [github_username]_attack.[extension]
- README.[github_username]_attack

This convention holds true in all cases except those were only a specific file may be loaded into the sandbox (e.g. a for a-sandbox.py).

The main project directory also holds this file, a general README.md, as well as another PDF which contains high-level analysis of each of the different sandboxes employed methodologies.

PankajMoolrajani

The author’s sandbox components are:

- **sandbox.py**
- **security.py**
- **operations.py**

I have a single attack spread between two different files:

- **PankajMoolrajani_attack_p1.py**
- **PankajMoolrajani_attack_p2.py**

My corresponding README is:

- **README.PankajMoolrajani_attack_p1p2**

mramdass

The author’s sandbox is:

- **sandbox.py**.

The author’s sample code includes:

- **fib.py**
- **pow.py**
- **combinatorics.py**

I have two distinct attack vectors:

- **mramdass_attack01.py**
- **mramdass_attack02.py**

My corresponding READMEs are:

- **README.mramdass_attack01**
- **README.mramdass_attack02**

kellender

The author’s sandbox components are:

- **turingComplete.c**
- **makefile**

The author’s sample code includes:

- **powerOf2.c**
- **first10Fib.c**

I have a single (potential) attack:

- **kellender_attack.c**

The corresponding README is:

- **README.kellender_attack**

fjm266

My attack strategy ultimately did not work here. The sandbox seems pretty elegant and I couldn’t come up with any exploits to break free or crash the sandbox in a novel way.

ceinfo

The author’s sandbox components are:

- **Turing1.java**
- **TuringTest.class**

The author’s sample code includes:

- **fibonacci.dat**

- **powersOfTwo.dat**
- **testErrors.dat**
- **addSub.dat**

I have a single “attack” (less of an attack and more of an exposé on a bug):

- **ceinfo_attack.dat**

The corresponding README is:

- **README.ceinfo_attack**

crimsonBeard

As with ceinfo’s, it was difficult to find direct attack strategies with crimsonBeard’s sandbox as it used such a limited instruction set. The things I leveraged in my other Python exploits (e.g. digging into the namespace, by passing text-based blacklists, etc.) do not really apply here since no conventional Python commands apply. However, I imagine there is a vuln to be found if for no other reason than the complexity of the code. Given more time, I might have brought it to the surface.

In contrast to the similar approach of ceinfo, crimsonBeard’s does appear to be truly Turing-complete as it implements instruction logic for conditional looping.

As with ceinfo, I did try to crash to the sandbox with some sneaky edge cases (e.g. an input without any instructions, an input with only an EXIT instruction, etc.) but none of these approaches “bore fruit.”

Prof. Cappos

- easytocode.py
- a-sandbox.py
- potentiallyhackablesandbox.py

Both easytocode.py and a-sandbox.py seemed to me to be some of the most impregnable of the bunch. With a-sandbox.py, though the namespace was intact, I couldn’t figure out away to access any functions or keywords given the character restrictions. I played around with the idea of feeding the the sandbox pure bytecode, but exec can only execute bytecode when given to it as a formal code object (what compile() returns). This whitelist approach definitely seems very thorough.

For easytocode.py, the namespace seemed to me to be pretty thoroughly cleared. Upon inspection, False, True and None didn't seem likely to yield and references to other more powerful modules/functions.

When I first started with potentiallyhackablesandbox.py, I had very high hopes of finding a good vulnerability. However, I had more difficulty than expected. Many useful keywords like exec were not blacklisted, but the namespace was thoroughly cleared, aside from the references for range and safecall. I have high confidence that some reference buried within one of those functions could be used in conjunction with exec (and maybe some clever string manipulation) to break free of the sandbox (similar to what I did for a few of my other attacks).