Handin Gearup

...

Overview

- You are not supposed to solve Ivy normally.
- Your task is to find and exploit vulnerabilities in the CS666 handin/autograder infrastructure.

- You have the full source for the infrastructure:
  - https://github.com/brown-csci1660/handin-source

- cs1660: 4 distinct vulnerabilities
- cs1620 / cs2660: 5 distinct vulnerabilities

Vulnerability Counting

- Examples of valid exploits:
  - view other students' grades
  - access other students' submissions
  - run code with TA-group permissions
  - modify data you should not control
  - obtain restricted metadata
- Distinctness is based on:
  - You cannot count two vulnerabilities with exactly the same (file, category) pair.
  - Each vulnerability must have a unique (file, category) tuple.
- Scope restriction:
  - The vulnerability must be in the cs666 course infrastructure.
  - Docker/container escapes do not count.
  - Breaking the container itself is explicitly out of scope

Setup and Logistics

Resources:

- Setup guide: https://hackmd.io/@cs1660/handin-setup-guide
- Starter repository: GitHub Classroom link in the handout
- Wiki: http://brown-csci1660.github.io/handin-wiki/
- Handin source code: https://github.com/brown-csci1660/handin-source

Submitting to *Ivy*

The autograder expects only:

- `KEY`
- `sol.go` (check allowed imports in handout)
- It replaces `main.go` with its own version during grading.
- Handin Command: `cs666_handin ivy`

Autograder

- Autograder must be run by students, but act with TA privileges (to use solutions, record grades) (uses `setgid`)
- No matter who runs cs666_handin, process with permissions of cs666-ta group
- Idea: make autograder do something unintended => do actions as TA!

```
alice@e1100aede57e:~$ ls -la /course/cs666/bin
total 6748
drwxrwsr-x 2 ta cs-666ta 4096 Mar 8 23:13 .
drwxrwsr-x 1 ta cs-666ta 4096 Mar 8 23:13 ..
-rwxrwsr-x 1 ta cs-666ta 2629538 Mar 8 23:13 cs666_handin
…
-rwxrwsr-x 1 ta cs-666ta 2411624 Mar 8 23:13 report
```

cs666_whoami: arbitrary code exec

```
alice@e1100aede57e:~$ cs666_whoami
uid: 1000
euid: 1000
gid: 1000
egid: 1000


alice@e1100aede57e:~$ getent group cs-666ta
cs-666ta:x:1101:ta # GID of TA group


alice@e1100aede57e:~$ <do some exploit>
. . .
uid: 1000
euid: 1000
gid: 1000
egid: 1101 # Success!
```

Hints for Getting Started I

- Poke around the filesystem, try to submit to the autograder
- Try and follow some details of what the scripts are doing
  - Which scripts are invoked when? What is their purpose?
- Based on class and wiki, what parts of the framework might you be able to control?

Hints for Getting Started II

1. How does the hand in system make sure you only turn in code it considers needed for the assignment? What other features/libraries/methods might Go have that are unaccounted for?
2. What ends up getting included in a submission when a student runs the hand in script? What does the archive file extraction code accept?
3. The autograde system creates temporary files at several steps when it runs, where/how are those files created and what actions are allowed on those files?
4. How is data passed between each component of the autograding pipeline? What kind of information about each step might be included in process data?

Handin *Handin*

- A single PDF README/report with your vulnerability reports
  - Metadata
    - vulnerability category
    - source file(s) where the bug lives
  - Procedure
    - how you found it
    - how it works
    - why the exploit succeeds
  - Impact
    - what unauthorized action it enables
    - classify severity
  - Mitigation
    - how to fix it technically
    - where in the source code the fix should go
    - why the fix would stop your exploit
- A demo video named demo.mp4.
- Any payloads, scripts, exploit files, or code needed to reproduce the attacks.

- Severity Labels:
  - Arbitrary Code Execution
  - Data Modification
  - Data Exfiltration
  - Data Theft
  - Metadata Exfiltration