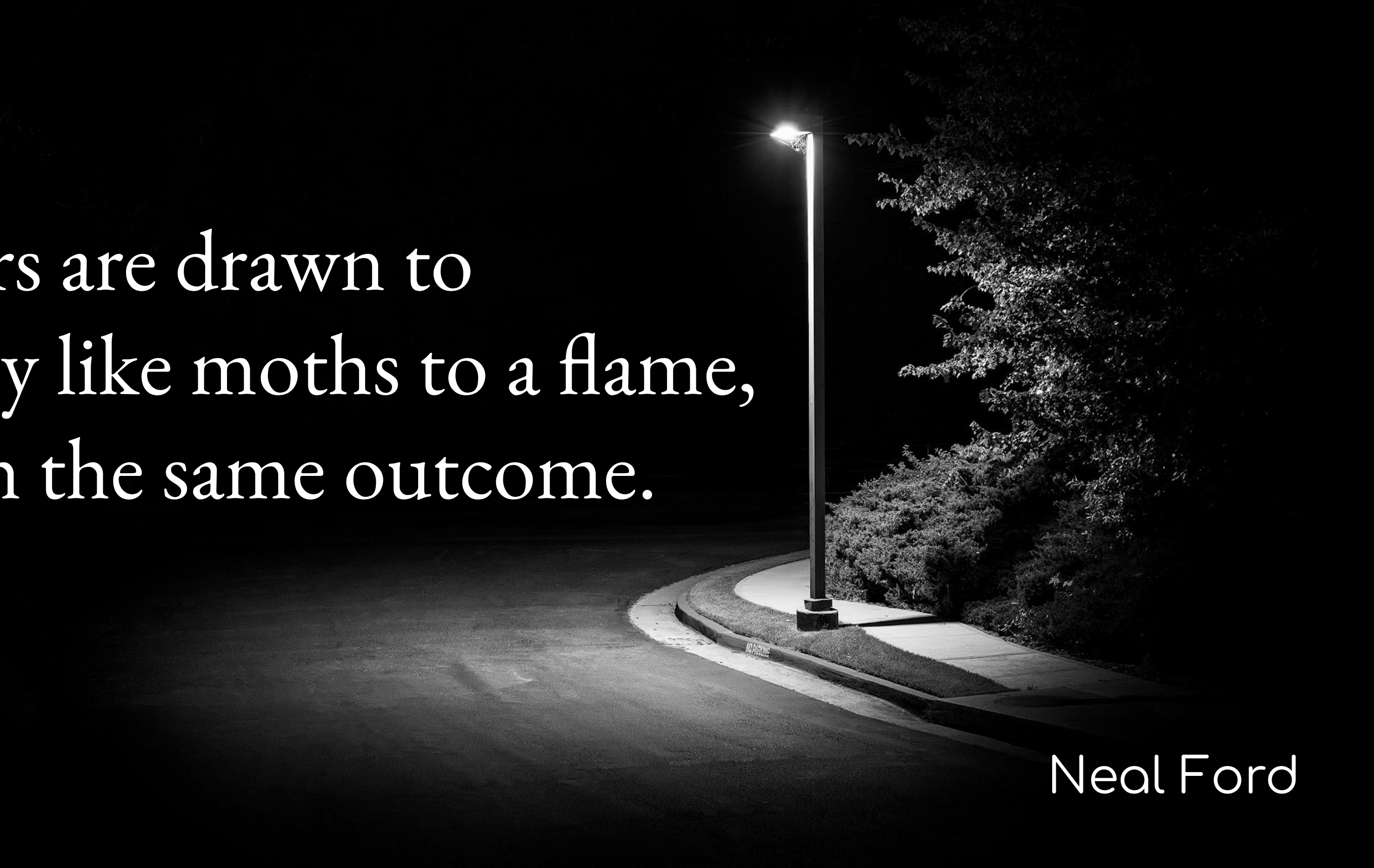
Over Engineering CronJobs

Building an Enterprise Ready™ CTF Infrastructure, For Fun

@mircodezorzi

DISTRIBUTED
SYSTEMS
AHEAD

Developers are drawn to complexity like moths to a flame, often with the same outcome.



Neal Ford

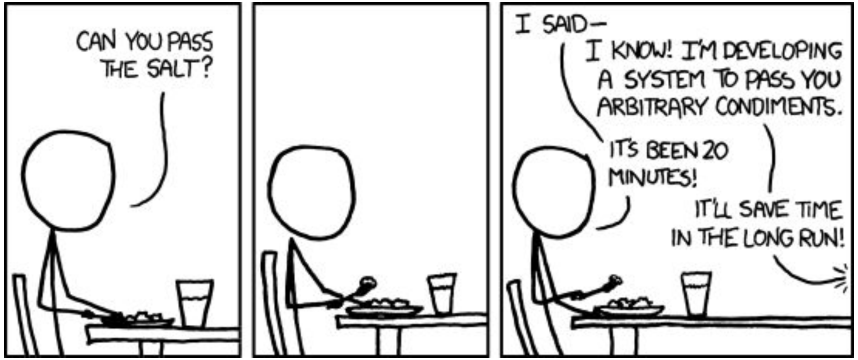
```
while true; do
    for target in ${TARGETS[@]}; do
        python exploit.py "$target" "$PORT"          \
            | grep -Eo "$FLAG_FORMAT"                \
            | jq -s -R 'split("\n")'                 \
            | curl "$ENDPOINT" --data @-             \
                -H 'Content: application/json'       \
                -H "Authentication: $TEAM_TOKEN"
    done
    sleep "$TICK"
done;
```

```
while true; do
    for target in ${TARGETS[@]}; do
        python exploit.py "$target" "$PORT"          \
            | grep -Eo "$FLAG_FORMAT"                \
            | jq -s -R 'split("\n")'                 \
            | curl "$ENDPOINT" --data @-             \
                -H 'Content: application/json'       \
                -H "Authentication: $TEAM_TOKEN"
    done
    sleep "$TICK"
done;
```

```
while true; do
    for target in ${TARGETS[@]}; do
        python exploit.py "$target" "$PORT"          \
            | grep -Eo "$FLAG_FORMAT"                \
            | jq -s -R 'split("\n")'                 \
            | curl "$ENDPOINT" --data @-             \
                -H 'Content: application/json'       \
                -H "Authentication: $TEAM_TOKEN"
    done
    sleep "$TICK"
done;
```

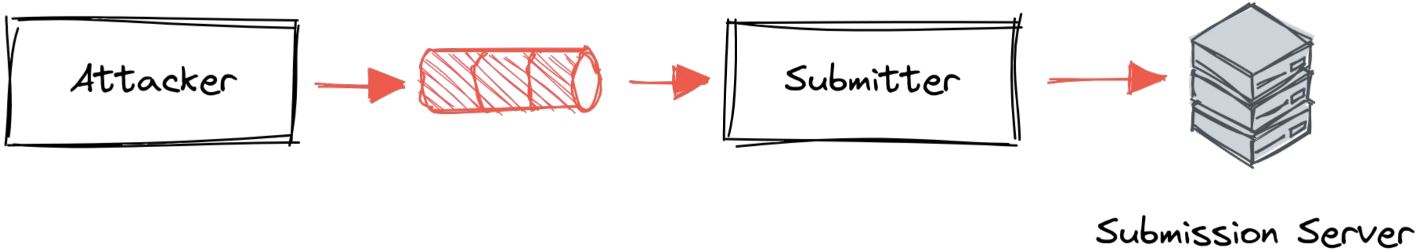
```
while true; do
    for target in ${TARGETS[@]}; do
        python exploit.py "$target" "$PORT"          \
            | grep -Eo "$FLAG_FORMAT"                 \
            | jq -s -R 'split("\n")'                 \
            | curl "$ENDPOINT" --data @-             \
                -H 'Content: application/json'       \
                -H "Authentication: $TEAM_TOKEN"
    done
    sleep "$TICK"
done;
```

```
while true; do
    for target in ${TARGETS[@]}; do
        python exploit.py "$target" "$PORT"          \
            | grep -Eo "$FLAG_FORMAT"                \
            | jq -s -R 'split("\n")'                 \
            | curl "$ENDPOINT" --data @-             \
                -H 'Content: application/json'       \
                -H "Authentication: $TEAM_TOKEN"
    done
    sleep "$TICK"
done;
```


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

XKCD

1. No isolation between exploits;
2. No versioning of exploits;
3. No retries or timeouts for exploit runs;
4. No observability (monitoring & logging);
5. No submission batching;
6. And most importantly, it’s boring!



Attacker

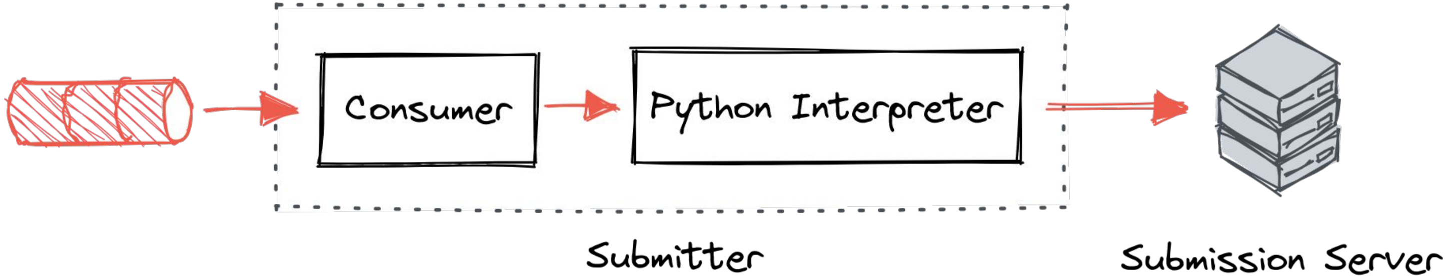
Submitter

Submission Server

The Submitter

1. Operable
2. Resilient
3. Scriptable

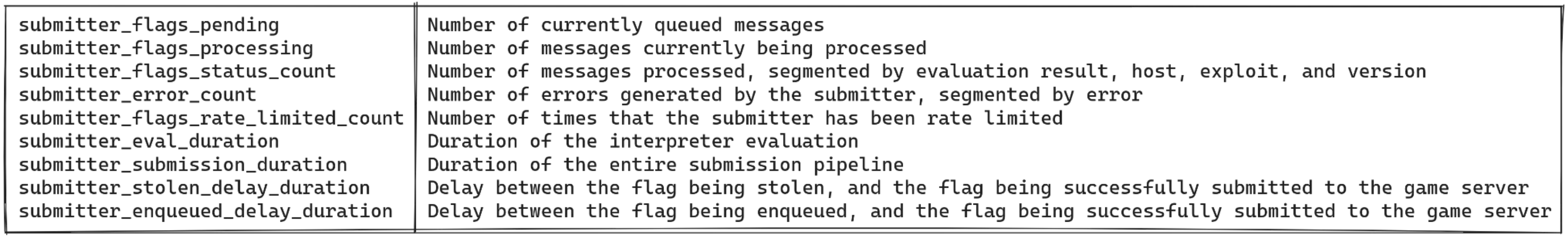
```python
import requests

from submitter import SubmissionResult as SR

def submit(flags):
    response = requests.post(
        url=SUBMISSION_ENDPOINT,
        headers={
            'Authorization': TEAM_TOKEN,
        },
        json=flags,
    )

    if response.status_code == 429:
        return SR.RateLimited

    return SR.Accepted if response.status_code == 200 else SR.Unknown
```


Consumer
Python Interpreter
Submitter
Submission Server

```json
{
  "flag": "FLAG{deadbeefdeadbeef}",
  "host": "10.10.5.10",
  "exploit": "web-sqli",
  "version": "edc8e75",
  "stolen_at": 1679584353,
  "enqueued_at": 1679584554
}
```

Considerations:

- Enrich flag information about tick of origin?
- Deduplication of flags using a persistent store

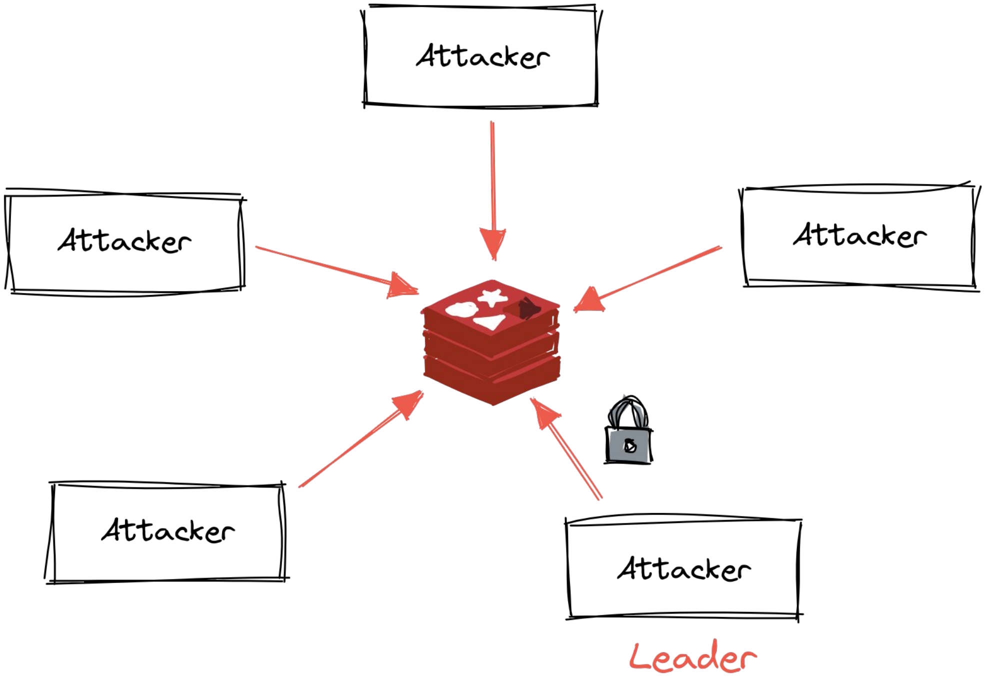


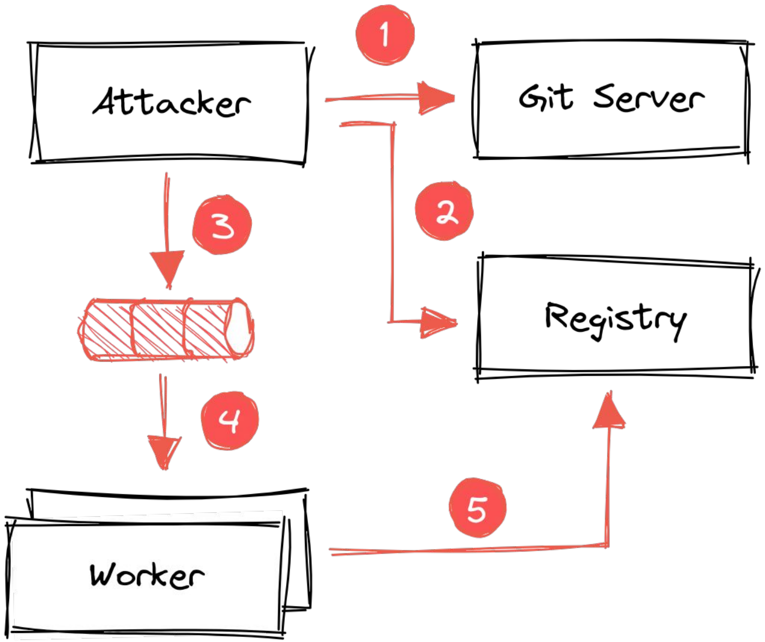
submitter_flags_pending	Number of currently queued messages
submitter_flags_processing	Number of messages currently being processed
submitter_flags_status_count	Number of messages processed, segmented by evaluation result, host, exploit, and version
submitter_error_count	Number of errors generated by the submitter, segmented by error
submitter_flags_rate_limited_count	Number of times that the submitter has been rate limited
submitter_eval_duration	Duration of the interpreter evaluation
submitter_submission_duration	Duration of the entire submission pipeline
submitter_stolen_delay_duration	Delay between the flag being stolen, and the flag being successfully submitted to the game server
submitter_enqueued_delay_duration	Delay between the flag being enqueued, and the flag being successfully submitted to the game server

The Attacker

1. Isolation
2. Versioning
3. Distributed

```
/
    Dockerfile
    main
    requests.txt
```


Attacker
Attacker
Attacker
Attacker
Attacker
Leader



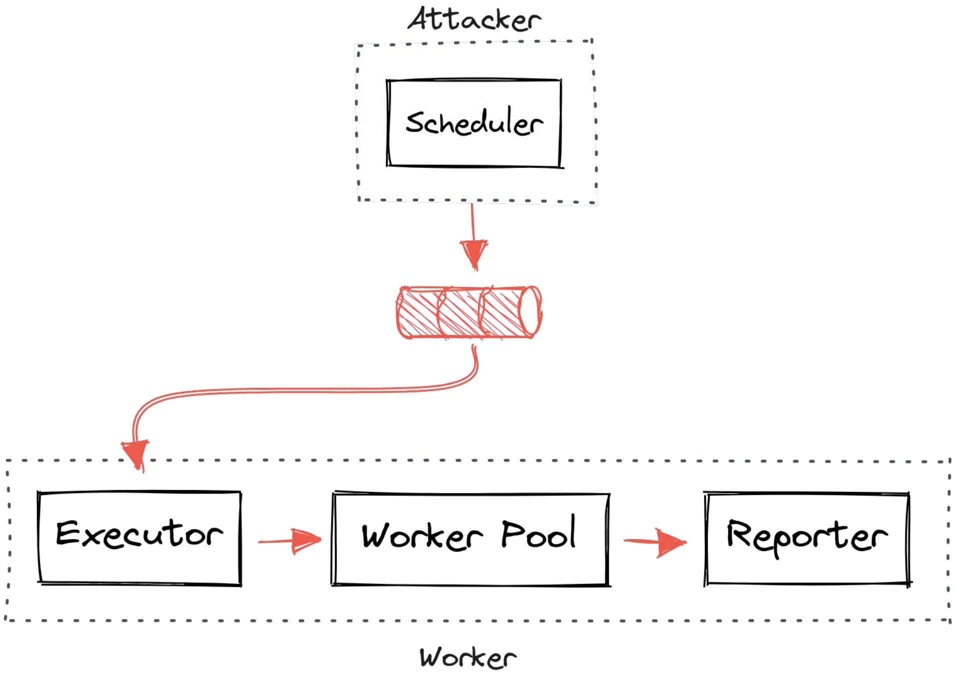
1 pull repository

2 build image and push it to OCI

3 push job to queue

4 consume job from queue

5 pull image from OCI


Attacker
Scheduler
Executor
Worker Pool
Reporter
Worker

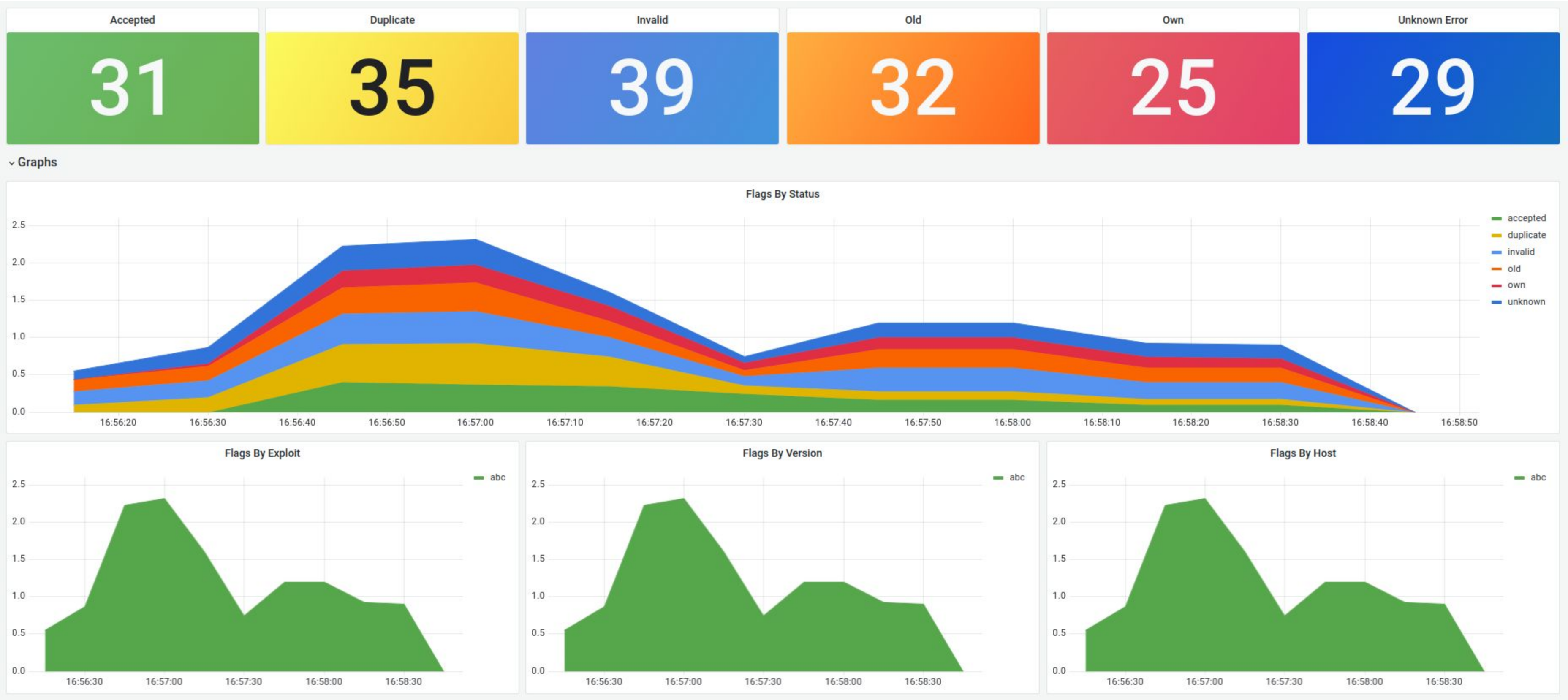
```
{
  "host": "10.0.1.1",
  "port': 8080,
  "image": "deadbeef:deadbeef",
  "enqueued_at": 1679584554
}
```

The CLI

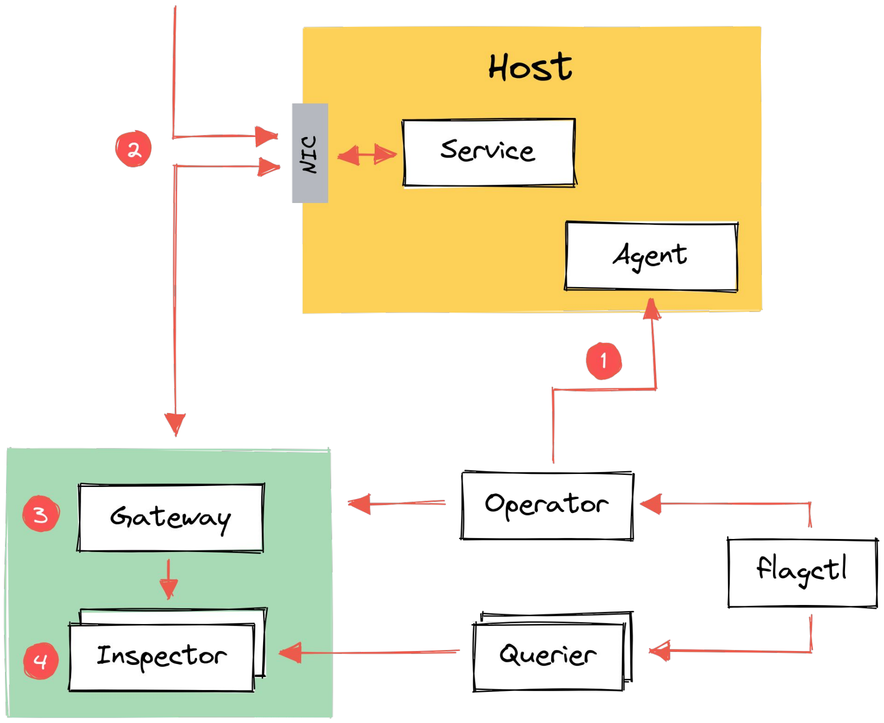
```
# Create a reference service
$ flagctl create service http-server --port=8080

# Create a reference bucket
$ flagctl create bucket default --hostsfile=/etc/targets

# Create an exploit using the pwntools template
$ flagctl create exploit pwn-rce \
    --service=http-server --bucket=default --template=python-pwntools

# Edit your exploit
$ vim pwn-rce/main

# Push the changes to remote
$ flagctl push pwn-rce

# Start the exploit
$ flagctl start pwn-rce
```

```
# Run exploits locally, use implicit bucket 'default'
$ flagctl run exploit.py --service=http-server

# Run command with remote service and bucket
$ flagctl run --command='python3 exploit.py {{ .Host }} {{ .Port }}' \
    --service=http-server --bucket=default /path/to/file

# Specify custom port and hosts
$ flagctl run --command='python3 exploit.py {{ .Host }} {{ .Port }}' \
    --port=8080 --hostsfile=/etc/targets /path/to/file

# Run dockerized exploit
$ flagctl run --docker --port=8080 --hostsfile=/etc/targets /path/to/dockerfile
$ flagctl help

State management:
  create      Create a resource
  get         Display one or more resources
  describe    Display detailed information about a resource
  edit        Edit a resources
  delete      Delete a resource

Exploit management:
  push        Push exploit
  clone       Clone an exploit

Exploit status management:
  start       Start an exploit
  stop        Stop an exploit
  checkout    checkout exploit to a particular commit
  logs        Fetch the logs for an exploit

Miscellaneous:
  submit      Manually submit flags through stdin
  run         Run exploit locally

Use "flagctl <command> --help" for more information about a given command.
```

attacker_job_executed_count	Number of executed jobs
attacker_job_duration	Duration of the jobs
attacker_job_error_count	Number of errors generated by the job
attacker_exploit_build_duration	Duration of the exploit building pipeline
attacker_enqueue_delay_duration	Delay between jobs being enqueued and being consumed



Honorable Mention: The Proxy



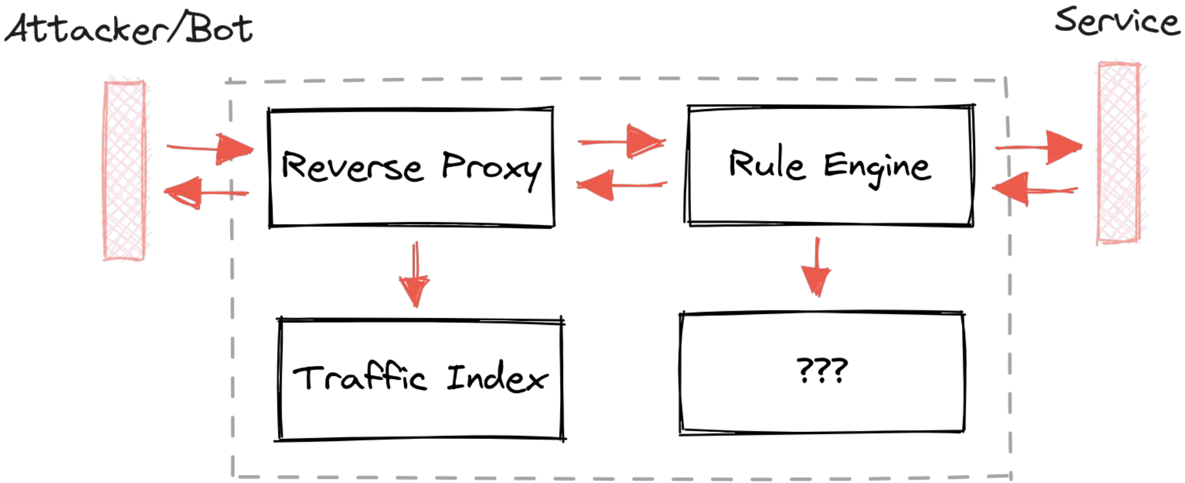
1 eBPF routing rules

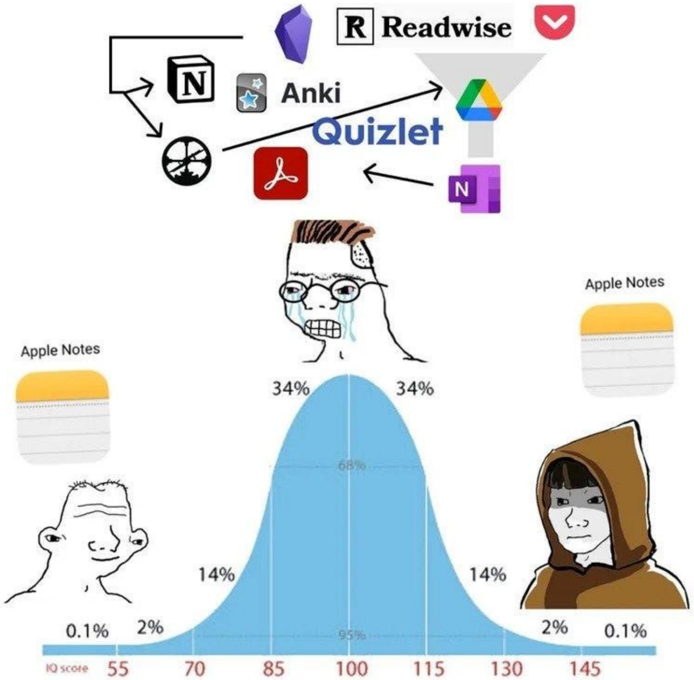
2 route traffic to analyzer

3 load balancer

4 rule engine

5 stateless query engine


Attacker/Bot
Service
Reverse Proxy
Rule Engine
Traffic Index
???


Readwise
Anki
Quizlet
Apple Notes
Apple Notes
34%
34%
68%
14%
14%
0.1%
2%
2%
0.1%
95%
IQ score
55
70
85
100
115
130
145

Questions?