ECHO · WRITE-UPS · ACE-T CORE

THE ACE-T WRITE-UPS

BY **DIGITAL ORUKAMI** FILED **JUN 2026** SCOPE **8 CHALLENGES**

ACE-T Core is not a multiple-choice badge. Section one is the written exam, section two is five challenges of your choosing off the Antisyphon Cyber Range, and section three — the part the certification actually turns on — is **report writing**: two write-ups on your approach, your methodologies and your findings, delivered for the public to read. These are those two reports, published in full.

PROVENANCE

Submitted **20 June 2026** for the **ACE-T Core** certification, Antisyphon Training, as the ECHO Club *Digital Forensics Examination Report*, Volumes I and II. The challenges were solved on the **MetaCTF Training Platform** (the Antisyphon Cyber Range). Supervising writer: **Digital Orukami**.

This page is the club's public setting of those two reports. The exam PDFs open on a cover block carrying the candidate's legal name and student ID — not the identity this site publishes under, and the only thing left out here. The findings, evidence tables, methodology, code, flags and recommendations are the submitted text.

Disclaimer. This report documents challenge solutions completed on the MetaCTF Training Platform for educational purposes. All analysis was performed in a controlled CTF environment. ECHOClub is not responsible for the organization or administration of the MetaCTF platform.

CONTENTS

VOLUME I

FORENSICS & MALWARE ANALYSIS

MALWARE PROCMON · FTP CAPTURE · SYSMON LOG · PROCESS DUMP

CHALLENGE 1 · DIGITAL FORENSICS / MALWARE ANALYSIS ·

THREAT LEVEL: CRITICAL

A NEEDLE IN A NEEDLESTACK

A Windows system was generating repeated crash popups for `malware.exe`. A Process Monitor (Procmon) capture log — **Logfile.PML, 54 MB, 108,560 events** — was provided. The objective was to identify the specific file responsible for launching `malware.exe`.

EVIDENCE

ARTIFACT	DESCRIPTION
Logfile.PML	54 MB Process Monitor binary log — 108,560 captured events
malware.exe	Crashing process located at `C:\malware.exe`
netutils.dll	Malicious DLL found at `C:\Windows\netutils.dll`
phHelper.exe	Process Hacker helper utility used as injector
WerFault.exe	Windows Error Reporting — triggered by the malware.exe crashes

METHODOLOGY

TOOL SETUP

The PML file was parsed using the Python `procmon-parser` library (v0.3.13), since version incompatibilities prevented opening it in the Procmon GUI.

```
pip install procmon-parser --break-system-packages
```

PROCESS TREE RECONSTRUCTION

All Process Create events were extracted to reconstruct the full execution chain. The key discovery was `phHelper.exe` being launched by `cmd.exe` rather than `ProcessHacker.exe`.

PROCESS	PARENT	COMMAND LINE
ProcessHacker.exe	cmd.exe	ProcessHacker.exe
phHelper.exe	cmd.exe (anomalous)	phHelper.exe 1016 phLib.dll
notepad.exe	ProcessHacker.exe	notepad.exe
explorer.exe (2528)	cmd.exe	explorer.exe
malware.exe (2104)	explorer.exe (2528)	"C:\malware.exe"

DLL LOAD ORDER ANALYSIS

Examining the events between the `shacct.dll` COM object load and the `malware.exe` Process_Create at event [70855], a critical DLL load-order anomaly was identified.

```
Event 70824: CreateFile C:\Windows\netutils.dll           ← LOADED FIRST (malicious)
Event 70831: CreateFile C:\Windows\System32\netutils.dll  ← skipped
```

Windows DLL search order causes `C:\Windows\` to be searched before `C:\Windows\System32\` when it appears in the PATH environment variable. A malicious `netutils.dll` planted at `C:\Windows\netutils.dll` was loaded instead of the legitimate System32 version.

FULL ATTACK CHAIN

- Attacker planted a malicious `netutils.dll` at `C:\Windows\netutils.dll`
- `cmd.exe` launched `phHelper.exe` with arguments targeting `ProcessHacker.exe` (PID 1016)
- `phHelper.exe` opened `ProcessHacker.exe` with PROCESS_CREATE_THREAD | PROCESS_VM_OPERATION | PROCESS_VM_WRITE (`0x0000002a`) — classic CreateRemoteThread injection
- `explorer.exe` processed the Start Menu ProgramsCache, triggering COM CLSID `{4F6BCD94-C2A5-42CE-8DBC-31E794BE4630}`
- `shacct.dll` (the legitimate Shell Computer Accounts DLL) loaded and attempted to import `netutils.dll`
- The malicious `netutils.dll` in `C:\Windows\` was loaded before the System32 version
- The malicious DLL spawned `C:\malware.exe` (PID 2104)
- `malware.exe` connected to **192.168.220.129:4444** — a Meterpreter reverse TCP shell

FLAG

`C:\Windows\netutils.dll`

THREAT ASSESSMENT

This attack demonstrates a sophisticated multi-stage compromise combining **DLL search-order hijacking** (MITRE ATT&CK **T1574.001**) with COM object abuse for persistence. The attacker leveraged Windows Explorer's own process to spawn the malware, making it appear as legitimate system activity.

- Reverse shell active on port 4444 — the attacker had full remote access
- Malware relaunched on every new Explorer session — persistent across logoffs
- Attribution to `ProcessHacker.exe` masked the malicious DLL activity in most security logs

RECOMMENDATIONS

- Delete `C:\Windows\netutils.dll` and verify the hash of `C:\Windows\System32\netutils.dll`
- Terminate and quarantine `malware.exe` and all associated processes
- Rotate all credentials for accounts active on this system — assume full compromise
- Enable SafeDllSearchMode: `HKLM\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode = 1`
- Audit the `C:\Windows\` directory for unexpected DLLs — it should not contain `netutils.dll`
- Block outbound TCP port 4444 at the perimeter firewall
- Deploy EDR rules to alert on `phHelper.exe` launched by any process other than `ProcessHacker.exe`
- Investigate the initial access vector — how did the attacker gain write access to `C:\Windows\`?

CHALLENGE 2 · NETWORK FORENSICS / PACKET ANALYSIS · THREAT LEVEL: HIGH

FLAGS OVER THE WIRE

A packet capture (`capture.pcap`, 233 KB) of an FTP session between two computers was provided. One machine downloaded a file from the other. The objective was to reconstruct the transferred file from the raw packet data and extract the flag contained within it.

EVIDENCE

ARTIFACT	DESCRIPTION
capture.pcap	83-packet network capture of an FTP session
flags.zip	230,329 bytes — ZIP file reconstructed from the FTP data stream
flags.png	1944×2844 pixel PNG extracted from the ZIP — contained the flag
FTP credentials	Username `hack0r` / password `Chiapet2` — cleartext

METHODOLOGY

PROTOCOL IDENTIFICATION

Packet analysis using Scapy revealed an FTP control channel on TCP port 21 and a data channel on port 55983 (FTP passive mode). The full FTP session command sequence was recovered.

CLIENT COMMAND	SERVER RESPONSE
USER hack0r	331 Please specify the password.
PASS Chiapet2	230 Login successful.
PASV	227 Entering Passive Mode (10,0,0,215,218,175)
RETR flags.zip	150 Opening BINARY mode data connection for flags.zip (230329 bytes). / 226 Transfer complete.
QUIT	221 Goodbye.

DATA CHANNEL PORT CALCULATION

In FTP passive mode the data port is encoded in the PASV response as `p1 * 256 + p2`:

```
227 Entering Passive Mode (10,0,0,215,218,175)
Data port = 218 * 256 + 175 = 55983
```

FILE RECONSTRUCTION

All TCP packets originating from port 55983 (server to client) were collected in sequence and their payloads concatenated to reconstruct the raw file.

```
data = b''
for p in pkts:
    if p.haslayer(TCP) and p.haslayer(Raw):
        if p[TCP].sport == 55983:
            data += p[Raw].load
```

The reconstructed 230,329 bytes carried magic bytes `504b0304` — the ZIP file signature. The ZIP contained `flags.png`, which displayed a checkered flag image and the flag text.

FLAG

`MetaCTF{wave_the_checkered_flag_on_this_one_youve_found_the_flag}`

THREAT ASSESSMENT

FTP transmits all data — including credentials and file contents — in cleartext over TCP. Anyone with network access between client and server can intercept usernames, passwords and every transferred file using tools as simple as Wireshark or tcpdump.

- Credentials exposed in plaintext: `hack0r` / `Chiapet2`
- All transferred file content fully visible in the capture
- No integrity protection — files could be modified in transit without detection

RECOMMENDATIONS

- Replace FTP immediately with SFTP (SSH File Transfer Protocol) or FTPS (FTP over TLS)
- Rotate the compromised credentials `hack0r` / `Chiapet2` — assume these are fully compromised
- Implement network monitoring to detect and block plaintext FTP sessions on port 21
- Enforce file transfer encryption policies across the organization
- Use key-based authentication for SFTP rather than password authentication

---

CHALLENGE 3 · THREAT HUNTING / WINDOWS FORENSICS · THREAT LEVEL: CRITICAL

CATCH ME IF YOU CAN, MON!

A Sysmon event log (`sysmon_logfile.evtx` , 2.1 MB) was provided from a Windows system exhibiting suspicious activity inside Process Hacker. Threat intelligence indicated the attacker was hiding inside legitimate security tools. The objective was to identify the specific DLL used for memory injection.

EVIDENCE

ARTIFACT	DESCRIPTION
sysmon_logfile.evtx	2.1 MB Windows Sysmon event log — 724 total events
Event ID 1 (×8)	Process Create — revealed the anomalous `phHelper.exe` launch
Event ID 10 (×590)	ProcessAccess — showed the DLL injection signature
Event ID 7 (×120)	Image Loaded — DLL load activity
phHelper.exe	Process Hacker helper utility — launched anomalously from `cmd.exe`
phLib.dll	Malicious DLL injected into `ProcessHacker.exe` via CreateRemoteThread

METHODOLOGY

LOG PARSING

The EVTX binary was parsed using the `python-evtx` library to extract and correlate events programmatically.

```
pip install python-evtx --break-system-packages
```

PARENT PROCESS ANOMALY DETECTION

Examination of all Process Create (Event ID 1) events revealed the critical anomaly: `phHelper.exe` was launched from `cmd.exe` instead of its expected parent `ProcessHacker.exe`.

PROCESS	PARENT	COMMAND LINE
ProcessHacker.exe	cmd.exe	ProcessHacker.exe
phHelper.exe	cmd.exe (anomalous)	phHelper.exe 1016 phLib.dll
phHelper.exe	cmd.exe (anomalous)	phHelper.exe 2156 phLib.dll
notepad.exe	ProcessHacker.exe	notepad.exe
whoami.exe	cmd.exe	whoami

ACCESS MASK DECODING

The ProcessAccess events from `phHelper.exe` to `ProcessHacker.exe` showed GrantedAccess mask `0x0000002a`.

FLAG VALUE	CONSTANT	PURPOSE
`0x0002`	PROCESS_CREATE_THREAD	Create a thread in the target process
`0x0008`	PROCESS_VM_OPERATION	Perform VirtualAllocEx in the target
`0x0020`	PROCESS_VM_WRITE	Write memory via WriteProcessMemory

This exact combination (`0x0000002a`) is the classic CreateRemoteThread DLL injection signature. `phHelper.exe` was injecting `phLib.dll` into the `ProcessHacker.exe` process space.

FULL PATH IDENTIFICATION

Extracting full paths from all ProcessHacker-related call traces confirmed the malicious DLL location. That DLL was accessing `notepad.exe` with GrantedAccess `0x001fffff` (PROCESS_ALL_ACCESS) — the maximum possible permission level. Once injected into the `ProcessHacker.exe` process space, all of its activity was attributed to the legitimate security tool.

FLAG

`C:\Users\Administrator\Downloads\processhacker-2.39-bin\x64\phLib.dll`

THREAT ASSESSMENT

This attack demonstrates a sophisticated **Living-off-the-Land** technique that abuses a legitimate security tool's own helper utility as a DLL injector. By hiding inside Process Hacker, the malicious activity inherits the tool's trusted reputation and bypasses security products that whitelist it.

- Process injection into a security tool — effectively blinds defenders
- PROCESS_ALL_ACCESS to `notepad.exe` — memory contents fully readable and writable
- The attacker used `cmd.exe` to manually invoke `phHelper.exe` — indicating an interactive post-exploitation session
- `whoami.exe` execution confirms active attacker presence performing reconnaissance

RECOMMENDATIONS

- Terminate Process Hacker and quarantine the malicious `phLib.dll`
- Assume `notepad.exe` memory contents were read — review what sensitive data was open
- Investigate how the attacker gained access to the system — check for initial access vectors
- Implement file integrity monitoring on Process Hacker's installation directory
- Configure Sysmon to alert on `phHelper.exe` launched by any process other than `ProcessHacker.exe`
- Alert on ProcessAccess events with mask `0x0000002a` from unexpected source processes
- Restrict write access to security tool installation directories to SYSTEM only
- Enable Windows Credential Guard to protect against memory-based credential theft
- Adjust and reconfigure AppLocker policies to restrict execution of `notepad.exe` and other commonly abused legitimate Windows binaries to authorized users, or to none at all

---

CHALLENGE 4 · MEMORY FORENSICS / MALWARE ANALYSIS ·

THREAT LEVEL: HIGH

CREATEREMOTEFLAG()

A Windows Minidump file (`notepad.exe_200310_020705.dmp` , 143 MB) was provided, captured from a `notepad.exe` process suspected of having malicious code injected into it by `KoVCxCjx.exe` via CreateRemoteThread. The objective was to identify the DLL responsible for the injection by analyzing the process memory dump.

EVIDENCE

ARTIFACT	DESCRIPTION
notepad.exe_200310_020705.dmp	143 MB Windows Minidump — captured 2020-03-10 02:07:05
18 streams	Minidump stream count — a full memory snapshot
KoVCxCjx.exe	Attacker-controlled process — name chosen to evade casual inspection
Injected DLL	`C:\temp\finding_a_dynamicly_linked_flag_for_some_cold_hard_points.dll`
PDB path	`C:\Users\Administrator\Documents\Payload\x64\Debug\ ... pdb`

METHODOLOGY

DUMP PARSING

The Minidump was parsed using the Python `minidump` library to enumerate all loaded modules, memory segments and thread information.

```
pip install minidump --break-system-packages

from minidump.minidumpfile import MinidumpFile
mf = MinidumpFile.parse('notepad.exe_200310_020705.dmp')
```

MODULE ENUMERATION

Enumerating all modules loaded into the `notepad.exe` process space revealed 80+ legitimate Windows DLLs and one highly suspicious entry.

MODULE	BASE ADDRESS	ASSESSMENT
`C:\Windows\System32\notepad.exe`	0x7ff6e8b80000	Legitimate
`C:\Windows\System32\ntdll.dll`	0x7fff71440000	Legitimate
`C:\Windows\System32\netutils.dll`	0x7fff6cf90000	Legitimate
`C:\temp\finding_a_dynamicly_linked_flag_for_some_cold_hard_points.dll`	0x7fff61ca0000	Malicious
`C:\Windows\System32\VCRUNTIME140D.dll`	0x7fff618f0000	Debug runtime

The DLL in `C:\temp\` stands out immediately — it sits in a writable directory rather than a system directory, it is loaded into a `notepad.exe` process, and it has a name that describes its own purpose. Its associated PDB debug path confirmed it was compiled by the attacker: `C:\Users\Administrator\Documents\Payload\x64\Debug\finding_a_dynamicly_linked_flag_for_some_cold_hard_points.pdb`

MEMORY REGION ANALYSIS

The injected DLL occupied memory segments from `0x7fff61ca0000` to `0x7fff61cc4000` (`0x24000` bytes). String extraction from these segments revealed:

- `MessageBoxA` import — the DLL displays a popup message
- `RegOpenKeyExW` / `RegQueryValueExW` — it reads from the Windows registry
- `Hey you!` — message box caption text, obviously suspicious
- `notepad_bonus_flags_are_the_best_of_flags` — a hidden window class name

The full path of the injected DLL, as loaded in the process module list, provides the flag.

> **FLAG**
>
> `C:\temp\finding_a_dynamicly_linked_flag_for_some_cold_hard_points.dll`

THREAT ASSESSMENT

CreateRemoteThread DLL injection is a well-established process injection technique that allows an attacker to execute arbitrary code within the context of another process. By targeting `notepad.exe` — a commonly whitelisted application — the attacker's malicious DLL inherits the process's trust level and evades many security controls.

- Full code execution within the `notepad.exe` process context
- Registry access from an injected DLL — potential credential or configuration theft
- A debug build (`VCRUNTIME140D.dll`) suggests this was a development/testing payload
- `C:\temp\` is a world-writable directory — low privilege required to plant the DLL

RECOMMENDATIONS

- Remove `C:\temp\finding_a_dynamicly_linked_flag_for_some_cold_hard_points.dll` and terminate any affected processes
- Audit `C:\temp\` and all other world-writable directories for unauthorized DLLs
- Investigate `KoVCxCjx.exe` — identify its origin, persistence mechanism and full capabilities
- Review registry keys accessed by the injected DLL for signs of data exfiltration or persistence
- Implement application whitelisting to prevent unauthorized DLLs from loading
- Configure EDR to alert on DLL loads from `C:\temp\` or other non-system directories
- Restrict write access to `C:\temp\` — only authorized processes should write there
- Deploy memory scanning to detect injected code regions in running processes

VOLUME II

NETWORK RECONNAISSANCE & FORENSICS

SNMP ENUMERATION · IP GEOLOCATION · PASSWORD CRACKING · HEXED PNG

CHALLENGE 1 · NETWORK RECONNAISSANCE / PROTOCOL ANALYSIS · THREAT LEVEL: HIGH

TAKING A WALK

A server at `host1.metaproblems.com` was identified as running SNMP on its default port. The challenge demonstrated how SNMPv1 and SNMPv2c lack meaningful security controls, allowing any party with the community string to enumerate the full Management Information Base (MIB) of a device. The flag was hidden in one of the standard SNMP OID fields.

EVIDENCE

ARTIFACT	DESCRIPTION
host1.metaproblems.com	Target host — SNMP service on UDP port 161
Community string	`public` — default read-only, unchanged from factory
sysContact.0 OID	`1.3.6.1.2.1.1.4.0` — contained the flag instead of an admin contact
sysName.0	`90f7a1a0c6d3` — hostname of the target device
sysLocation.0	`metactf` — device location field

METHODOLOGY

SNMP VERSION SECURITY CONTEXT

VERSION	AUTHENTICATION	ENCRYPTION	RISK
SNMPv1	Community string	None	High — cleartext
SNMPv2c	Community string	None	High — cleartext
SNMPv3	Username + password	AES/DES	Low — encrypted

SNMP WALK EXECUTION

Cloud infrastructure firewalls blocked outbound UDP from the analysis environment, so the SNMP walk was performed from a local Windows terminal using the `pysnmp` Python library.

```
pip install pysnmp

import asyncio
from pysnmp.hlapi.v3arch.asyncio import *

async def run():
    snmpEngine = SnmpEngine()
    async for errorIndication, errorStatus, errorIndex, varBinds in walk_cmd(
        snmpEngine,
        CommunityData('public'),
        await UdpTransportTarget.create(('host1.metaproblems.com', 161)),
        ContextData(),
        ObjectType(ObjectIdentity('1.3.6.1')),
        lexicographicMode=False
    ):
        for varBind in varBinds:
            print(varBind)

asyncio.run(run())
```

Note: pysnmp v7 uses snake_case function names (`walk_cmd`) rather than the camelCase names used in older versions. This is a common compatibility trap.

RESULTS

OID / FIELD	VALUE
SNMPv2-MIB::sysObjectID.0	SNMPv2-SMI::enterprises.8072.3.2.10
SNMPv2-MIB::sysUpTime.0	1047691385
SNMPv2-MIB::sysContact.0	**MetaCTF{walking_your_way_to_the_flag}**
SNMPv2-MIB::sysName.0	90f7a1a0c6d3
SNMPv2-MIB::sysLocation.0	metactf

FLAG

`MetaCTF{walking_your_way_to_the_flag}`

THREAT ASSESSMENT

SNMP with default community strings is one of the most commonly exploited network misconfigurations in enterprise environments. The `sysContact` field — normally used for administrator contact information — was repurposed to hide the flag, demonstrating how arbitrary data can be stored in SNMP fields and exfiltrated by any party that can enumerate the MIB.

- Default community string `public` left unchanged — any device on the network can enumerate the full MIB
- All SNMP data transmitted in cleartext over UDP — interceptable by network-level attackers
- An SNMP MIB can reveal OS version, running services, network interfaces, routing tables and user accounts
- A read-write community string (`private`) could allow remote configuration changes if similarly misconfigured

RECOMMENDATIONS

- Upgrade to SNMPv3 with authentication (SHA) and encryption (AES-256) immediately
- Change all community strings from their default values (`public` , `private`) to strong random strings
- Apply ACLs to restrict SNMP access to authorized management hosts only
- Disable SNMP entirely on devices that do not require remote monitoring
- Block UDP port 161 at the network perimeter — SNMP should never be exposed to the internet
- Audit all SNMP-enabled devices on the network for default configurations
- Implement SNMP monitoring to detect excessive GET/WALK requests indicative of reconnaissance

---

CHALLENGE 2 · OSINT / THREAT INTELLIGENCE · THREAT LEVEL: LOW

THREAT INTEL TEAM

A threat intelligence scenario presented a suspicious IP address (`45.126.128.11`) and required identification of its geographic origin. This type of IP geolocation analysis is a fundamental OSINT technique used in incident response and threat intelligence workflows to attribute malicious activity to geographic regions and identify hosting infrastructure.

EVIDENCE

ARTIFACT	VALUE
Target IP	`45.126.128.11`
Country code	NZ
Country	New Zealand
City	Auckland
Region	Auckland
ASN / org	Retrieved via the ipinfo.io API

METHODOLOGY

IP LOOKUP

The IP address was queried against the ipinfo.io geolocation API using PowerShell.

```
Invoke-RestMethod -Uri 'http://ipinfo.io/45.126.128.11/json' | Select-Object country, city, region, org
```

The API returned country code `NZ`, confirming the address is geographically registered to New Zealand.

FLAG

`MetaCTF{New Zealand}`

THREAT ASSESSMENT

IP geolocation is an intelligence gathering technique rather than an attack in itself. However, understanding the geographic origin of suspicious IP addresses is critical for:

- Attributing attacks to geographic regions or nation-state actors
- Identifying hosting providers and cloud infrastructure used by threat actors
- Supporting legal and law enforcement actions with geographic context
- Configuring geo-blocking rules on firewalls and WAFs

Note: IP geolocation is not perfectly accurate — VPNs, Tor exit nodes and compromised systems can all cause an address to appear to originate from a different country than the actual attacker.

RECOMMENDATIONS

- Cross-reference suspicious IPs against threat intelligence feeds (VirusTotal, Shodan, AbuseIPDB)

- Implement geo-blocking for regions with no legitimate business relationship, where appropriate
- Log and alert on connections from flagged IP ranges at the perimeter firewall
- Never rely solely on IP geolocation for attribution — correlate with other indicators
- Use multiple geolocation sources (ipinfo.io, MaxMind, ip-api.com) to cross-validate results

---

CHALLENGE 3 · CRYPTOGRAPHY / PASSWORD SECURITY ·

THREAT LEVEL: HIGH

A CONFIDENT HASH

A bcrypt password hash was provided alongside a custom wordlist of 75 entries. The objective was to identify the plaintext password that produced the hash using a dictionary attack. This challenge demonstrates both the application of bcrypt as a password hashing algorithm and the risk of using predictable passwords regardless of the algorithm employed.

EVIDENCE

ARTIFACT	VALUE
Hash	`$2a$04$KMCaaiytS5OIsg2UZtthzugkZUPDqQ/Zyoys8XAY6AJVgirU/MWOS`
Algorithm	bcrypt (`$2a$`) with cost factor 4
Wordlist	`confident_dict.txt` — 75 entries, tech-themed compound words
Cracked password	**galactica_hash**
Attempts	1 of 5 allowed — cracked on the first wordlist pass

METHODOLOGY

HASH ANALYSIS

The hash prefix `$2a$04$` identifies it as bcrypt with a cost factor of 4 — a relatively low work factor, making it faster to crack than production bcrypt (typically cost 10–12). The hash was 60 characters, the standard bcrypt output length.

DICTIONARY ATTACK

The Python `bcrypt` library was used to test each word in the provided wordlist against the target hash.

```
import bcrypt

hash_val = b'$2a$04$KMCaaiytS5OIsg2UZtthzugkZUPDqQ/Zyoys8XAY6AJVgirU/MWOS'

with open('confident_dict.txt', 'r') as f:
    words = [w.strip() for w in f.readlines()]

for word in words:
    if bcrypt.checkpw(word.encode(), hash_val):
        print(f'CRACKED: {word}')
        break
```

The word `galactica_hash` was found at position **53 of 75** in the wordlist. The entire crack took under two seconds, due to the low bcrypt cost factor of 4.

```
FLAG
MetaCTF{galactica_hash}
```

THREAT ASSESSMENT

bcrypt is a strong, deliberately slow hashing algorithm designed to resist brute force attacks. However, even bcrypt cannot protect predictable passwords when an attacker has access to the hash and a relevant wordlist. The combination of a low cost factor (4) and a password drawn from a finite, themed wordlist made this hash highly vulnerable.

- Cost factor 4 is roughly 64× faster to crack than cost factor 10 — inadequate for production use
- Password drawn from a themed wordlist of only 75 entries — a trivial dictionary attack
- If this hash were obtained from a database breach, the plaintext would be recovered in seconds
- bcrypt correctly prevents rainbow table attacks — but dictionary attacks with relevant wordlists remain effective

RECOMMENDATIONS

- Use bcrypt with a minimum cost factor of 12 for password hashing in production systems
- Enforce password complexity requirements that make passwords resistant to domain-specific wordlists
- Implement account lockout and rate limiting to prevent online brute force attacks
- Use a password manager and ensure all passwords are randomly generated with high entropy
- Consider upgrading to Argon2id (winner of the Password Hashing Competition) for new systems
- Conduct regular password audits using offline cracking tools to identify weak hashes in your database
- Never use a cost factor below 10 in production — the extra computation is negligible for legitimate users but exponentially costly for attackers

CHALLENGE 4 · DIGITAL FORENSICS / FILE FORMAT ANALYSIS · THREAT LEVEL: LOW

HEXED

A file named `hexed.png` was provided that appeared to be a cursed image — it could not be opened as a standard PNG. The `file` command identified it as ASCII text rather than a binary image. The objective was to identify what had been done to the file, reverse the transformation, and recover the original PNG image containing the flag.

EVIDENCE

ARTIFACT	DESCRIPTION
hexed.png	ASCII text file — 1,910 lines of xxd-format hex dump
file command output	`hexed.png: ASCII text` — not a binary PNG
Magic bytes in hex	`89 50 4E 47 0D 0A 1A 0A` — a valid PNG signature
Recovered image	611×248 pixel RGBA PNG containing the flag text
Reconstructed size	30,547 bytes — a valid PNG binary

METHODOLOGY

FILE IDENTIFICATION

The `file` command immediately identified the anomaly — the file was ASCII text, not binary PNG data. Reading the first few lines revealed an xxd-format hex dump.

```
00000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452  .PNG........IHDR
00000010: 0000 0263 0000 00f8 0806 0000 00a0 f6e1  ...c............
```

The hex dump format is `[file offset]: [hex bytes] [ASCII representation]`. The magic bytes `8950 4e47` = `.PNG` confirmed this was originally a valid PNG file that had been converted to a hex dump — the curse.

HEX DUMP RECONSTRUCTION

A Python script parsed each line of the hex dump, extracted the hex byte values, and reassembled the original binary.

```
with open('hexed.png', 'r') as f:
    content = f.read()

data = bytearray()
for line in content.strip().split('\n'):
    if ':' in line:
        hex_part = line.split(':')[1].split('  ')[0].strip()
        hex_bytes = hex_part.replace(' ', '')
        data.extend(bytes.fromhex(hex_bytes))

with open('hexed_fixed.png', 'wb') as f:
    f.write(data)
```

The script reconstructed exactly 30,547 bytes. The first eight bytes (`89 50 4E 47 0D 0A 1A 0A`) confirmed a valid PNG header. Opening the image revealed the flag displayed in green pixel text on a light background.

FLAG

`flag{h3xdump_15n7_4_cur53}`

The flag text itself — `h3xdump_15n7_4_cur53` — is a leetspeak rendering of “hexdump isn’t a curse”, confirming the intended solution.

KEY CONCEPTS

FILE SIGNATURES / MAGIC BYTES

Every file format has a unique signature in its first few bytes that identifies its type regardless of the file extension. The `file` command reads these magic bytes to determine the true file type.

MAGIC BYTES (HEX)	FILE TYPE	ASCII
`89 50 4E 47 0D 0A 1A 0A`	PNG image	.PNG....
`FF D8 FF`	JPEG image	...
`50 4B 03 04`	ZIP archive	PK..
`4D 5A`	Windows EXE/DLL	MZ
`7F 45 4C 46`	Linux ELF binary	.ELF

RECOMMENDATIONS

- Never trust a file extension — verify the true type from its magic bytes before opening or processing it
- Treat a text file carrying binary magic bytes as an encoding or exfiltration channel, not a corruption

- Automate file-type verification in intake pipelines so mismatches are surfaced rather than silently accepted

---

ECHOCLUB DIGITAL FORENSICS EXAMINATION REPORT — VOLUMES I AND II

Supervising writer: **Digital Orukami** · MetaCTF · Antisyphon Training · educational use only.