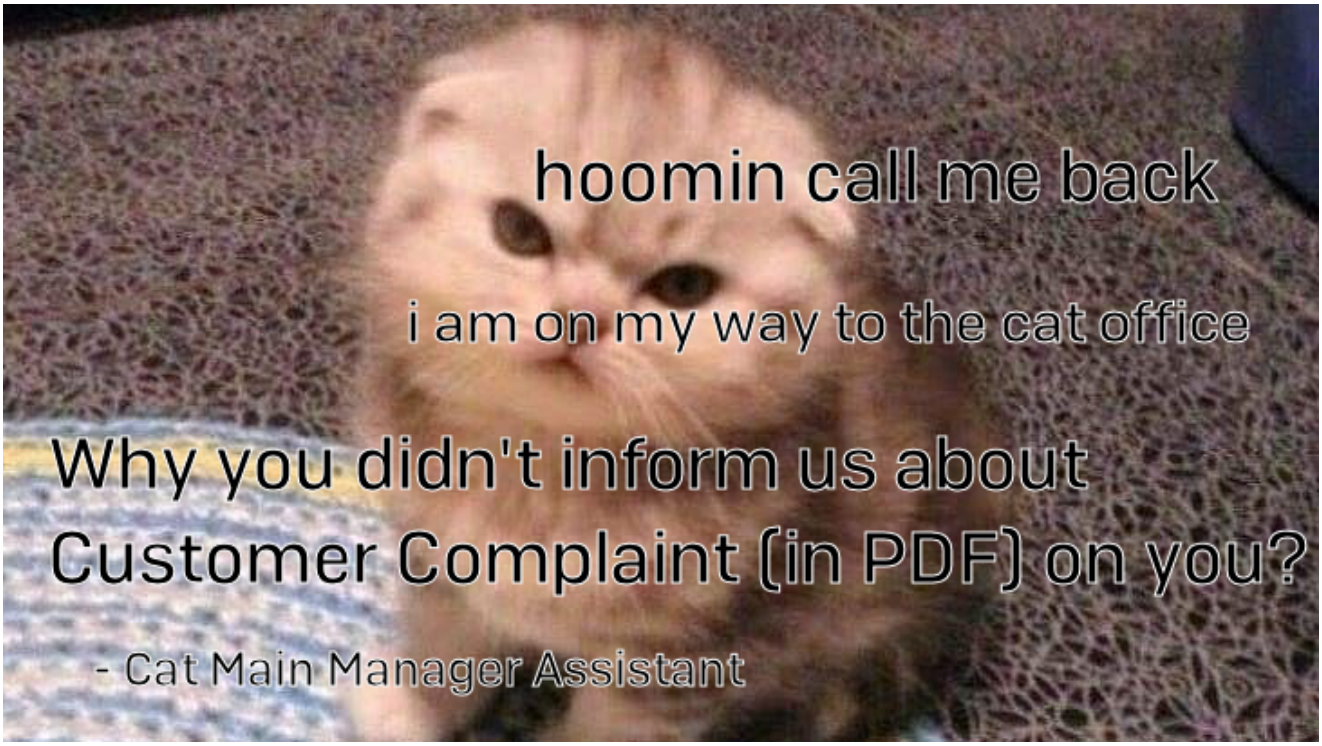
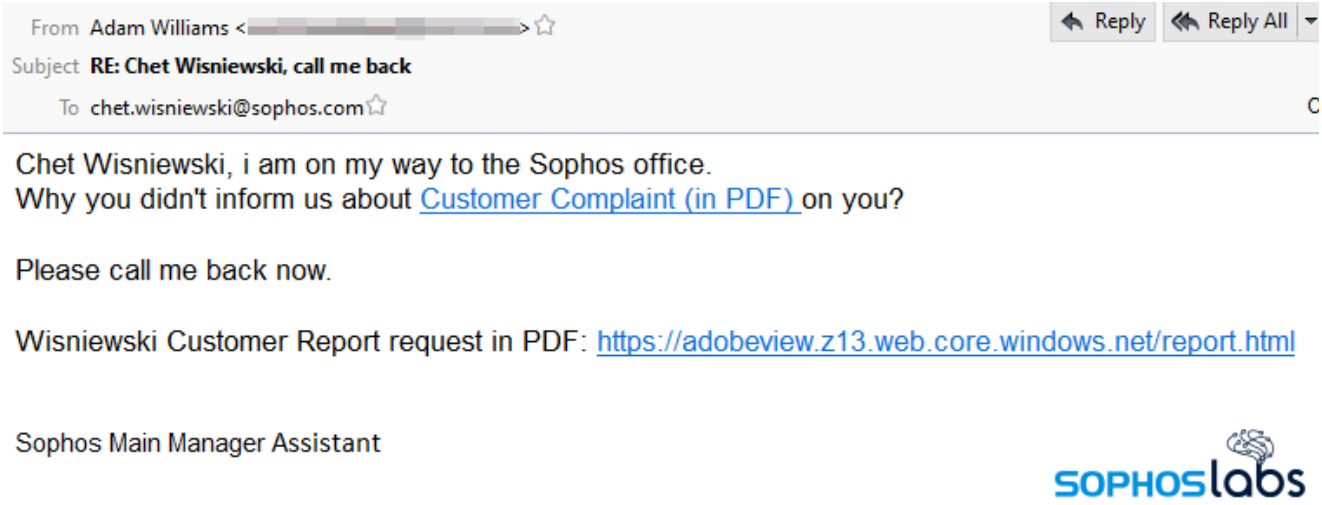
BazarLoader ‘call me back’ attack abuses Windows 10 Apps mechanism

news.sophos.com/en-us/2021/11/11/bazarloader-call-me-back-attack-abuses-windows-10-apps-mechanism/

Andrew Brandt November 11, 2021



The emails that flooded into inboxes last week came from someone named Adam Williams, who sounded *annoyed*. “Andrew Brandt, i am on my way to the Sophos office. Why you didn’t inform us about Customer Complaint (in PDF) on you? Please call me back now.” the email addressed to me read.



I later heard from my colleague Chet that he had also received an email from an Adam Williams, but with a different email address in the From: header.

But the messages did not come from the “Sophos Main Manager Assistant,” because such a role doesn’t even exist. It originated with a threat actor. The payloads, belonging to a malware family variously known as *BazarBackdoor* and *BazarLoader*, were delivered by abusing a novel mechanism – at least, one that my colleagues and I were unfamiliar with: The Windows 10 Apps installer process.

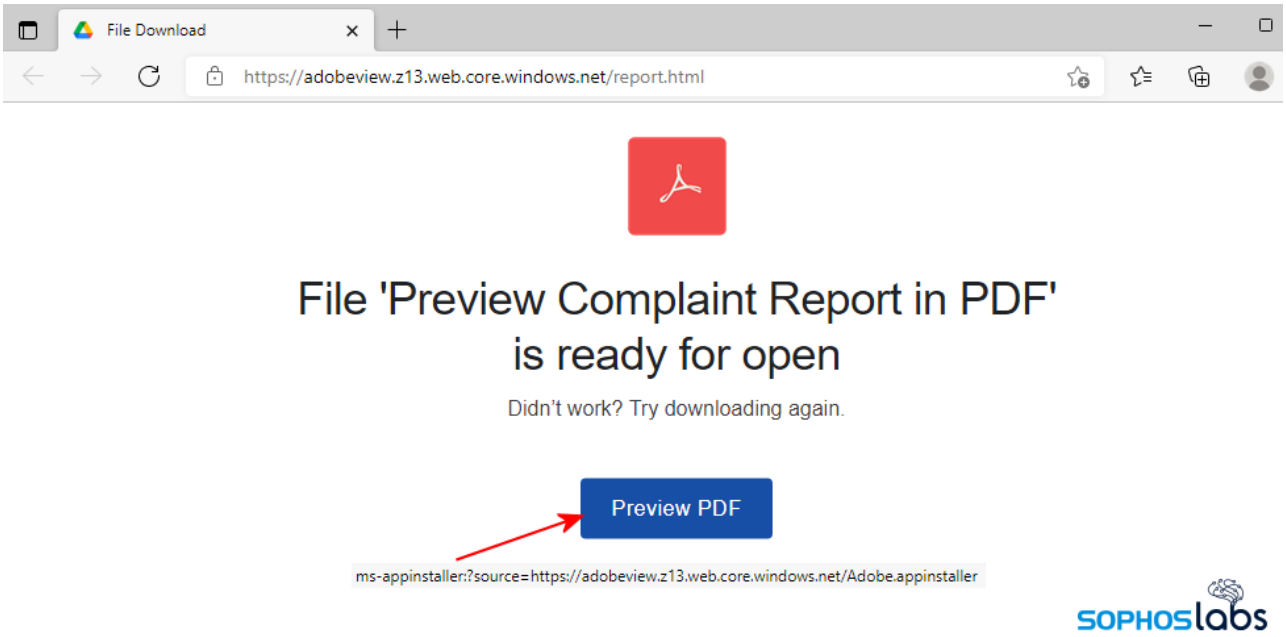
The spam targeted people around the world. Many customers correctly identified them as a malicious spam and reported them to us. While the attacker’s websites were still operational (Microsoft turned off the pages hosting the malicious files late in the evening of November 4), I pulled down several samples of the files and then stepped through the attack while recording the network traffic, collecting behavioral data from the machine, and taking screenshots.

Here’s what we found.

A careful lure

The messages themselves were very short, but they were crafted with an understanding of the human psychology behind the adrenaline-rush of fear, and had been personalized with both the name of the recipient and the targeted organization in both the subject line and the body. The spam trope here – a complaint, filed against you, and the insinuation that you’ve been attempting to cover it up.

The messages urge the recipient to click through to a website that, purportedly, is where the complaint has been posted for you to review.



The link points to an ms-appinstaller object

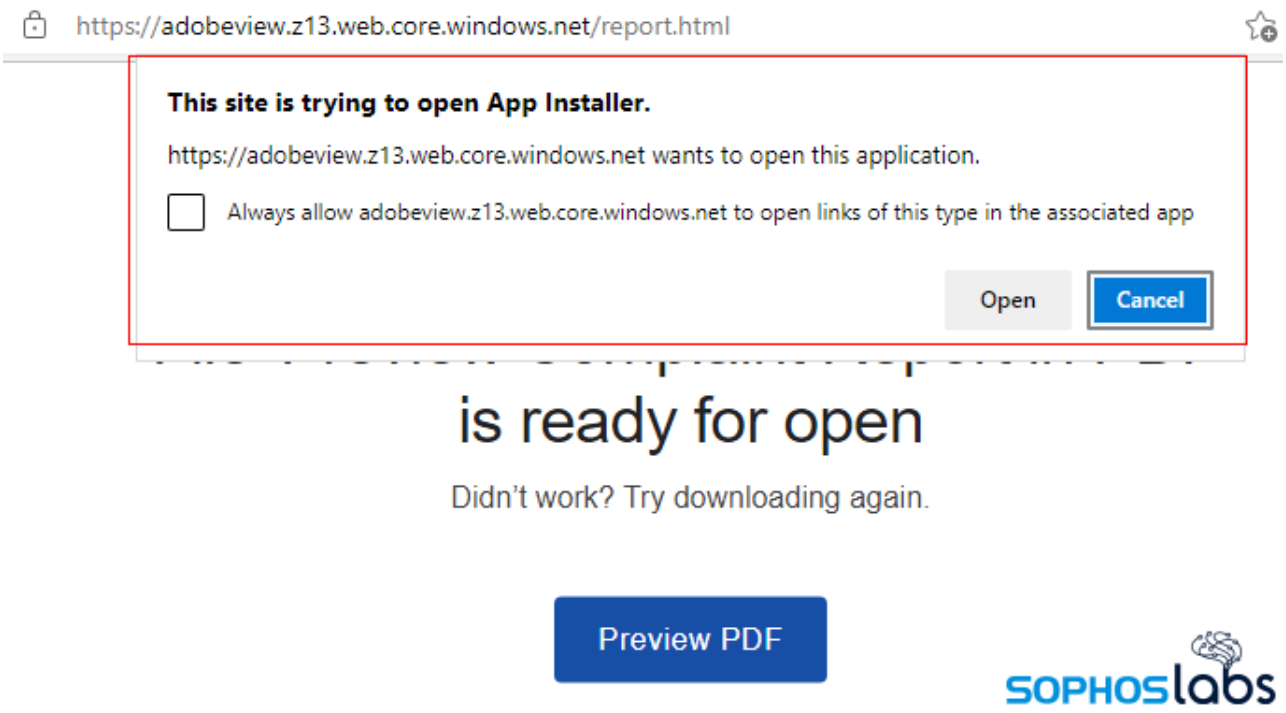
But there's something amiss with this link: Instead of being prefixed with the expected **https://** the link instead begins with what was (for me, at least) an unfamiliar **ms-appinstaller:** prefix. In the course of running through an actual infection I realized that this construction of a URL triggers the browser (in my case, Microsoft's Edge browser on Windows 10) to invoke a tool used by the Windows Store application, called *AppInstaller.exe*, to download and run whatever's on the other end of that link.

What's interesting about this is that there are actually two stages, apparently, to this phase of the attack. The link points to a 482-byte text file named **Adobe.appinstaller**. The contents of that file is just plain text, in xml format, that points to a URL where a larger file containing the malware, named **Adobe_1.7.0.0_x64.appbundle**, was located.

The attackers used two different web addresses for hosting this fake "PDF download" page throughout the day. Both pages were hosted in Microsoft's cloud storage, which perhaps lends it a sense of (unearned) authenticity, and both the .appinstaller and .appbundle files were hosted in the root of each webpage's storage.

The website also hosted all the payload components
But we can't learn how attacks work by doing the right thing, so of course, we clicked Open, which you should never do.

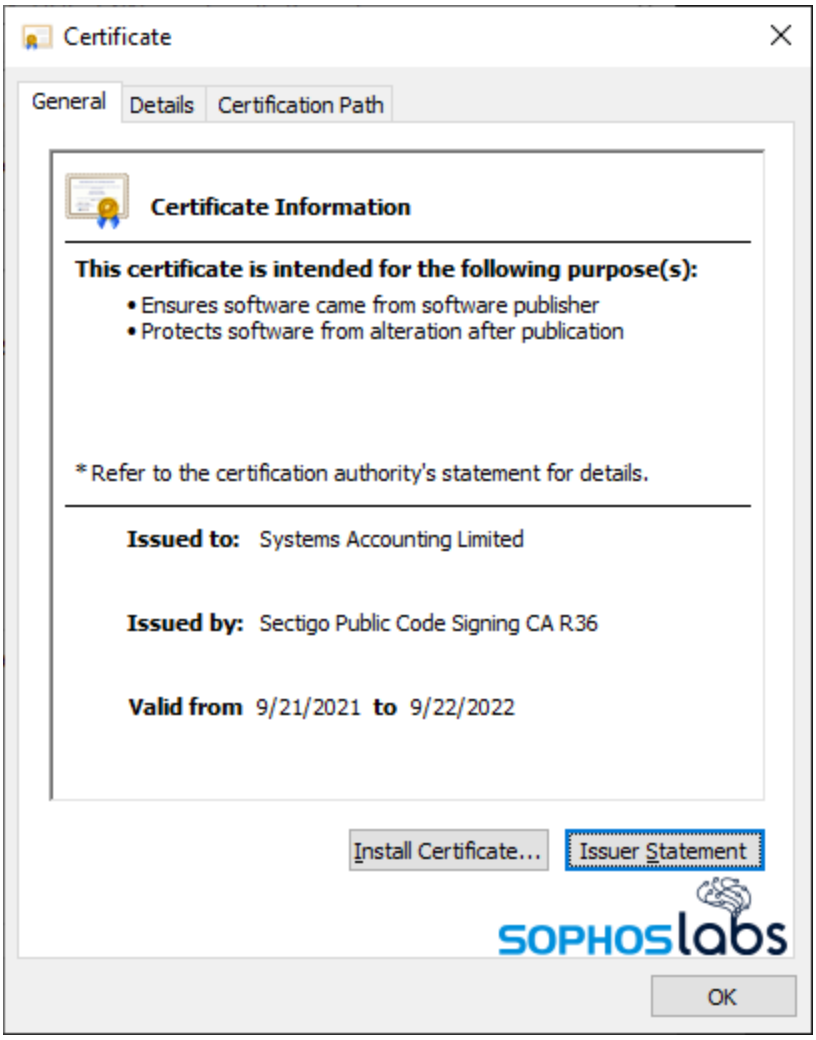
When I did that, Edge prompted me with a warning that *This site is trying to open App Installer*. Since this site's subdomain was **adobeview** and the root domain was windows.net, it stands to reason that a user who is unaware of how an arcane aspect of the operating system works could easily be fooled into not only clicking the *Open* button in that dialog box but might also fill in the *Always Allow* checkbox.



Throughout the morning, I repeatedly retrieved the *.appbundle* file itself, directly, getting several different versions. But to learn the real magic of how this attack works I needed to let it play itself out, so I just took the same steps a potential victim might take, and just clicked the link in the page.

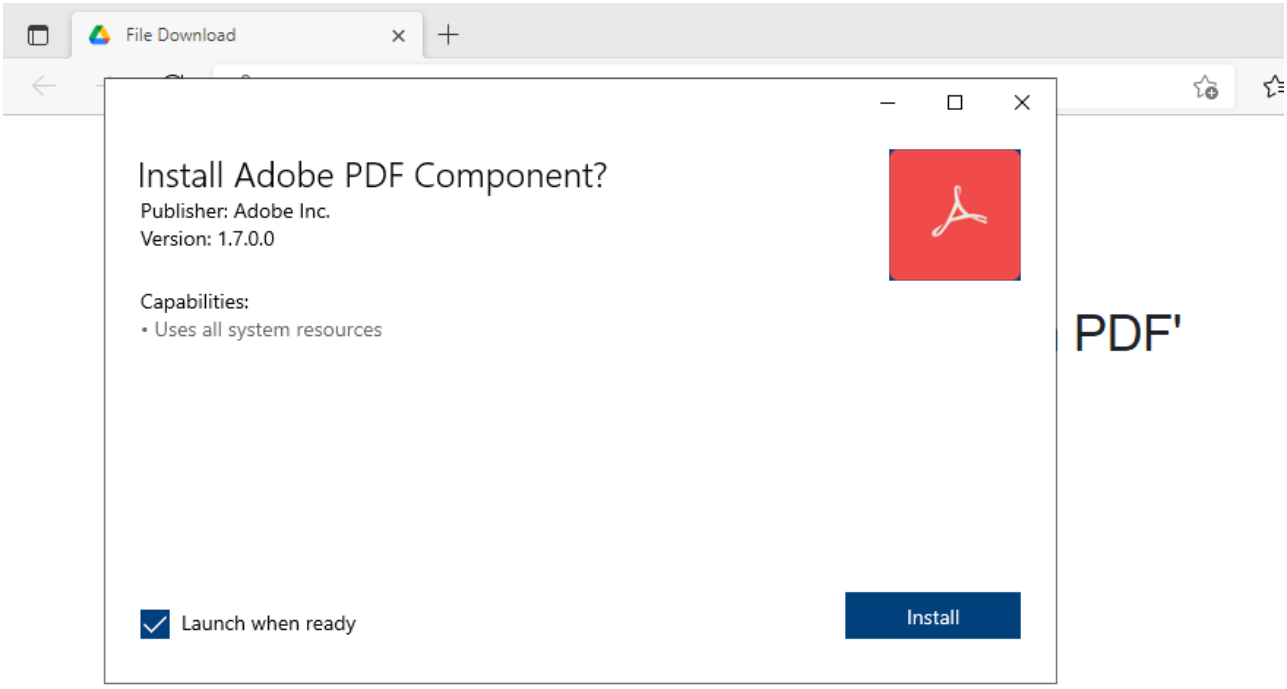
When appinstaller attacks

The initial file referenced in the webpage was this .appinstaller. As you can see, it contains not only a link to the actual payload (the .appxbundle URL at the bottom) but also information about a publisher's digital signature. In this case, the malicious appinstaller indicates that the .appxbundle was digitally signed by a company calling itself Systems Accounting Limited, based in the UK. The certificate was issued just a few months ago. Sophos has contacted Sectigo to alert them about this abuse of the certificate they issued.



(Out of curiosity, I did check and there is, in fact, a business registered in the UK's Companies House by that name and in the same county listed in the digital certificate referenced in this file. However, the company's registered address points to a private residential home, raising serious doubt that this company had anything to do with the attack. The domain **systems-accounting.com**, embedded in the certificate's admin, is hosted on an IP address on which every other domain hosted there has a .ru TLD.)

Clicking *Open* in the App Installer warning dialog prompts the browser to invoke AppInstaller.exe, which downloads the .appinstaller file, then the .appxbundle linked inside of it.



When the .appxbundle file has been downloaded, the installer prompts the user to begin installing it. The process only took a few seconds.

Taking a closer look inside of the .appxbundle file, which is just a .zip archive containing several other files, the AppsManifest.xml file contains the textual information shown on the installation screen. It also clearly references the certificate from Systems Accounting Limited, but the framework doesn't show (or validate) that certificate's information on this screen. In fact, the attacker simply added individual display properties for the program's name ("Adobe PDF Component"), publisher ("Adobe Inc."), and an Adobe Acrobat logo graphic stored in a subfolder.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Package IgnorableNamespaces="uap mp rescap build" xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
xmlns:mp="http://schemas.microsoft.com/appx/2014/phone/manifest" xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities"
xmlns:build="http://schemas.microsoft.com/developer/appx/2015/build">

  <Identity Name="d15c6d37-d51b-498a-a1af-143d07014234" Publisher="CN=Systems Accounting Limited, O=Systems Accounting Limited, S=Essex, C=GB"
Version="1.7.0.0" ProcessorArchitecture="x64"/>
  <Properties>
    <DisplayName>Adobe PDF Component</DisplayName>
    <PublisherDisplayName>Adobe Inc.</PublisherDisplayName>
    <Logo>Images\StoreLogo.png</Logo>
  </Properties>
  <Dependencies>
    <TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.17134.0" MaxVersionTested="10.0.18362.0"/>
    <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.17134.0" MaxVersionTested="10.0.18362.0"/>
  </Dependencies>
  <Resources>
    <Resource Language="EN-US"/>
    <Resource uap:Scale="200"/>

  </Resources>
  <Applications>
    <Application Id="App" Executable="UpdateFix\SecurityFix.exe" EntryPoint="Windows.FullTrustApplication">
      <uap:VisualElements DisplayName="Adobe PDF Component" Description="Adobe PDF Component Update" BackgroundColor="transparent"
Square150x150Logo="Images\Square150x150Logo.png" Square44x44Logo="Images\Square44x44Logo.png">
        <uap:DefaultTile Square71x71Logo="Images\SmallTile.png" Square310x310Logo="Images\LargeTile.png" Wide310x150Logo="Images\WideTile.png">
```

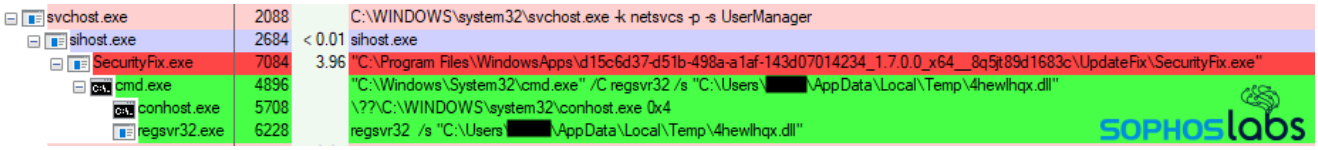
The manifest file clearly shows the signing certificate name Systems Accounting Limited, alongside a properties value that falsely identifies the publisher as Adobe

The .appxbundle file contains a malicious executable nested in the **Adobe_1.7.0.0_x64.appx** file stored inside of it. That file is also a .zip archive, and contains many of the same xml files, as well as a subfolder named **UpdateFix** inside of which is the malicious payload, **SecurityFix.exe**.

Even though the .appxbundle contains the signing certificate assigned to Systems Accounting Limited, the SecurityFix.exe payload is not, itself, signed. The certificate references a domain (systems-accounting.com) that was registered on September 8, and the code signing certificate was issued on September 21.

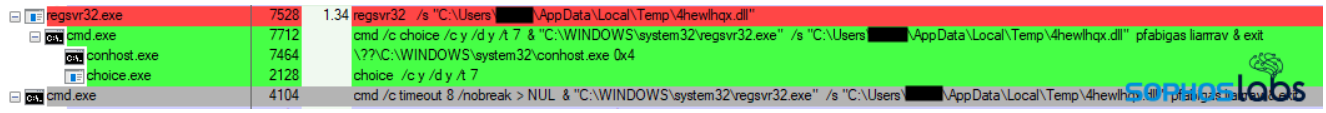
Injection games

After the dog-and-pony show of the fake installer running, the system ran the SecurityFix executable, which downloaded a DLL (from the same server that hosted the .appxbundle file) into the %temp% directory and then runs it using regsvr32.exe. It ran for a few seconds, then spawned a child process of itself running the same way, but with random-looking function calls at the end of the command.



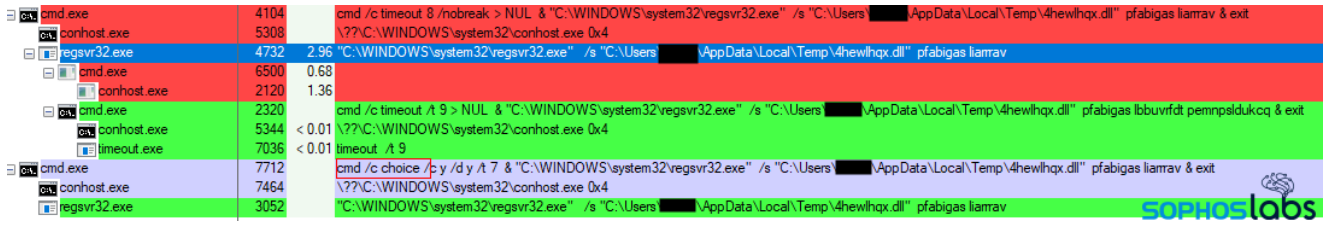
SecurityFix.exe pulls down and runs the first instance of the malicious DLL

It's not easy to see unless you're using an alternative Task Manager tool (like Process Explorer), but the DLL ran for a few seconds and then spawned yet another a child process of itself, using a program called Timeout.exe. Timeout is a legitimate Windows console application used for manually configuring a delay starting another program. In this case, it waited 8 seconds and then ran the same regsvr32 command, with the same random function calls, and with an additional "& exit" at the end of the command.



And then that execution spawns yet another child process, in which the attackers have used a different timeout method to delay execution: The choice.exe console command lets you specify a timeout just like Timeout.exe does. With a 7 second delay, it runs for the fifth time in rapid succession.

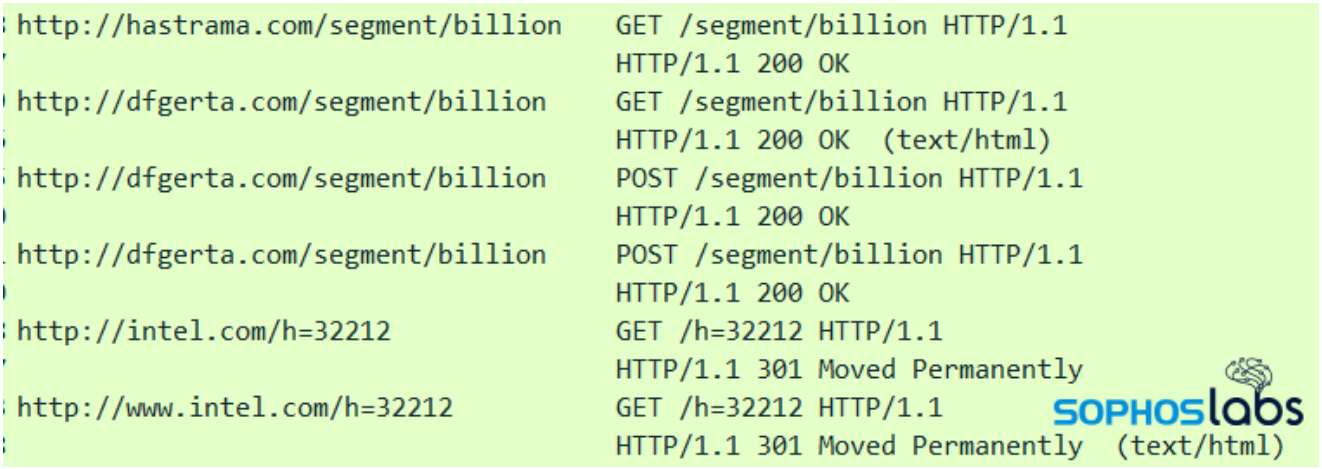
By this time, the DLL terminates and an instance of the Edge browser (Chromium version) spawns, with the code injected into a headless instance of msedge.exe. The malware is now fully installed, and begins beaconing to its command-and-control servers.



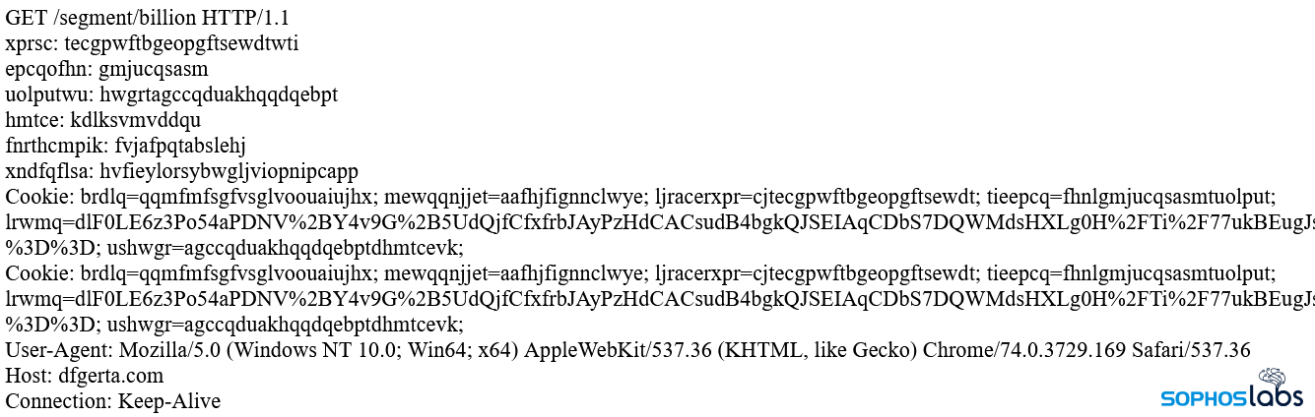
Telephone game with cookies

The malware that eventually was installed is BazarBackdoor. We know this because of a few things: The malware has a distinctive style in the patterns it follows for its command-and-control traffic, and the detections we've developed that read its in-memory behavior positively identified it by the definition name **Mem/BazarLd-C**.

Like many other malware, BazarBackdoor (and its related sibling BazarLoader) communicates over HTTPS, but the way that it transmits and receives instructions are distinctive. And it generates a lot of noisy, unnecessary traffic — to pages that don't exist — on unrelated websites (such as Intel, shown at the bottom of the screen below), usually with a query string that has five or six numbers at the end.



In brief, the malware uses "cookies" in the HTTPS GET or POST headers to transmit information to the server, and receives commands from the C2 in the form of one or more "Set-Cookie" response headers.

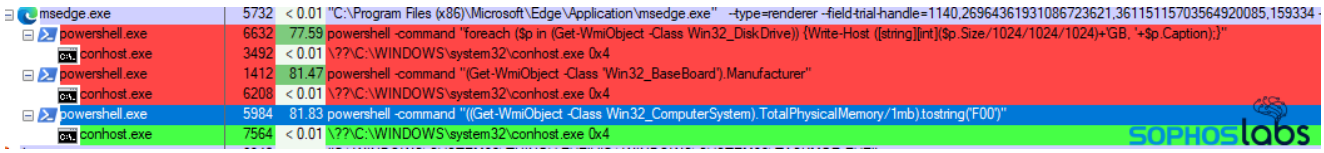


Cookie data transmitted in the GET request's header exfiltrates information from the infected device

During this attack, the malware used specific URI paths – the same one for all the requests. The sample I ran for an extended period used **/segment/billion** , but other industry analysts shared that their samples used the URI paths of **/recite/drink** or **/mission/revolt** or **/discreet/marble** or **/note/actual.**

```
HTTP/1.1 200 OK
Server: nginx/1.14.2
Content-Type: text/html; charset=UTF-8
Content-Length: 2
Connection: keep-alive
Date: Thu, 04 Nov 2021 18:57:53 GMT
Set-Cookie: mode2=d4qHAUp8LDtH8rBsdYknhJMK8MV8C5oWyJtoAx1Wlw2UKpx4vOB7K
Set-Cookie:
alligment=2WoShje4PjZ6D5pvs7nC2Q8eNfHB2r3iHKdDupnD6Q9PLiMPwMGP8ieTPWVrFxWhv6eW
Set-Cookie:
clearing=dtgGS2CKOKK9jbb7mxtyGT9m%2FO%2BKZZRTMuklREAnN4jMLlUSLBiW2Fkg4nR3t5N
%3D%3D
Set-Cookie: uTrack=RU6khLq18rx6K79W
Set-Cookie: BDATA=621656
Set-Cookie: cvalue=false
Set-Cookie:
dValue4=qbf9KXgFrlR3jiEx4FypQp5YqQPOkIqVW7VrGVh8R7tw6WYEhglmQDOrUKuZPbufttAGOl
Set-Cookie: notfirst=dAwazeiJ3HAE08Na
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
```

Set-Cookie items in the response headers from C2 check-ins contain instructions for the bot
In this instance, the C2 sent a series of commands in quick succession that performed a number of different profiling activities on the infected system. It was pretty easy to see this happening because the headless Edge process kept spawning instances of PowerShell or other tools.



A sequence of PowerShell commands executed by the malware, which had been injected into the Edge browser, profiles the hard disks, processor and motherboard, and RAM on the infected system

These three queries profiled the hard drive, processor information, and RAM on the system. The malware, running in the context of the headless Edge process, also ran other commands like **net view /all** to learn more about servers on the network where it's installed.

A PowerShell command retrieves the public-facing IP address of the network where the infected computer is running from one or more websites at random

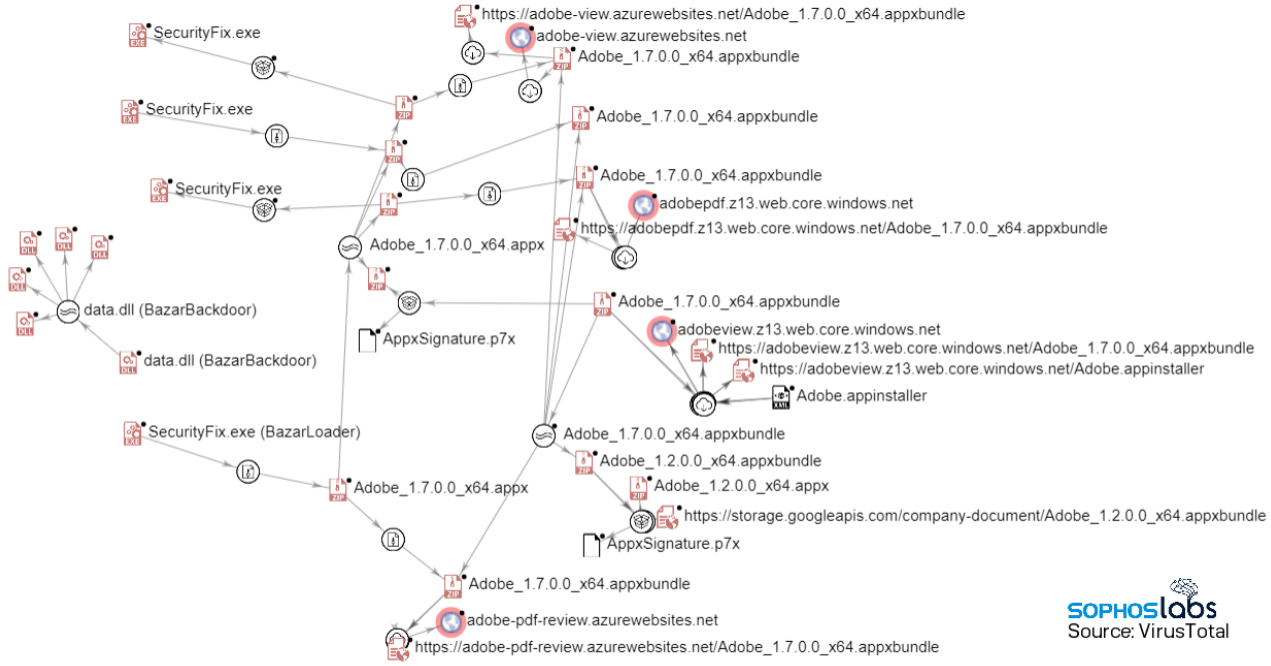
And this very long PowerShell command chooses one or more of these URLs, at random, and uses it to identify the public-facing IP address of the network where the system is located.

```
powershell -executionpolicy bypass -command
"$Servers=@('http://checkip.amazonaws.com','https://ipinfo.io/ip','http://api.ipify.org',
'https://myexternalip.com/raw','http://wtfismyip.com/text',
'http://ip.anysrc.net/plain/clientip','http://api.ipify.org/?format=text',
'http://api.ip.sb/ip','http://ident.me/ip');$i=Get-Random -Minimum 0 -Maximum 8; <#Write-
Host HTTP-DNS request via $Servers[$i];#>try { $ip=Invoke-WebRequest -UseBasicParsing -
Uri $Servers[$i]; Write-Host $ip.content -NoNewline; }catch { <#Write-Error
$_.Exception.Message;#> Write-Host '' -NoNewline; }"
```

The PowerShell command used to identify the public-facing IP address. It was too long to show the entire command in the Process Explorer screenshot, above.

Avoidance and detection

Malware that comes in AppX packages is novel, but now that the process has been demonstrated, it's likely to be here to stay. These apps are supposed to be digitally signed with certificates, but it doesn't appear that there's any mechanism to make a sanity check between what's on the certificate and the code it's supposed to certify.



A map of the relationship of web hosts to malware payloads. Data: VirusTotal

Our advice might be to ignore emails from random addresses which claim to originate from within your company, but that's a problem the spammers are likely to solve, probably by forging the From: addresses. The right thing to do would be to check with someone in the relevant department, perhaps HR, if you suspect the email might be legitimate, and to ignore it if it has these signs of obvious forgery.

Obviously, Microsoft needs to work on the mechanism that validates not only whether code contains an authentic digital certificate, but if there's any logical connection between the certificate and the organization purportedly behind the program. Right now, that mechanism doesn't offer any guarantees – and in fact may make it easier for an attacker to just pretend to be any company they want.

Users of Sophos endpoint products will be protected from this malware at multiple stages of the process: The SophosXL reputation service is blocking the source and C2 addresses, and endpoint protection will detect various elements of this infection as **Troj/Bazar-T, Troj/Bazar-S, Troj/DwnLd-TA, Troj/DwnLd-TE, Troj/MSIL-RYU, Troj/MSIL-RYT,** and/or **Troj/MSIL-RXW** if the files are found on disk, pre-execution, or as **Mem/BazarLdr-C** if it evades static detection, and through behavioral detection of unexpected PowerShell commands running in the context of a browser process.

SophosLabs has published IoCs relating to this attack on its Github page, and wishes to thank Microsoft and Sectigo for a prompt response to the attack method.