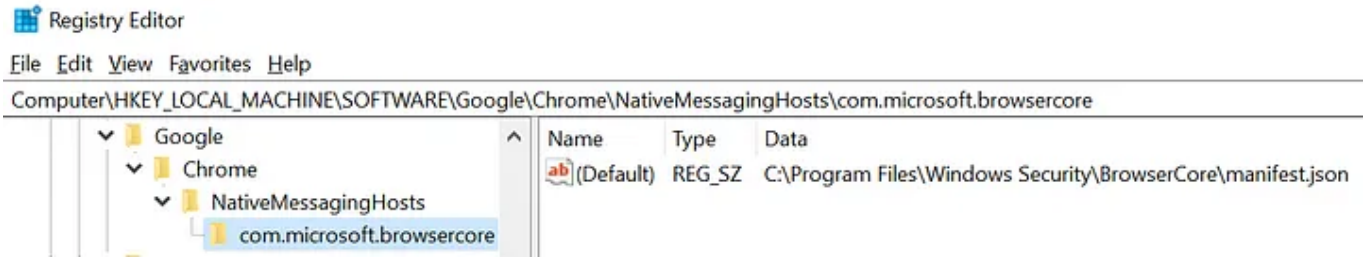
Requesting Azure AD Request Tokens on Azure-AD-joined Machines for Browser SSO

I installed the Windows 10 Accounts extension in Chrome on an Azure-AD-joined machine. Upon doing so, I could now login to Azure-AD-authenticated services like portal.azure.com without needing to enter my password! I thought that being able to generate Azure AD authentication material was a useful piece of tradecraft to have, so I began reversing how the Chrome extension works.

The Windows 10 Accounts Chrome extension communicates with Windows using the `chrome.runtime.sendNativeMessage` function from the Chrome Native Messaging APIs:

created in the registry when Chrome is installed:



Looking at the JSON manifest specified there, we can see that the actual application that the extension communicates with is is BrowserCore.exe

how Chrome or BrowserCore would actually implement that, so to maybe give me more clues about what's going on, I thought I'd just use the extension and see the messaging dance in action.

First attempt was to open Process Explorer, try and open Chrome's incognito window repeatedly with the extension enabled, and try and authenticate a bunch of times - racing the process to see if I could click on it before it closed. Ultimately, my clicks (and Process Explorer) were too slow to catch BrowserCore.exe before it exited. However, since I have process and command line logging enabled for occasions exactly like this (like any

Next I wanted to know what BrowserCore.exe was actually doing to get the refresh token. So I cracked open dnSpy, crossing my fingers that it'd be a .NET executable since my binary reversing skills are crap and was hoping the fates would be nice to me today, but was sadly disappointed - it wasn't .NET. Okay fine. I put a couple extra sticks of RAM into my machine, opened Ghidra, (re)figured out how to setup symbols in Ghidra, and started looking at a bunch of windows that I mostly don't know how to use. One of the first things I do when looking at an unknown binary is look at the imports, and while doing that with BrowserCore.exe I noticed it imports `CoCreateInstance`:

Looking up the CLSID `{a9927f85-a304-4390-8b23-a75f1c668600}` in the registry revealed our friendly neighborhood COM DLL MicrosoftAccountTokenProvider.dll:

1. Obtain access to a Azure AD user context on an Azure-AD-joined device. An easy way to tell is to run the command `dsregcmd.exe /status`. If this is abusable, there will be a section titled “SSO State” and `AzureAdPrt` will be set to YES.
2. On an Azure-AD-joined machine where an Azure AD user has logged in, run RequestAADRefreshToken.exe. This will return the refresh token.

6. Refresh the page (or visit https://login.microsoftonline.com/login.srf again) and you'll be logged in. How neat is that? That's pretty neat!

- MicrosoftEdge.exe - ServerName:MicrosoftEdge.AppXdnhjhccw3zf0j06tkg3jtqr00qdm0khc.mca

TraceLogging/Event Tracing for Windows (ETW)

TraceLogging provides some telemetry when executing RequestAADRefreshToken. While reversing MicrosoftAccountTokenProvider.dll I noticed the code was littered with trace

Finding references to EventRegister via the file's PE import table

```
$Events = Start-EtwTrace `
    -ProviderGuid '05f02597-fe85-4e67-8542-69567ab8fd4f' `
    -OutputFile .\out3.evtx `
    -ProcessPath C:\RequestAADRefreshToken.exe

$Events | select
ProcessId,Id,LevelDisplayName,ProviderName,KeywordsDisplayNames,Me
ssage | ft -AutoSize
```

Summary

Regardless of the technology, network defenders should be diligent in their understanding of their authentication systems and how users and administrators can access the system's credentials. As more and more Windows hosts are using Azure AD for authentication, it's important to understand authentication changes on these machines and what new credential sources might be on these machines. RequestAADRefreshToken.exe takes advantage of one of these "new" credentials and grants an attacker the ability to generate a user credential