MSBuild Injection

22-26 FEB 21

Tasking

Inject arbitrary code into a running MSBuild process to be packaged with the resultant binary without negatively affecting its stability or touching disk.

Injection Potential

First, this week was a massive learning curve working with debuggers to reverse engineer and follow MSBuild as it operated. Pairing this with some open source content and a good amount of sysinternals magic rendered the following.

MSBuild leverages .props and .targets sheets to carry most of its configurations. At the beginning of the process, it will check the project file (e.g. .vcxproj) for any to be loaded in. Each of those sheets may carry their own dependencies as well, all to provide a comprehensive customization to the build process. As they are loaded into msbuild.exe, these sheets offer a robust plaintext injection point likely best used for including attacker-controlled files or simply undermining/removing security features the binary is to be built with.

For a mixed C/C++ project, the MSBuild process (msbuild.exe) of course has a clear two-stage workflow of compiling (cl.exe) and linking (link.exe). In the compiling stage, cl.exe will delegate a portion of the work to a front-end c1.exe or c1xx.exe and back-end c2.exe. Here is almost certainly the most opportune place to inject. The attacker can do absolutely anything they want in plain text and generate header dependencies for the same potential as the first stage. Additionally, most of these intermediate files are stored in the temp dir, only to be retrieved later for the next stage. As the user has full control over the files, injection here would be trivial at little risk of racing to modify before retrieval.

The linker will delegate to cvtres.exe to compile C++ resource files into objects that can be linked in. Although there are other assets being leveraged, none have a noticeable presence or have been identified as a reliable injection point. I am unsure of the ease of injecting with these files, which are also stored in the temp dir.

Of note, a lot was missed here and many of these injection points are quasi-TOCTOU attacks via files on disk—out of scope for this project. However, the potential is still staggering, affording a lot of opportunity to favor attack that can be used across targets. For example, targeting c1.exe or c1xx.exe leaves only about a 50/50 to hit. However, targeting cl.exe will affect all MSVC compilations, and hitting msbuild.exe will of course be a full break to the platform.

Future Plans

Make some POCs.

1. Start by hooking ReadFile calls to reliably filter out and handle junk coming in.
2. Inject more code as we go, then expand to different stages of the build.
3. Finally, attempt to inline/hook in PIC during compilation.