Classification: Internal — PCSIRT / Red Team Use Only
Author: Red Team Operations
Date: February 2026 | Version: 1.1

1. Objective

Validate the PCSIRT detection signal for `msbuild.exe` abuse across three threat categories:

- **Manual MSBuild execution** from interactive command prompts
- **Atypical project file extensions** (`.csproj`, `.xml`, `.rsp`, `.proj`)
- **Nonstandard command-line usage** deviating from normal developer build workflows

Each test case includes the expected telemetry the detection should capture, enabling the PCSIRT team to confirm signal fidelity and identify coverage gaps.

2. Scope & Prerequisites

In-Scope Systems

- Windows 10/11 endpoints with .NET Framework 4.x+ or .NET SDK installed
- Endpoints with EDR/SIEM telemetry forwarding enabled

Prerequisites

- MSBuild available at standard paths: `C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe` or Visual Studio install path
- Test user account (non-admin preferred for baseline; admin for escalation tests)
- Purple team coordination channel open with SOC/PCSIRT
- **Change ticket approved** and test window scheduled

• Sysmon (or equivalent) configured with ProcessCreate (Event ID 1), FileCreate (Event ID 11), and command-line logging

Safety Controls

• All test payloads are **benign** (calc.exe, whoami, or write to a local temp file)
• No persistence mechanisms are installed
• All test artifacts are cleaned up post-execution
• Tests should be conducted on isolated/lab endpoints first

3. Test Matrix Overview

Test ID	Category	Description	MITRE ATT&CK
MSB-01	Manual Exec	MSBuild from cmd.exe	T1127.001
MSB-02	Manual Exec	MSBuild from powershell.exe	T1127.001
MSB-03	Manual Exec	MSBuild from non-standard parent (wscript)	T1127.001
MSB-04	Atypical Files	Inline task in .csproj file	T1127.001
MSB-05	Atypical Files	Inline task in .xml file	T1127.001
MSB-06	Atypical Files	Response file (.rsp) execution	T1127.001
MSB-07	Atypical Files	Custom .proj file with UsingTask	T1127.001
MSB-08	Nonstandard CLI	Execution from temp/user directories	T1127.001
MSB-09	Nonstandard CLI	Unusual flags & verbosity suppression	T1127.001
MSB-10	Nonstandard CLI	MSBuild with no VS/SDK context	T1127.001
MSB-11	Nonstandard CLI	MSBuild spawning suspicious child processes	T1127.001
MSB-12	Nonstandard CLI	Logger DLL sideloading via /logger: switch	T1127.001
MSB-13	Negative Test	Legitimate developer build (should NOT alert)	N/A

4. Benign Payload Templates

All test cases reference these project file templates. Save each to the test endpoint before starting. The test kit includes a `deploy.ps1` script that stages all files automatically.

4A. Inline C# Task — calc.exe (benign_calc.csproj)

```
<Project ToolsVersion="4.0"
    xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="TestTarget">
    <TestTask />
  </Target>
  <UsingTask TaskName="TestTask"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.NET\Framework64\
      v4.0.30319\Microsoft.Build.Tasks.v4.0.dll">
    <Task>
      <Code Type="Fragment" Language="cs"><![CDATA[
        System.Diagnostics.Process.Start("calc.exe");
      ]]></Code>
    </Task>
  </UsingTask>
</Project>
```

4B. Inline C# Task — whoami to file (benign_whoami.xml)

Same structure as 4A but writes the current username to `C:\Temp\msbuild_test_output.txt`. Uses `.xml` extension to test extension-based filtering evasion.

4C. Response File (test.rsp)

```
/nologo
/verbosity:quiet
benign_calc.csproj
```

4D. Logger DLL Source (TestLogger.cs)

```
using Microsoft.Build.Framework;
using System; using System.IO;

public class TestLogger : ILogger
{
    public LoggerVerbosity Verbosity { get; set; }
    public string Parameters { get; set; }

    public void Initialize(IEventSource eventSource)
    {
        string output = $"[MSB-12 TEST] Logger DLL loaded"
            + $" at {DateTime.Now} by {Environment.UserName}";
        File.WriteAllText(
            @"C:\Temp\logger_dll_test_output.txt", output);

        var psi = new System.Diagnostics.ProcessStartInfo(
            "cmd.exe",
            "/c whoami >> C:\\Temp\\logger_dll_test_output.txt")
        { CreateNoWindow = true, UseShellExecute = false };
        System.Diagnostics.Process.Start(psi);
    }
    public void Shutdown() { }
}
```

4E. Legitimate Build Project (legit_build.csproj)

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
  </PropertyGroup>
</Project>
```

Accompanied by a simple `Program.cs` for MSB-13 negative testing.

5. Detailed Test Procedures

MSB-01: Manual MSBuild from cmd.exe

Objective:

Detect MSBuild launched interactively from `cmd.exe` executing an inline task.

Steps:

1. Open `cmd.exe` as the test user
2. Execute: `C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe C:\Temp\benign_calc.csproj`
3. Confirm `calc.exe` launches
4. Record timestamp

Expected Telemetry:

- Process Create: `MSBuild.exe` with parent `cmd.exe`
- Command line contains a `.csproj` path outside a typical source/build directory
- Child process: `calc.exe` spawned by `MSBuild.exe`

EXPECTED: Detection Signal Should Fire: **YES**

MSB-02: Manual MSBuild from PowerShell

Objective:

Detect MSBuild launched from `powershell.exe` or `pwsh.exe`.

Steps:

1. Open PowerShell as the test user
2. Execute: `& "C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe" C:\Temp\benign_calc.csproj`
3. Repeat with `pwsh.exe` if installed
4. Record timestamps

Expected Telemetry:

- Process Create: `MSBuild.exe` with parent `powershell.exe` or `pwsh.exe`
- Command line referencing project file in non-development directory

EXPECTED: Detection Signal Should Fire: **YES**

MSB-03: MSBuild from Non-Standard Parent Process

Objective:

Detect MSBuild spawned by unusual parent processes (simulates malware dropper behavior).

Steps:

1. Create `launch_msbuild.vbs` (included in test kit)
2. Double-click the .vbs file — parent will be `wscript.exe`
3. Alternatively, create and run a scheduled task (parent = `svchost.exe` / `taskhostw.exe`)
4. Record parent process chains

Expected Telemetry:

- `MSBuild.exe` with parent `wscript.exe`, `svchost.exe`, or `taskhostw.exe`
- These parent processes almost never legitimately invoke MSBuild

EXPECTED: Detection Signal Should Fire: **YES** (high confidence)

MSB-04: Atypical File — .csproj with Inline Task (Non-Build Context)

Objective:

Detect `.csproj` files containing inline C# tasks executed outside of Visual Studio or `dotnet build`.

Steps:

1. Copy `benign_calc.csproj` to `C:\Temp\`
2. Execute from cmd: `MSBuild.exe C:\Temp\benign_calc.csproj`
3. Record command line and file hash

Expected Telemetry:

- MSBuild executing `.csproj` from `C:\Temp\` (not a VS solution directory)
- Project file contains `<UsingTask>` with `CodeTaskFactory`
- No corresponding `.sln` file in directory

EXPECTED: Detection Signal Should Fire: **YES**

PCSIRT NOTE: The key differentiator from legitimate use is the file location and absence of a solution context. Consider detection logic that correlates file path patterns.

MSB-05: Atypical File — .xml Extension

Objective:

Detect MSBuild processing XML files containing project definitions (commonly used to evade extension-based filtering).

Steps:

1. Copy payload template 4B as `C:\Temp\benign_whoami.xml`
2. Execute: `MSBuild.exe C:\Temp\benign_whoami.xml`
3. Verify `C:\Temp\msbuild_test_output.txt` was created
4. Record command line

Expected Telemetry:

- MSBuild command line references a `.xml` file (not .csproj, .vbproj, .sln)
- File contains MSBuild project schema

EXPECTED: Detection Signal Should Fire: **YES**

MSB-06: Atypical File — Response File (.rsp)

Objective:

Detect MSBuild using response files, which can obscure the actual project being built from command-line logging.

Steps:

1. Save payload template 4C as `C:\Temp\test.rsp`
2. Ensure `benign_calc.csproj` is at `C:\Temp\`
3. Execute: `MSBuild.exe @C:\Temp\test.rsp`
4. Confirm calc launches; record command line

Expected Telemetry:

- Command line contains `@` prefix indicating response file usage
- The actual project file name may not appear in the process command line
- `/verbosity:quiet` and `/nologo` flags present

EXPECTED: Detection Signal Should Fire: **YES**

PCSIRT NOTE: This is a critical gap to test. If your signal only inspects the command line for project file names, .rsp usage will evade it. Consider alerting on the @ prefix itself when combined with other indicators.

MSB-07: Atypical File — .proj Extension with UsingTask

Objective:

Detect `.proj` files, which are valid MSBuild files but rarely used in standard development workflows.

Steps:

1. Copy `benign_calc.csproj` content to `C:\Temp\payload.proj`
2. Execute: `MSBuild.exe C:\Temp\payload.proj`
3. Confirm calc launches

Expected Telemetry:

- `.proj` extension in MSBuild command line
- File contains `CodeTaskFactory` / inline code

EXPECTED: Detection Signal Should Fire: **YES**

MSB-08: Execution from User/Temp Directories

Objective:

Detect MSBuild processing project files from directories that are not typical build paths.

Steps:

1. Copy `benign_calc.csproj` to each: Downloads, AppData\Local\Temp, C:\Users\Public, C:\ProgramData

2. Execute MSBuild against each path

3. Record each execution's command line and parent process

Expected Telemetry:

- Project file path in user-writable directories (Downloads, Temp, AppData, Public, ProgramData)
- Legitimate builds almost always reference paths under source repos or VS solution directories

EXPECTED: Detection Signal Should Fire: **YES**

MSB-09: Suspicious Flags & Verbosity Suppression

Objective:

Detect command-line patterns that suppress output or use unusual flag combinations.

Steps:

1. Execute with stealth flags: `/nologo /verbosity:quiet /noconsolelogger`

2. Execute with property overrides: `/p:Configuration=Release /p:Platform=x64 /nologo /v:q`

3. Execute with explicit target: `/t:TestTarget`

4. Record all command lines

Expected Telemetry:

- `/noconsolelogger` — suppresses all console output (very rare in legitimate use)
- `/verbosity:quiet` or `/v:q` combined with non-standard file paths
- Explicit `/t:` targeting unusual target names (not Build, Clean, Rebuild, Publish)

EXPECTED: Detection Signal Should Fire: **YES**

PCSIRT NOTE: Especially watch for /noconsolelogger combinations — this flag is extremely rare in legitimate developer workflows.

MSB-10: MSBuild Without Visual Studio / SDK Context

Objective:

Detect direct invocation of the .NET Framework MSBuild binary without any developer tooling context.

Steps:

1. Open a plain `cmd.exe` (not Developer Command Prompt)
2. Invoke MSBuild by full path with no VS environment variables set
3. Compare: Open a Visual Studio Developer Command Prompt and run a legitimate build
4. Document environment differences (VSINSTALLDIR, MSBuildSDKsPath, etc.)

Expected Telemetry:

- MSBuild invoked without `VSINSTALLDIR` or related environment variables
- No prior `devenv.exe`, `dotnet.exe`, or `nuget.exe` process in the session

EXPECTED: Detection Signal Should Fire: **YES** (lower confidence — use as signal enrichment, not standalone)

MSB-11: MSBuild Spawning Suspicious Child Processes

Objective:

Detect MSBuild spawning processes that are inconsistent with compilation (cmd, PowerShell, net, whoami, etc.).

Steps:

1. Use `child_proc_test.csproj` from the test kit (inline task spawns cmd.exe → whoami)
2. Execute: `MSBuild.exe C:\Temp\child_proc_test.csproj`
3. Verify `C:\Temp\msbuild_child_test.txt` contains the username
4. Record the full process tree

Expected Telemetry:

- Process tree: `cmd.exe` → `MSBuild.exe` → `cmd.exe` → `whoami.exe`
- Legitimate MSBuild children: `csc.exe`, `vbc.exe`, `al.exe`, `ResGen.exe`
- Suspicious children: `cmd.exe`, `powershell.exe`, `whoami.exe`, `net.exe`, `certutil.exe`, `rundll32.exe`

EXPECTED: Detection Signal Should Fire: **YES** (high confidence)

MSB-12: Logger DLL Sideloading via /logger: Switch

Objective:

Detect MSBuild loading a custom logger DLL that executes arbitrary code at build time. The `/logger:` parameter instructs MSBuild to load a .NET assembly implementing `ILogger`, which runs `Initialize()` the moment MSBuild starts — no inline task or project file payload required.

BACKGROUND: This technique is distinct from inline task abuse because the payload lives entirely in a compiled DLL. An attacker only needs a trivial .csproj (or even an empty one that fails to build) — the logger DLL executes during MSBuild initialization before any targets run.

Steps:

1. Compile the benign test logger DLL from `TestLogger.cs` (included in test kit) using: `csc.exe /target:library /reference:Microsoft.Build.Framework.dll /out:TestLogger.dll TestLogger.cs`
2. Use `empty_build.csproj` — a minimal clean project file (no malicious content)
3. Execute standard /logger: invocation: `MSBuild.exe empty_build.csproj /logger:TestLogger,C:\Temp\TestLogger.dll`
4. Execute abbreviated /l: flag: `MSBuild.exe empty_build.csproj /l:TestLogger,C:\Temp\TestLogger.dll`
5. Execute with verbosity parameter: `MSBuild.exe empty_build.csproj /logger:TestLogger,C:\Temp\TestLogger.dll;Verbosity=quiet`
6. Execute with multiple loggers (malicious + legitimate FileLogger chained together)
7. Verify `C:\Temp\logger_dll_test_output.txt` is created after each variant
8. Record all command lines and process trees for each variant

Expected Telemetry:

- MSBuild command line contains `/logger:` or `/l:` flag referencing a DLL path
- The DLL path is in a non-standard location (Temp, Downloads, AppData, ProgramData)
- The DLL is **not** a known/signed Microsoft logger
- Child process: `cmd.exe` → `whoami.exe` spawned by MSBuild
- Sysmon Event ID 7 (Image Loaded): MSBuild loading an unsigned DLL from a user-writable directory
- File creation: `TestLogger.dll` created shortly before MSBuild execution (staging indicator)

Key Detection Indicators for /logger: Abuse:

- `/logger:` or `/l:` with a DLL path outside of Program Files, VS install directories, or NuGet package caches
- Logger DLL is unsigned or has no Authenticode signature
- Logger DLL was created/modified within minutes of MSBuild execution
- Logger DLL loaded from the same directory as the project file (common in staged attacks)
- MSBuild process loads a DLL not in the Microsoft known-logger list

EXPECTED: Detection Signal Should Fire: **YES** (high confidence)

PCSIRT NOTE: This technique is particularly dangerous because the project file itself can be completely clean — all malicious logic is in the DLL. If your signal only inspects project file content for CodeTaskFactory/UsingTask, this will bypass it entirely. Detection must also inspect the command line for /logger: and /l: flags and correlate the referenced DLL against known-good logger paths. Additionally, consider monitoring Sysmon Event ID 7 (Image Load) for DLLs loaded by MSBuild from non-standard directories.

MSB-13: Negative Test — Legitimate Developer Build

Objective:

Confirm the detection signal does **NOT** fire on standard developer build activity.

Steps:

1. Open Visual Studio Developer Command Prompt

2. Navigate to a legitimate .NET project directory
3. Execute standard builds: `msbuild MySolution.sln /t:Build /p:Configuration=Debug`
4. Run `dotnet build` from a standard repo checkout
5. Execute NuGet restore followed by build (common CI/CD pattern)
6. Record all telemetry

Expected Telemetry:

- Parent process: `devenv.exe`, `dotnet.exe`, or Developer Command Prompt with VS environment
- Project files in source directories
- Standard targets: Build, Clean, Rebuild, Restore, Publish
- Child processes: `csc.exe`, `vbc.exe`, etc.

EXPECTED: Detection Signal Should Fire: **NO**

6. Results Tracking

Test ID	Executed	Timestamp	Alert Fired	Alert ID / Rule	Latency	Notes
MSB-01						
MSB-02						
MSB-03						
MSB-04						
MSB-05						
MSB-06						
MSB-07						
MSB-08						
MSB-09						
MSB-10						
MSB-11						
MSB-12						
MSB-13						

7. Detection Signal Recommendations

Based on these tests, the PCSIRT detection signal should ideally cover the following indicators, ranked by confidence:

High Confidence (alert)

- MSBuild spawning suspicious child processes (cmd, PowerShell, whoami, net, certutil, etc.)
- MSBuild parent is `wscript.exe`, `cscript.exe`, `mshta.exe`, `winword.exe`, or `excel.exe`
- MSBuild executing files from Temp, Downloads, AppData, ProgramData, or Public directories
- Command line contains `@` (response file) with project files in non-development paths
- Command line contains `/logger:` or `/l:` referencing a DLL in user-writable directories
- MSBuild loading unsigned DLLs from non-standard paths (Sysmon Event ID 7)

Medium Confidence (alert with enrichment)

- MSBuild executing `.xml` or `.proj` files
- Command line includes `/noconsolelogger`
- MSBuild invoked from plain `cmd.exe` or `powershell.exe` without VS environment context
- `.csproj` containing `CodeTaskFactory` or `RoslynCodeTaskFactory` in file content

Low Confidence (enrich / correlate)

- `/verbosity:quiet` or `/v:q` outside CI/CD context

- MSBuild invoked by a user account that has never previously run MSBuild
- Project file created within the last 60 seconds before MSBuild execution (staging indicator)

8. Cleanup Procedures

After all tests are complete, run `cleanup.ps1` from the test kit, or manually execute the following:

```
del C:\Temp\benign_calc.csproj
del C:\Temp\benign_whoami.xml
del C:\Temp\test.rsp
del C:\Temp\payload.proj
del C:\Temp\child_proc_test.csproj
del C:\Temp\msbuild_test_output.txt
del C:\Temp\msbuild_child_test.txt
del C:\Temp\TestLogger.cs
del C:\Temp\TestLogger.dll
del C:\Temp\empty_build.csproj
del C:\Temp\logger_dll_test_output.txt
del C:\Temp\build.log
del C:\Users\<testuser>\Downloads\build.csproj
del C:\Users\<testuser>\AppData\Local\Temp\build.csproj
del C:\Users\Public\build.csproj
del C:\ProgramData\build.csproj
del C:\Temp\launch_msbuild.vbs
schtasks /delete /tn "MSBuildTest" /f 2>nul
```

Confirm all test artifacts are removed and no scheduled tasks persist.

9. Sign-Off

Role	Name	Date	Signature
Red Team Lead			
PCSIRT Lead			
SOC Manager			
Change Manager			