.NET-Based Malware Techniques

Assembly Loading and Reflection

Malware can load .NET assemblies into memory using `[System.Reflection.Assembly]::Load`, `[System.Reflection.Assembly]::LoadFrom`, `Assembly.Load`, and `[System.AppDomain]::CurrentDomain.Load`. These allow attackers to execute code without writing files to disk. Reflective execution can be further invoked via `System.Reflection.MethodInfo::Invoke`.

Runtime Compilation and Dynamic Code Generation

The `Add-Type` cmdlet, `System.CodeDom.Compiler`, `System.Reflection.Emit.AssemblyBuilder`, `System.Reflection.Emit.ILGenerator`, `System.Delegate`, and `DynamicMethod` can be used to compile and execute C# code at runtime, enabling Just-In-Time (JIT) malware and dynamic payload generation.

PowerShell Script Execution

`System.Management.Automation.ScriptBlock::Create` and `Invoke-Expression` allow attackers to construct and run PowerShell code entirely in memory, often used for fileless malware or payload stagers.

Native Interop and Shellcode Execution

Attackers can bridge to native APIs using `[System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer`, enabling shellcode execution via `VirtualAlloc`, `VirtualProtect`, `CreateThread`, and `QueueUserAPC`, typically invoked through P/Invoke mechanisms in .NET.

Living-Off-the-Land (LotL) Techniques Using .NET

Network and Payload Retrieval

Malware commonly uses `System.Net.WebClient::DownloadString`, `DownloadData`, `DownloadFile`, and `System.Net.Http.HttpClient` to download code or second-stage payloads over HTTP/S.

Obfuscation and Decoding

Payloads are often encoded and decoded in memory using `System.Convert::FromBase64String`, `System.Text.Encoding::UTF8.GetString`, and cryptographic APIs like those in `System.Security.Cryptography` (e.g., AES, RSA) to avoid detection.

In-Memory Execution

Assemblies can be read and executed in memory using `System.IO.File::ReadAllBytes`, `System.IO.MemoryStream`, and `Assembly.Load(byte[])`, allowing malware to remain fileless and stealthy.

Threading and Process Control

Execution flow is managed using `System.Threading.Thread`, `ThreadStart`, `ParameterizedThreadStart`, and `System.Diagnostics.Process::Start`, often to launch new threads or spawn system processes like `cmd.exe` or `powershell.exe`.

Trusted Binary Abuse (LOLBins)

To evade detection, malware may use legitimate system tools such as `regsvr32.exe`, `rundll32.exe`, `mshta.exe`, `InstallUtil.exe`, `cmd.exe`, and `powershell.exe` to execute scripts, DLLs, or downloaded code under the guise of trusted binaries.