Proof-of-Concept Report:-

Tool Name:

Cmder & **CodeTrack**

Description:

- **Cmder** is a portable terminal emulator for Windows that enhances command-line usage by integrating with ConEmu and offering Unix-like commands and Git support.
- **CodeTrack** is a .NET performance profiler that helps developers and analysts visualize method calls, track performance bottlenecks, and identify inefficient code execution paths in real-time.

What Is This Tool About?

- **Cmder** improves productivity by offering a more functional and visually enhanced CLI environment, ideal for system admins, developers, and analysts.
- **CodeTrack** provides detailed profiling of .NET applications, useful for both performance optimization and understanding runtime behavior during reverse engineering or forensic analysis.

Key Characteristics / Features:

Cmder:

1. Portable and requires no installation
2. Git integration and UNIX command emulation (ls, grep, etc.)
3. Tabbed interface for multitasking
4. Custom aliasing and scripting support
5. Integration with PowerShell, CMD, Bash, and WSL
6. Auto-completion and syntax highlighting
7. Environment variable and path management

8. Supports SSH sessions and SCP

CodeTrack:

9. Visualizes call graphs of .NET methods
10. Provides CPU time breakdowns for profiling
11. Real-time performance tracing
12. Captures stack traces and memory allocations
13. Zoomable timeline of method calls
14. Differentiates user code from framework calls
15. Open-source and actively maintained

Types / Modules Available:

Cmder:

- Terminal Emulator
- Bash/PowerShell Integrator
- Git Shell Interface

CodeTrack:

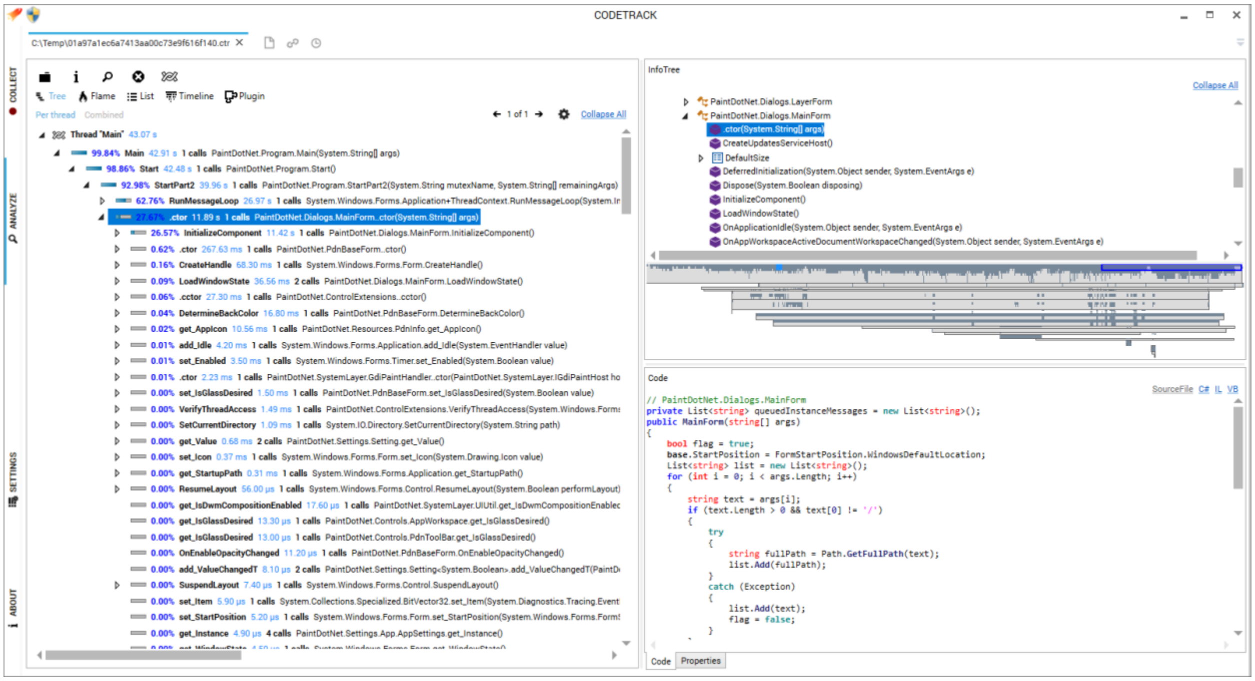
- Call Tree Visualizer
- Method Timing Analyzer
- Stack Trace Navigator
- Performance Profiler
- Real-Time Monitor

How Will This Tool Help?

- **Cmder** improves workflow and scripting capability in command-line heavy environments.
- **CodeTrack** helps identify inefficient .NET code execution and visualize program flow for debugging or forensic understanding.
- Both tools are helpful during static and dynamic analysis, especially for red/blue teams, reverse engineers, or dev teams under IR.

Proof of Concept (PoC) Images:

Official CodeTrack homepage: (https://www.getcodetrack.com/)



```
Thread "Main" 43.07 s
  99.84% Main 42.91 s 1 calls PaintDotNet.Program.Main(System.String[] args)
    98.86% Start 42.48 s 1 calls PaintDotNet.Program.Start()
      92.98% StartPart2 39.96 s 1 calls PaintDotNet.Program.StartPart2(System.String mutexName, System.String[] remainingArgs)
        62.76% RunMessageLoop 26.97 s 1 calls System.Windows.Forms.Application+ThreadContext.RunMessageLoop(System.In
        27.67% .ctor 11.89 s 1 calls PaintDotNet.Dialogs.MainForm..ctor(System.String[] args)
          26.57% InitializeComponent 11.42 s 1 calls PaintDotNet.Dialogs.MainForm.InitializeComponent()
          0.62% .ctor 267.63 ms 1 calls PaintDotNet.PdnBaseForm..ctor()
          0.16% CreateHandle 68.30 ms 1 calls System.Windows.Forms.Form.CreateHandle()
          0.09% LoadWindowState 36.56 ms 2 calls PaintDotNet.Dialogs.MainForm.LoadWindowState()
          0.06% .cctor 27.30 ms 1 calls PaintDotNet.ControlExtensions..cctor()
          0.04% DetermineBackColor 16.80 ms 1 calls PaintDotNet.PdnBaseForm.DetermineBackColor()
          0.02% get_AppIcon 10.56 ms 1 calls PaintDotNet.Resources.PdnInfo.get_AppIcon()
          0.01% add_Idle 4.20 ms 1 calls System.Windows.Forms.Application.add_Idle(System.EventHandler value)
          0.01% set_Enabled 3.50 ms 1 calls System.Windows.Forms.Timer.set_Enabled(System.Boolean value)
          0.01% .ctor 2.23 ms 1 calls PaintDotNet.SystemLayer.GdiPaintHandler..ctor(PaintDotNet.SystemLayer.IGdiPaintHost ho
          0.00% set_IsGlassDesired 1.50 ms 1 calls PaintDotNet.PdnBaseForm.set_IsGlassDesired(System.Boolean value)
          0.00% VerifyThreadAccess 1.49 ms 1 calls PaintDotNet.ControlExtensions.VerifyThreadAccess(System.Windows.Forms
          0.00% SetCurrentDirectory 1.09 ms 1 calls System.IO.Directory.SetCurrentDirectory(System.String path)
          0.00% get_Value 0.68 ms 2 calls PaintDotNet.Settings.Setting.get_Value()
          0.00% set_Icon 0.37 ms 1 calls System.Windows.Forms.Form.set_Icon(System.Drawing.Icon value)
          0.00% get_StartupPath 0.31 ms 1 calls System.Windows.Forms.Application.get_StartupPath()
          0.00% ResumeLayout 56.00 µs 1 calls System.Windows.Forms.Control.ResumeLayout(System.Boolean performLayout)
          0.00% get_IsDwmCompositionEnabled 17.60 µs 1 calls PaintDotNet.SystemLayer.UIUtil.get_IsDwmCompositionEnablec
          0.00% get_IsGlassDesired 13.30 µs 1 calls PaintDotNet.Controls.AppWorkspace.get_IsGlassDesired()
          0.00% get_IsGlassDesired 13.00 µs 1 calls PaintDotNet.Controls.PdnToolBar.get_IsGlassDesired()
          0.00% OnEnableOpacityChanged 11.20 µs 1 calls PaintDotNet.PdnBaseForm.OnEnableOpacityChanged()
          0.00% add_ValueChangedT 8.10 µs 2 calls PaintDotNet.Settings.Setting<System.Boolean>.add_ValueChangedT(PaintD
          0.00% SuspendLayout 7.40 µs 1 calls System.Windows.Forms.Control.SuspendLayout()
          0.00% set_Item 5.90 µs 1 calls System.Collections.Specialized.BitVector32.set_Item(System.Diagnostics.Tracing.Event
          0.00% set_StartPosition 5.20 µs 1 calls System.Windows.Forms.Form.set_StartPosition(System.Windows.Forms.Form
          0.00% get_Instance 4.90 µs 4 calls PaintDotNet.Settings.App.AppSettings.get_Instance()
```

```
PaintDotNet.Dialogs.LayerForm
PaintDotNet.Dialogs.MainForm
  .ctor(System.String[] args)
  CreateUpdatesServiceHost()
  DefaultSize
  DeferredInitialization(System.Object sender, System.EventArgs e)
  Dispose(System.Boolean disposing)
  InitializeComponent()
  LoadWindowState()
  OnApplicationIdle(System.Object sender, System.EventArgs e)
  OnAppWorkspaceActiveDocumentWorkspaceChanged(System.Object sender, System.EventArgs e)
```

```csharp
// PaintDotNet.Dialogs.MainForm
private List<string> queuedInstanceMessages = new List<string>();
public MainForm(string[] args)
{
    bool flag = true;
    base.StartPosition = FormStartPosition.WindowsDefaultLocation;
    List<string> list = new List<string>();
    for (int i = 0; i < args.Length; i++)
    {
        string text = args[i];
        if (text.Length > 0 && text[0] != '/')
        {
            try
            {
                string fullPath = Path.GetFullPath(text);
                list.Add(fullPath);
            }
            catch (Exception)
            {
                list.Add(text);
                flag = false;
            }
```

15-Liner Summary:

1. Cmder improves CLI experience on Windows
2. Portable and customizable
3. Git and UNIX command support
4. Supports aliases, scripting, and SSH
5. CodeTrack visualizes .NET performance
6. Captures stack traces and CPU usage
7. Zoomable and detailed timelines
8. Differentiates between .NET and framework calls
9. Ideal for performance profiling

10. Useful in reverse engineering workflows
11. Compatible with Visual Studio outputs
12. Helps identify bottlenecks in applications
13. Cmder integrates well in DevOps pipelines
14. Both tools are open source and maintained
15. Lightweight and fast to use

⏱️ Time to Use / Best Case Scenarios:

- **Cmder:**
  - During incident response and scripting
  - When working across multiple terminals
  - Portable CLI usage on forensic jump kits
- **CodeTrack:**
  - When profiling .NET executables
  - During malware analysis of .NET droppers
  - Performance benchmarking and optimization

🕵️ When to Use During Investigation:

- Reverse engineering .NET-based malware
- Tracing method calls in obfuscated .NET apps
- CLI-based script execution and automation
- Terminal management during live forensics
- Performance profiling for sandboxed programs

🧑‍💻 Best Person to Use This Tool & Required Skills:

Best Users:

- Reverse Engineers
- Developers / .NET Analysts
- IR Engineers and Blue Teamers

Required Skills:

- Familiarity with terminal commands and scripting
- Basic understanding of .NET internals
- Debugging and profiling experience
- CLI environment configuration skills

❌ Flaws / Suggestions to Improve:

Cmder:

- Can slow down with large Git repos
- Lacks built-in plugin store for shells
- Occasional font rendering issues

CodeTrack:

- Limited to .NET applications
- May require admin access to attach
- No direct export to popular profiler formats

✅ Good About the Tool:

- **Cmder:**
  - Simple, elegant, and powerful terminal
  - Great Git and Unix support for Windows users
- **CodeTrack:**
  - Deep .NET introspection
  - Visual and insightful method tracking