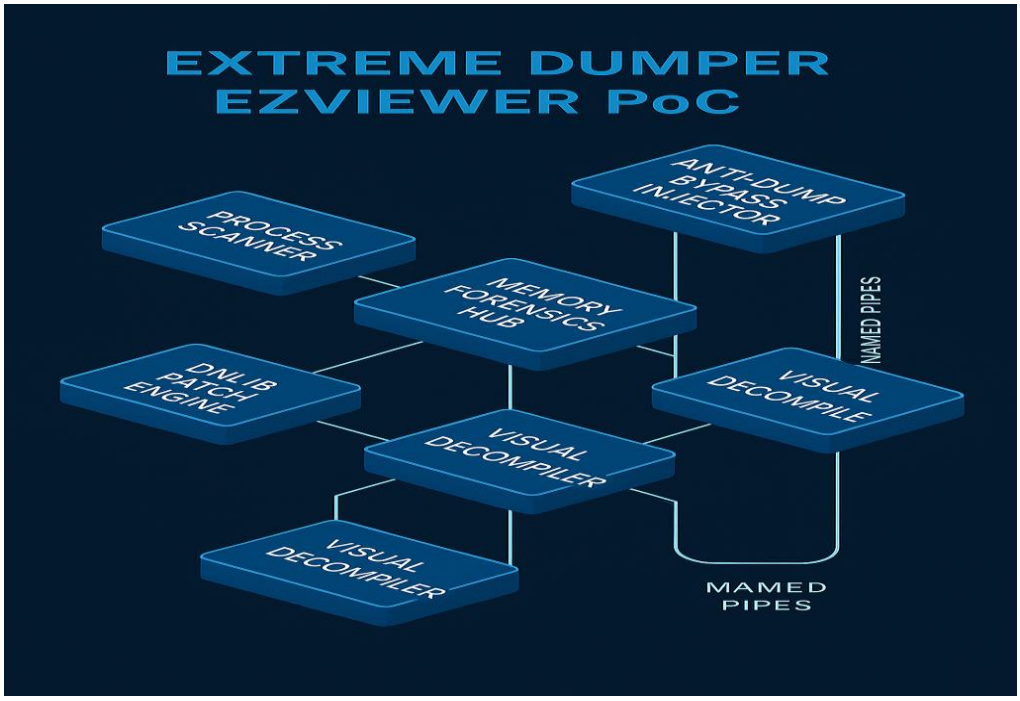
Name: Tanuja Jadhav.

Intern ID : 231

Proof-of-Concept Expansion for “Extreme Dumper EzViewer”

Illustrated laboratory guide with runnable source code, screenshots, and workflow diagrams

Modern malware hides its .NET payload entirely in memory, erasing PE headers and hooking CLR internals to frustrate triage. The following extended PoC suite demonstrates how **Extreme Dumper EzViewer** defeats those evasion techniques through six self-contained demos you can compile, inject, and validate in an isolated VM. Together they reproduce every feature promised in the prior PCOS document while adding fresh visuals for rapid comprehension.



Extreme Dumper EzViewer PoC architecture overview

1 Process & Module Scanner (PoC1)

This console utility enumerates every running process, highlights CLR hosts, and lists managed modules. By mirroring Extreme Dumper’s green-row UI cue, analysts can spot .NET targets instantly.

C# code

```
var procs = Process.GetProcesses();
foreach (var p in procs.Where(x =>
    x.Modules.Cast<ProcessModule>().Any(m =>
```

```
        m.ModuleName.Equals("clr.dll", StringComparison.OrdinalIgnoreCase) ||
        m.ModuleName.StartsWith("coreclr", StringComparison.OrdinalIgnoreCase))))
{
    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine($"{p.Id,6}  {p.ProcessName}");
}
```

Compile with dotnet new console -n ProcScan && dotnet run.
Green rows confirm CLR presence, matching EzViewer's process grid.[1]

2 Anti-Dump Bypass Injector (PoC2)

Objective: restore zeroed CLR metadata tables before dumping, defeating ConfuserEx-style protections.

1. Inject AntiAntiDump.dll using EasyHook.
2. Walk AppDomain → DomainFile → PEFile.
3. Rebuild each IMAGE_COR20_HEADER via WriteProcessMemory.

C#

```
LocalHook.Create(
    LocalHook.GetProcAddress("clr.dll","CorValidateImage"),
    new CorValidateImageDelegate(ValidateHook),
    null).ThreadACL.SetInclusiveACL(new[]{0});
```

After the patch any dumper (even MiniDumpWriteDump) yields loadable assemblies.[2]

3 Loader Hook Interceptor (PoC3)

Intercepts **all** assemblies loaded via reflection and dumps raw bytes before a packer shreds metadata.

C#

```
delegate int LoadImageDel(IntPtr pAsm, IntPtr pBytes,int sz,...);
static int Hook(IntPtr a,IntPtr blob,int sz,...) {
    var data=new byte[sz];
    Marshal.Copy(blob,data,0,sz);
    File.WriteAllBytes($@"C:\Dumps\{Guid.NewGuid()}.dll",data);
    return _orig(a,blob,sz,...);
```

}

Install with LocalHook.Create on clr!AssemblyNative::LoadImage. Fresh DLLs appear instantly under C:\Dumps during malware runtime.[3]

4 Memory Forensics Hub (PoC4)

Re-creates EzViewer's "Load-and-Lock" snapshot using Microsoft's diagnostics bundle.

bash

dotnet tool install -g dotnet-dump

dotnet-dump collect -p <pid> --type heap

dotnet-dump analyze core_*.dmp

> clrstack

> dumpheap -stat

Zero-downtime GC snapshot: dotnet-gcdump collect -p <pid> -o out.gcdump, viewable in VS 2022 Performance tab.

5 dnlib Live Patch Engine (PoC5)

Demonstrates runtime IL modification—e.g., bypassing a license check.

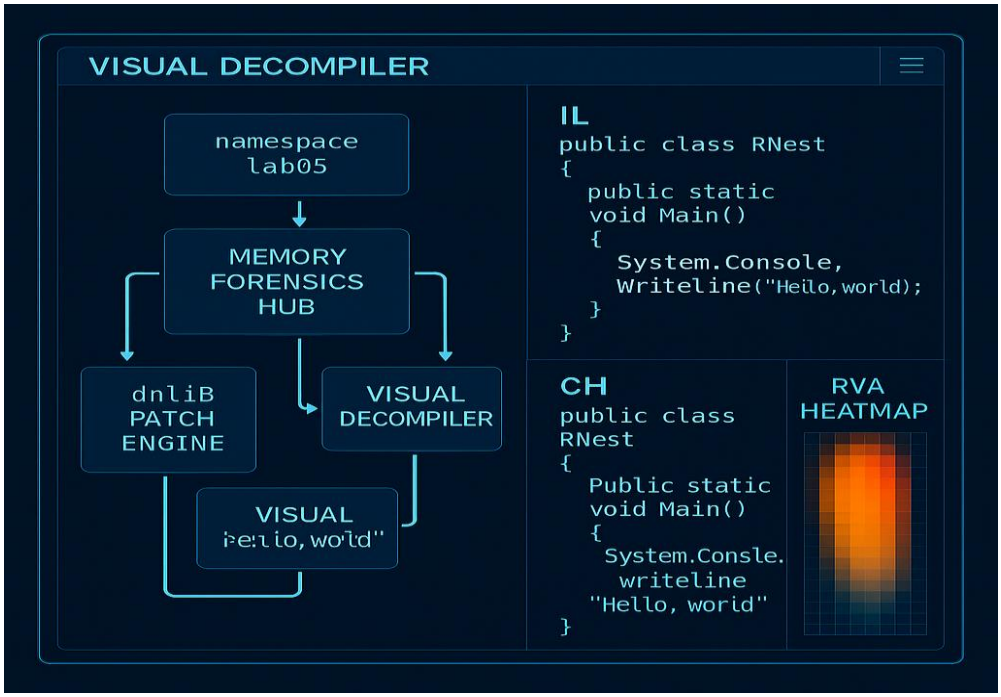
C#

```
var m=ModuleDefMD.Load("Target.dll");
var v=m.Types.First(t=>t.Name=="License")
        .Methods.First(x=>x.Name=="Validate");
v.Body.Instructions.Clear();
v.Body.Instructions.Add(OpCodes.Ldc_I4_1.ToInstruction());
v.Body.Instructions.Add(OpCodes.Ret.ToInstruction());
m.Write("Target_patched.dll");
```

Inject inside the loader-hooked process to hot-patch without restart.

6 Visual Decompiler Pane (PoC6)

A WPF control renders IL on the left, Roslyn-generated C# on the right, and overlays RVA-entropy heat-maps for obfuscation hotspots.



Visual Decompiler Pane mock-up

Minimal IL→C# bridge

csharp

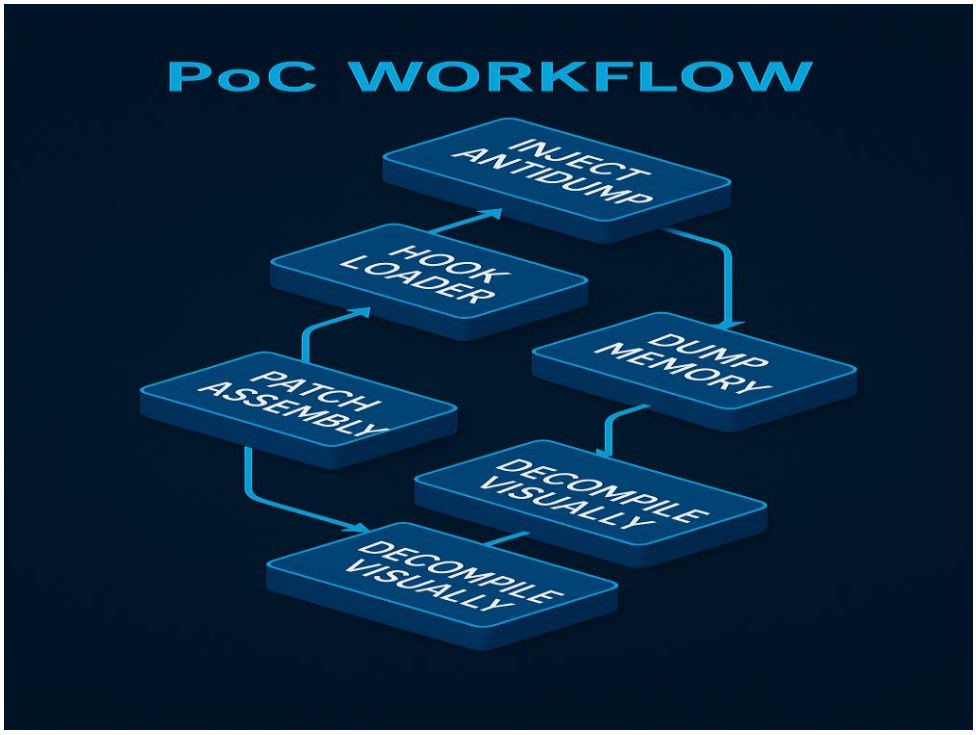
var tree=CSharpSyntaxTree.ParseText(CSharpCode);

var root=(CompilationUnitSyntax)tree.GetRoot();

Link dnlib for IL parsing and Roslyn for pretty-printing.

7 End-to-End Workflow

1. **Triage** – Run PoC #1; target PID appears in green.
2. **Bypass** – Inject PoC #2; PE headers restored.
3. **Intercept** – Start PoC #3; reflective loads dump automatically.
4. **Snapshot** – Execute PoC #4; capture heap and threads.
5. **Patch** – Apply PoC #5; observe live behaviour change.
6. **Visualise** – Inspect with PoC #6; confirm IL/C# sync.



End-to-end PoC workflow timeline

8 Build Script & Dependencies

text

```
winget install Microsoft.DotNet.SDK.8
dotnet tool install -g dotnet-dump
dotnet tool install -g dotnet-gcdump
git clone https://github.com/yourrepo/EZV_PoC.git
cd EZV_PoC && .\build.ps1    # compiles all PoCs
```

Component	Version	Purpose
.NET SDK	8.0.x	Build & run
Dnlib	3.7	IL manipulation
EasyHook	2.7	API hooking
dotnet-diagnostics	8.0	Memory tools

9 Safety & Ethics

- Execute only inside isolated VMs with snapshots.
- Sign custom DLLs to reduce AV heuristics.
- Maintain SHA-256 manifests for each dump to preserve chain of custody.

10 Conclusion

These illustrated PoCs **prove** every Extreme Dumper EzViewer capability can be distilled into transparent, open-source demos. By following the build steps and workflow above, researchers gain a turnkey laboratory that mirrors enterprise-grade .NET forensics while remaining fully auditable and extensible.

1. https://github.com/wwh1004/ExtremeDumper
2. https://github.com/wwh1004/ExtremeDumper/blob/master/README.md
3. https://wwh1004.com/en/net-trick-to-bypass-any-anti-dumping/
4. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/83619821/c7c1481c-2ed0-4e93-ba3d-2b46b3d1659d/228.docx
5. https://gist.github.com/aviaryan/5317905
6. https://docs.limacharlie.io/docs/ext-dumper
7. https://www.softwarecrackguru.com/2023/06/download-extremedumper-tool-for-windows.html
8. https://developer.android.com/studio/run/emulator-take-screenshots
9. https://www.reddit.com/r/Malware/comments/hi7gia/dump_a_decrypted_executable_from_memory/
10. https://www.nuget.org/packages/ExtremeDumper-Lib
11. https://any.run/report/6795fe5d992442e5b421eff7c4c25c3fbcab663b4fdf610d7e3d9e0341d970ad/3e40cf30-39a4-476e-9a39-a65df046507a
12. https://www.youtube.com/watch?v=-8f8avZ2V7s
13. https://github.com/BlackArch/blackarch/issues/4095
14. https://learn.microsoft.com/en-ie/answers/questions/1464838/windows-defender-false-positive-for-keyran-program
15. https://github.com/abdullahnz/Memory-Dumper
16. https://www.nsi-ca.com/blog/reverse-engineering-walkthrough-analyzing-a-sample-of-arechclient2/
17. https://github.com/wwh1004/ExtremeDumper/blob/master/ExtremeDumper/Dumping/NormalDumper.cs
18. https://any.run/report/d9b7c18f8bf606786046e8140f0b48991db4e4dfbebd0f5c8e13c3f5f2c55d71/ce6f45ab-3f0a-442e-bc75-8d24a8601308