December 03, 2024

Discovering a Deserialization Vulnerability in LINQPad

Written by James Williams

Red Team Adversarial Attack Simulation

SKIP TO MAIN CONTENT

Tal

1.

1.

Like most red teamers, I spend quite a lot of time looking for novel vulnerabilities that could be used for initial access or lateral movement. Recently, my focus has been on deserialization vulnerabilities in .NET applications, especially those that are commonly used or installed by default. This research led to me discovering a deserialization vulnerability in LINQPad, a .NET scratchpad application commonly used by developers. In this post, we will look at how this vulnerability was found.

1.1 A Needle in a Haystack

If we want to find vulnerabilities in .NET apps, we need .NET apps to analyze (shocking, I know). Windows ships with quite a few .NET DLLs and executables already installed, but there are also plenty of popular third-party tools written in .NET. When looking for novel exploits, we really want to focus our search on commonly used applications; there is little point putting in hours of work to find a vulnerability in an app with 10 users. An easy way to load up a VM with commonly used apps is to simply use it for a while, do some work on it, and install whatever tools you need as you go. I used to be a .NET developer and spent quite a bit of time working in that ecosystem, so I spun up I ran CerealKiller and started an st, identify the ones built with .N provide. In this case, we are loo

Th xt step is to triage the results, loo ser-controlled data. If we find one of d find out where the data comes from. This process is repeated unt on is not vulnerable or find a path

from “run this exe with these arguments" to "trigger a deserialization call”. Being rather lazy, I decided to triage the .exe results first, as these would, hopefully, make it easier to find the entry point. Searching the CerealKiller results for .exe highlighted 64 results with many duplicates. The results also contained a few tools we can immediately rule out, such as ysoserial.net and some proof-of-concept code I had been working on. This left a handful of results, including apps like MSBuild, msdeploy, and LINQPad.

A quick triage of msdeploy.exe revealed that the vulnerable code was not called anywhere, and a Google search returned no hits for that method. At this stage, that was enough for me to put that one to the side and move on. MSBuild is a well-known LOLBin, so again I left that one for another time. While a new vulnerability in MSBuild would be interesting, there’s a very good chance it would be flagged if we ever tried to run it. That left LINQPad.

Throwing LINQPad.exe into dotPeek and looking at the identified method confirmed a call to BinaryFormatter.Deserialize.

```
private static bool PopulateFromCache()
{
  if (!File.Exists(AutoRefManager._cachePath))
    return false;
  try
  {
    Dictionary<string, string[]> dictionary;
    using (FileStream serializationStream = File.OpenRead(AutoRefManager._cachePath))
      dictionary = (Dictionary<string, string[]>) new BinaryFormatter().Deserialize((Stream) serializationStream);
    if (dictionary == null)
      return false;
    TypeResolver.AutoRefCaseLookup = dictionary.Keys.ToLookup<string, string>((Func<string, string>) (s => s), (IEqualityCompar
    TypeResolver.AutoRefLookup = AutoRefManager._refLookup = dictionary;
    return true;
  }
  catch
  {
    return false;
  }
}
```

e Method

Th serialize on the contents.

Re file is expected to be found in the

us

```
internal class AutoRefManager
{
  private static Dictionary<string, string[]> _refLookup;
  private static string _cachePath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "LINQPad\\AutoRefCache46.1.dat");
  private static readonly string _runtimeDir = RuntimeEnvironment.GetRuntimeDirectory();
  private static readonly string ExtraRefs = "Microsoft.CodeAnalysis\r\nMicrosoft.CodeAnalysis.Desktop\r\nMicrosoft.CodeAnalysis.CSharp\r\nMicrosoft.CodeAnalysis
```

Figure 2 - Cache File Path

So far, so good. We have a deserialization call taking data from a file the user will have permissions to write to. Now all we need to do is find where that method is called from.

1.2 Are We Still Thinking in Graphs?

We could spend our time looking for uses of the vulnerable method, or we could spend even more time building a tool to make it slightly easier. I'd been toying with the idea of using Neo4j to graph the relationships between functions, showing paths to functions of interest (in this case, BinaryFormatter.Deserialize), and now seemed like as good a time as any to start work on it. Of course, when I started researching this idea, I quickly discovered that XPN had gotten there first, so this idea isn't novel, but at least we know it should work.

With a little time alone with ChatGPT, I got some working proof-of-concept code together that scans a binary, extracts all the functions it contains, and builds a cypher file containing the functions and details of where they are called.

We're using Mono.Cecil to do the decompilation, then going through each method, looking for calls to other methods. This code is still quite rough around the edges, but it produces some usable output.

```
namespace Nebuchadnezzar
{

    class Program
    {
        static void Main(string[] args)
        {
            if (args.Length &lt; 2)
            {
                Console.WriteLine("Usage: &lt;assembly path&gt; &lt;output cypher
                return;
            }

            string assemblyPath = args[0];
            string cypherOutputPath = args[1];

            // Extract the assembly name (without path) for use in the graph
            string assemblyName = Path.GetFileNameWithoutExtension(assemblyPath);

            // Create or overwrite the cypher output file
            using (StreamWriter writer = new StreamWriter(cypherOutputPath))
            {
                var assembly = AssemblyDefinition.ReadAssembly(assemblyPath);

                // Dictionary to avoid creating duplicate method nodes
                Dictionary&lt;string, bool&gt; methodNodes = new Dictionary&lt;st
```


SKIP TO MAIN CONTENT

```
                if (!method.HasBody) continue;

                // Create a unique method node for the current method inc
                string methodNodeName = $"{type.FullName}.{method.Name}";
                string uniqueMethodNodeName = $"{assemblyName}.{methodNod

                // Create a method node and link it to the Assembly node
                if (!methodNodes.ContainsKey(uniqueMethodNodeName))
                {
                    writer.WriteLine($"MERGE (m:Method {{name: \"{methodN
                    writer.WriteLine($"MERGE (a)-[:CONTAINS]-&gt;(m);");
                    methodNodes[uniqueMethodNodeName] = true;
                }

                // Iterate through method instructions to find method cal
                foreach (var instruction in method.Body.Instructions)
                {
                    if (instruction.OpCode == OpCodes.Call || instruction
                    {
                        var methodRef = (MethodReference)instruction.Oper
                        string calledMethodName = $"{methodRef.DeclaringT
                        string uniqueCalledMethodName = $"{assemblyName}.

                        // Get the parameters of the called method
                        var parameterList = new List&lt;string&gt;();
                        foreach (var parameter in methodRef.Parameters)

                                                                 (parameter.ParameterType.Nam

                                                                 string.Join(", ", parameterLi

                                                                 method node if it doesn't ex
                                                                 tainsKey(uniqueCalledMethodNa
```

```
                                writer.WriteLine($"MERGE (m:Method {{name: \"
                                writer.WriteLine($"MERGE (a)-[:CONTAINS]-&gt;
                                methodNodes[uniqueCalledMethodName] = true;
                            }

                            // Create the CALLS relationship between methods
                            writer.WriteLine($"MATCH (caller:Method {{name: \
                            writer.WriteLine($"(callee:Method {{name: \"{call
                            writer.WriteLine($"MERGE (caller)-[:CALLS {{param
                        }
                    }
                }
            }

            Console.WriteLine("Cypher script generated successfully.");
        }
    }

}
```

Running the code is as simple as calling the .exe and giving it an output file location.

```
PS E:\Dev\Nebuchadnezzar\Nebuchadnezzar\bin\Debug> .\Nebuchadnezzar.exe 'C:\Program Files (x86)\LINQPad5\LINQPad.exe' e:\test.cypher
Cypher script generated successfully.
PS
```

Th eo4j database.

File Edit View

```
MERGE (a:Assembly {name: "LINQPad"});
MERGE (m:Method {name: "<>f__AnonymousType0`2.get_da", assembly: "LINQPad"});
MERGE (a)-[:CONTAINS]->(m);
MERGE (m:Method {name: "<>f__AnonymousType0`2.get_ai", assembly: "LINQPad"});
MERGE (a)-[:CONTAINS]->(m);
MERGE (m:Method {name: "<>f__AnonymousType0`2..ctor", assembly: "LINQPad"});
MERGE (a)-[:CONTAINS]->(m);
MERGE (m:Method {name: "System.Object..ctor", assembly: "LINQPad"});
MERGE (a)-[:CONTAINS]->(m);
MATCH (caller:Method {name: "<>f__AnonymousType0`2..ctor", assembly: "LINQPad"}),
(callee:Method {name: "System.Object..ctor", assembly: "LINQPad"})
MERGE (caller)-[:CALLS {parameters: ""}]->(callee);
MERGE (m:Method {name: "<>f__AnonymousType0`2.Equals", assembly: "LINQPad"});
MERGE (a)-[:CONTAINS]->(m);
MERGE (m:Method {name: "System.Collections.Generic.EqualityComparer`1<<da>j__TPar>.get_Default", assembly: "LINQPad"});
MERGE (a)-[:CONTAINS]->(m);
MATCH (caller:Method {name: "<>f__AnonymousType0`2.Equals", assembly: "LINQPad"}),
(callee:Method {name: "System.Collections.Generic.EqualityComparer`1<<da>j__TPar>.get_Default", assembly: "LINQPad"})
MERGE (caller)-[:CALLS {parameters: ""}]->(callee);
```

Figure 4 - Cypher File Contents

Now all we need to do is import this data into Neo4j. Doing this with Docker is a little complicated, but easy when you know how.

First, we spin up a Neo4j container using the following command:

```
docker run --interactive --tty --name neo \
    --publish=7474:7474 --publish=7687:7687 \
    --volume=/Users/james/Desktop:/foo \
    neo4j
```

Th l mounts my Desktop folder to /fo : should be accessible to Neo4j.

W 7687. You'll need to change the pa esh Neo4j instance.

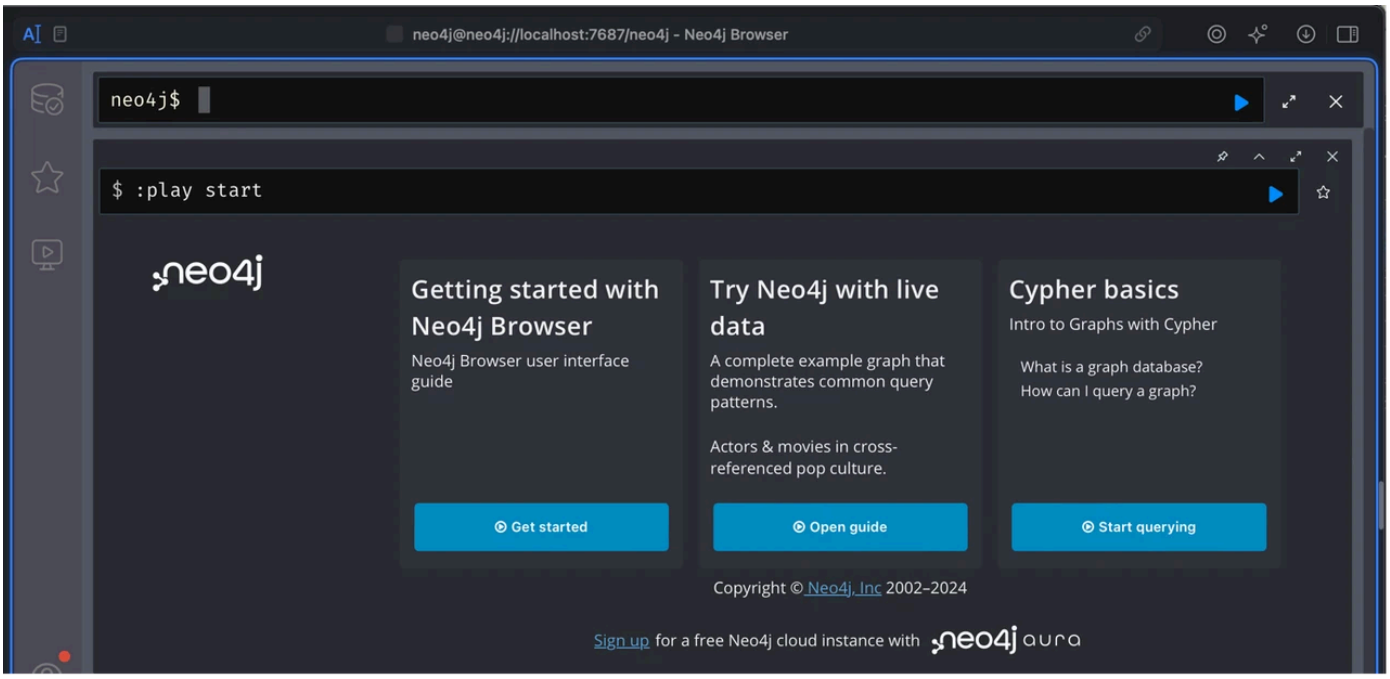


Figure 5 - Neo4j Web UI

Now, we need to import that data. To do this, we must use the terminal inside the container. We can run docker exec -it neo sh to get a shell on our container.

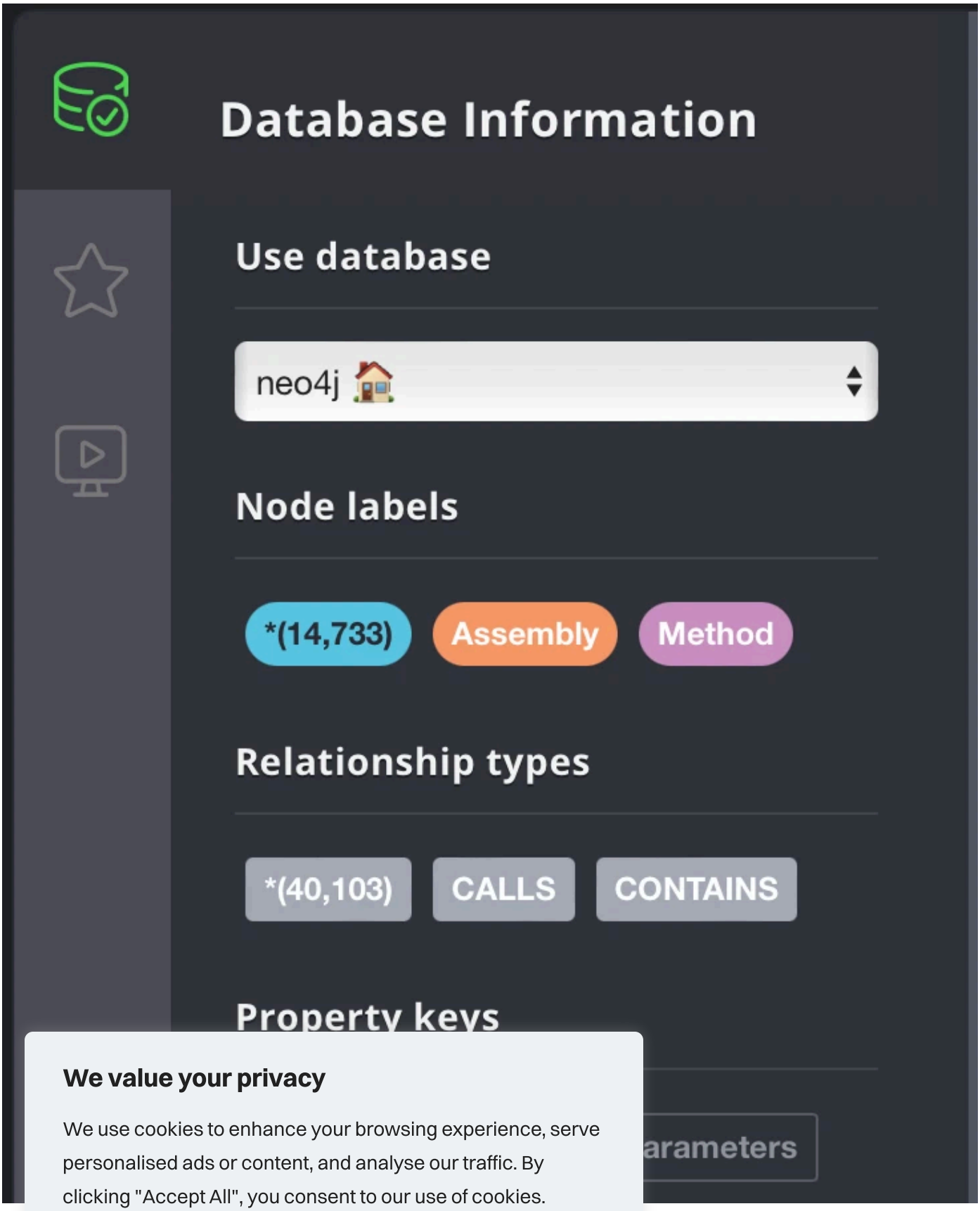
To import the data, we are going to use cypher-shell. We cat the file and pipe it into cypher-shell, providing our Neo4j username and password as arguments.

```
# pwd
/foo
# cat test.cypher | cypher-shell -u neo4j -p password
```


·Shell

Th s.

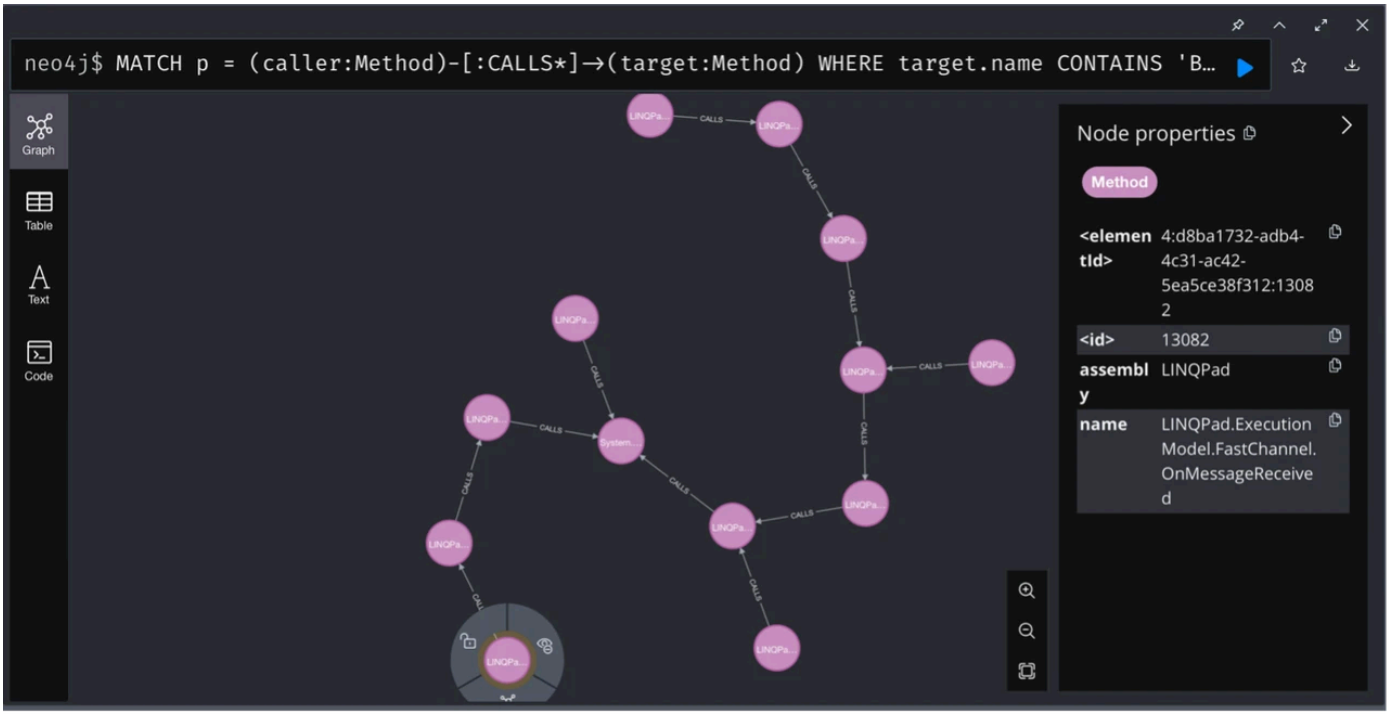
W data.



Now, we just need some queries. We want to find all paths to a BinaryFormatter.Deserialize call, so we can use the following cypher query:

```
MATCH p = (caller:Method)-[:CALLS*]->(target:Method)
WHERE target.name CONTAINS 'BinaryFormatter.Deserialize'
RETURN caller, target, p;
```

Digging into the results of this query, we see a couple of places where BinaryFormatter is called.



er

Fo ethod is called.

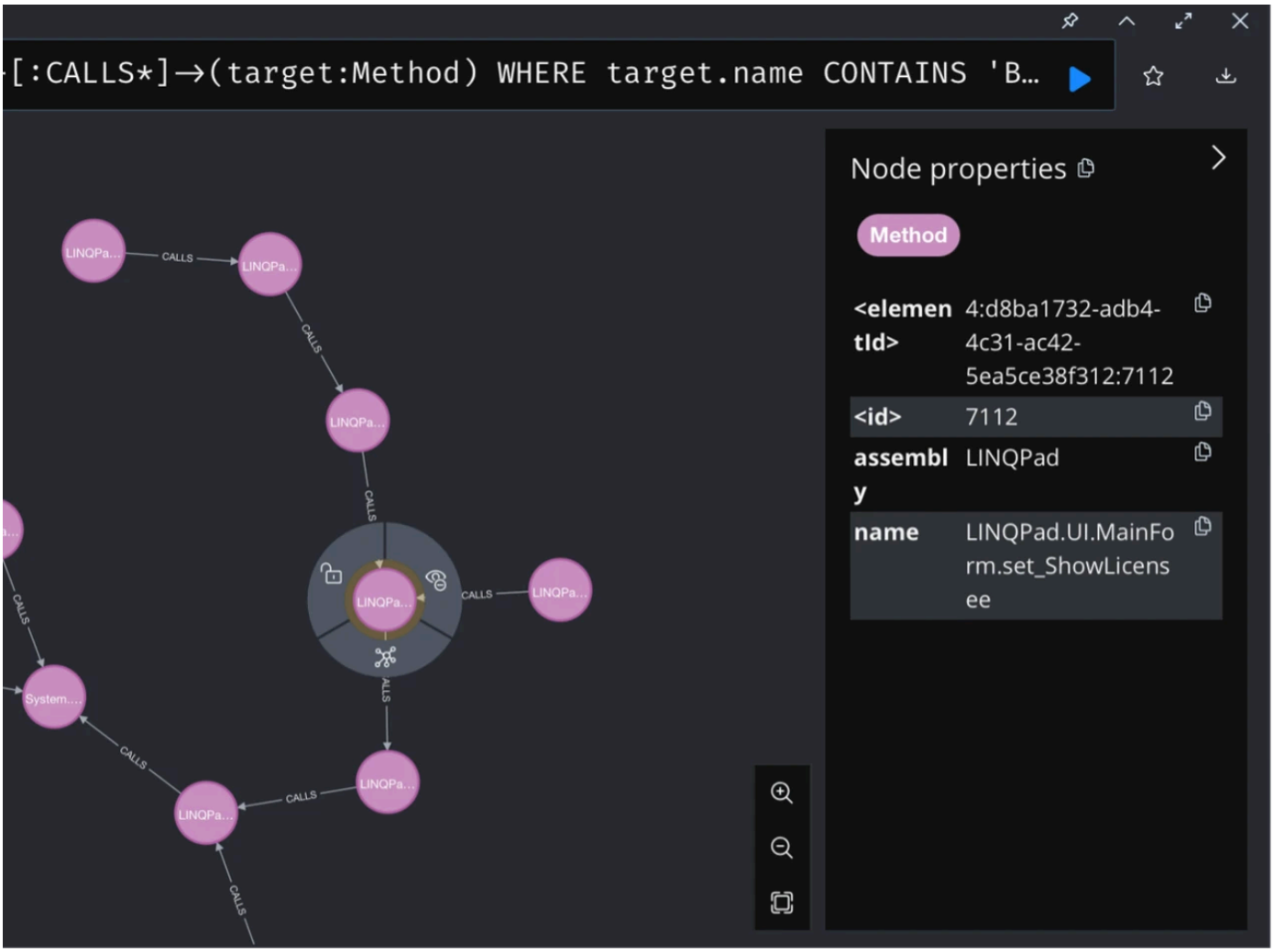
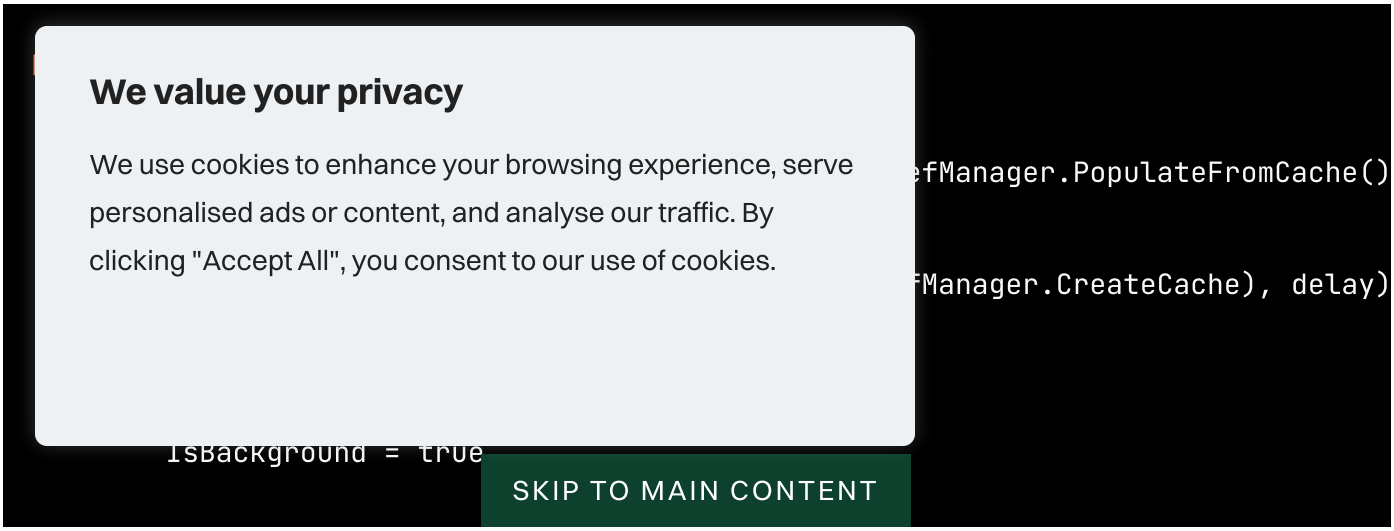


Figure 9 - Finding Paths to our Vulnerable Method

Following the graph, we can see that AutoRefManager.PopulateFromCache is called by AutoRefManager.Initialize.



```
        Priority = ThreadPriority.Lowest
      }.Start();
    }
The Initialize method is called by UI.Mainform.set_ShowLicensee.

internal bool ShowLicensee
    {
      get => this._showLicensee;
      private set
      {
        if (this._showLicensee == value)
          return;
        this._showLicensee = value;
        foreach (QueryControl queryControl in this.GetQueryControlsWithCache())
        {
          queryControl.UpdateAutocompletionMsg();
          queryControl.UpdateOutliningEnabled();
        }
        if (value)
          AutoRefManager.Initialize(1000);
        if (!value)
          return;
        this.CurrentQueryControl?.WarmupServices();
      }
    }
```

Th oreActivationMessage, within
Ma

Th ely obvious where this code was
ca sed copy of LINQPad was used.

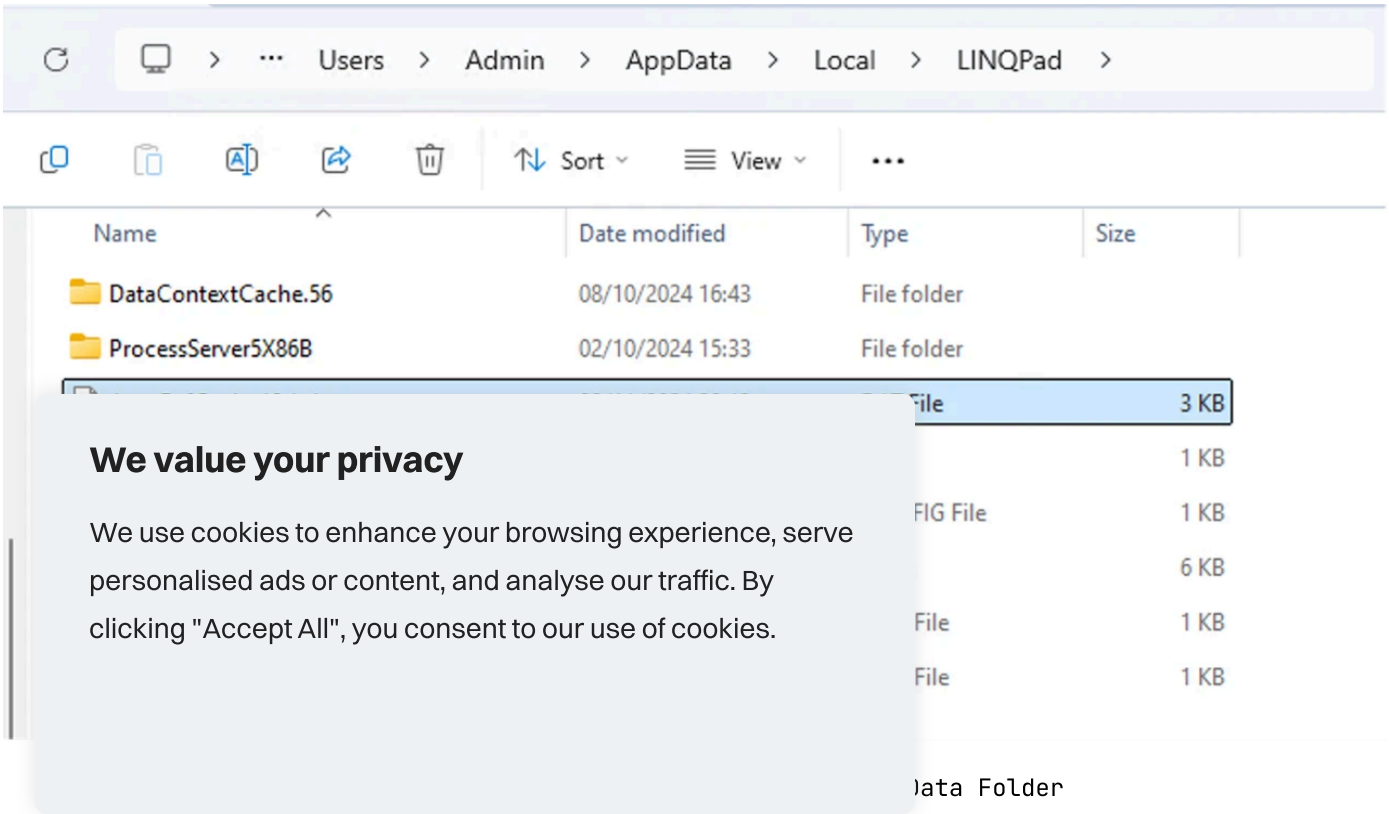
1.3 Building a Proof of Concept

Let's recap. We have a potential vulnerability in an app that, according to its website, has more than 5 million downloads and 50,000 customers who use a paid edition, including 30 fortune 100 companies and four (4) of the world's largest banks. Oh, and the biggest corporate user is Microsoft. But we don't have a proof of concept yet because we're pretty sure the vulnerable code is only called when a licensed version is in use. Of course, I bought a license.

With my freshly purchased license in hand, it was time to build a proof-of-concept payload. For this, we turn once again to ysoserial.net.

```
ysoserial.exe -f binaryformatter -g typeconfusedelegate -c calc.exe -o raw > e:\A
```

This payload was then copied to %localappdata%l\LINQPad\.



LINQPad was then launched, which caused the payload to trigger and calc.exe to be launched.

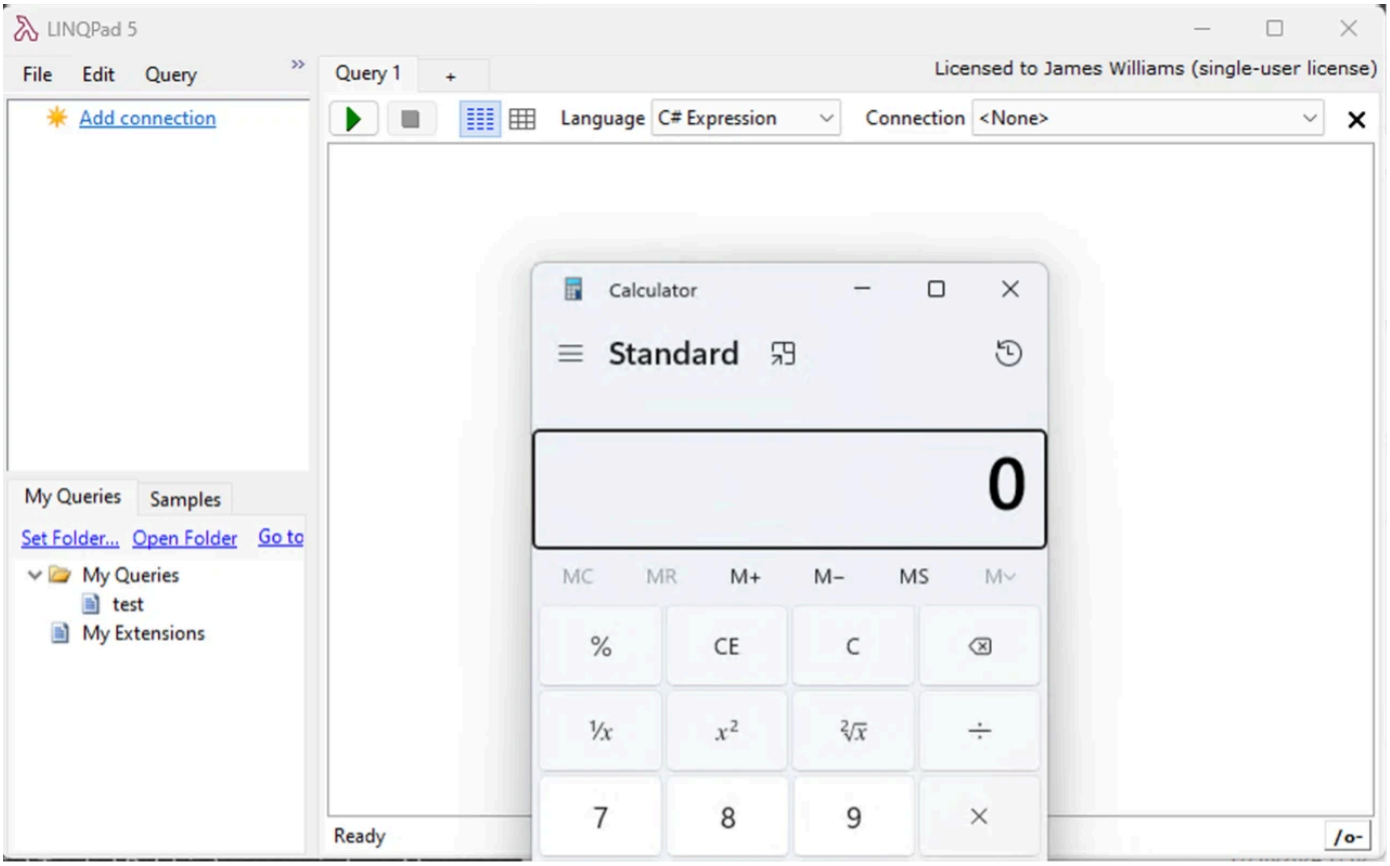


Figure 11 - calc.exe Loaded by LINQPad

The AutoRefCach46.1.dat file is then overwritten, which does pose a small problem, but nothing we couldn't work around if we really wanted to.

So now we have a working exploit, but what to do with it? I had two choices: hoard the vuln and hope it was useful on a Red Team one day or disclose it to the vendor. In the end, I chose to dis the vulnerability isn't all that us the number of times we'd likely us ight thing to do.

Co uding a fix ready in just five days fro

1.4 Wrapping Up

In this post, we've seen some of the methodology I use when looking for novel deserialization vulnerabilities. We've used Neo4j to make the process of finding paths to vulnerable functions easier, which ultimately led to us finding and exploiting a novel deserialization vulnerability in LINQPad. There is plenty of scope for further work here, especially around the use of Neo4j. Building a graph containing data from all DLLs and executables on a default Windows installation would be a very interesting exercise.

This was tested against LINQPad v5.48.00 (Pro Edition). Unlicensed versions were not vulnerable. This issue was patched in LINQPad 5.52.01, which is now RTM and can be tracked under CVE-2024-53326.

Terms Of Service

Privacy Policy