Microsoft IIS servers hacked by Blue Mockingbird to mine Monero

bleepingcomputer.com/news/security/microsoft-iis-servers-hacked-by-blue-mockingbird-to-mine-monero/

Ax Sharma

By
Ax Sharma

- May 28, 2020
- 01:13 PM
- 0

This month news broke about a hacker group, namely Blue Mockingbird, exploiting a critical vulnerability in Microsoft IIS servers to plant Monero (XMR) cryptocurrency miners on compromised machines.

According to the security firm Red Canary, the estimated number of infections is thought to have surpassed 1,000.

Exploiting a Telerik vulnerability

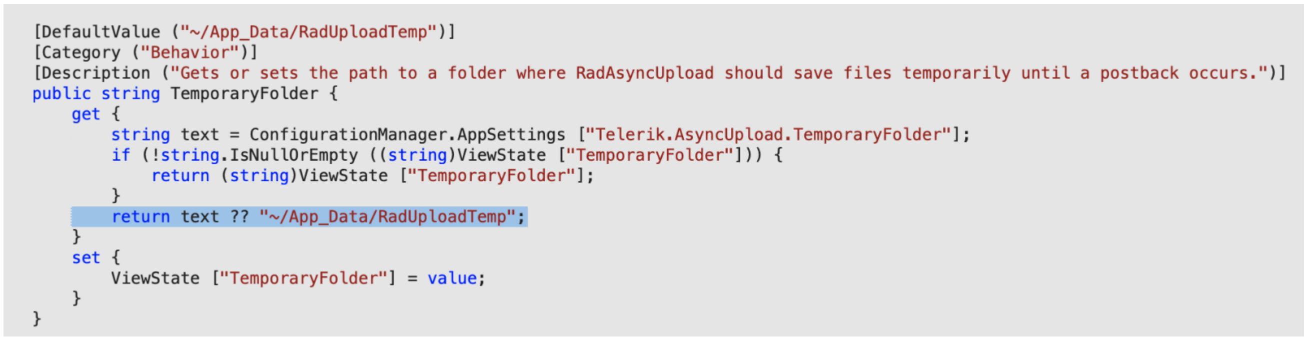
The CVE-2019-18935 vulnerability, with its critical 9.8 severity score, is an untrusted deserialization vulnerability within the proprietary Progress Telerik UI (for ASP.NET AJAX) library which is often bundled with .NET components, including some open-source ones.

While originally published in December 2019, the flaw continues to be exploited even today despite patches and fixes having been made available. The reason for this could be that patching production IIS systems running vulnerable Telerik UI DLLs may be a challenge in practice.

As previously highlighted by BleepingComputer, multiple government organisations including the NSA and Australian authorities have issued warnings against this and related Progress Telerik UI vulnerabilities, because of their prominent exploitation by hackers to gain backdoor and web shell access.

It is interesting to note the vulnerability isn't all that novel either. Versions of Telerik Web UI were previously reported to have path traversal, remote code execution, and weak encryption flaws such as CVE-2014-2217, CVE-2017-11317, and CVE-2017-11357, with exploits and PoCs readily available.

All of these vulnerabilities and the current CVE-2019-18935 somewhat concern the same functionality, `RadAsyncUpload`: the class responsible for asynchronously (via AJAX) uploading files to a temporary folder and keeping them there up until a postback event occurs.



```
[DefaultValue ("~/App_Data/RadUploadTemp")]
[Category ("Behavior")]
[Description ("Gets or sets the path to a folder where RadAsyncUpload should save files temporarily until a postback occurs.")]
public string TemporaryFolder {
    get {
        string text = ConfigurationManager.AppSettings ["Telerik.AsyncUpload.TemporaryFolder"];
        if (!string.IsNullOrEmpty ((string)ViewState ["TemporaryFolder"])) {
            return (string)ViewState ["TemporaryFolder"];
        }
        return text ?? "~/App_Data/RadUploadTemp";
    }
    set {
        ViewState ["TemporaryFolder"] = value;
    }
}
```

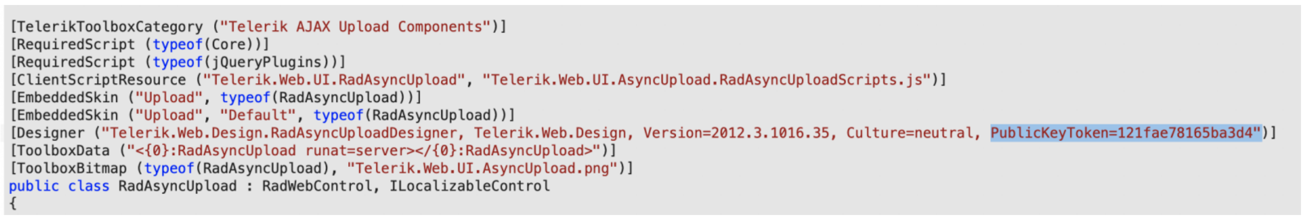
A screenshot taken from the decompiled Telerik UI DLL showing the default location of the uploaded temporary files. The DLL is often bundled with open source components e.g. select versions of DotNetNuke.Web.

The asynchronous uploader uses encryption as a security mechanism to prevent an attacker from influencing sensitive settings: such as the '*type'* of the uploaded file (used to prepare the deserializer), and the ultimate destination of the uploaded file on the server (values such as 'TempTargetFolder' and 'TargetFolder').

Therefore, for a successful exploit to occur, an attacker needs to know beforehand the encryption keys being used by the Telerik UI async uploader. These could be obtained as a result of the attacker exploiting older Telerik UI vulnerabilities on the same vulnerable IIS server.

There's also an easier route: if the server is using the default encryption keys, which were never regenerated afresh (see hardcoded `PublicKeyToken` below).

Older versions of `Telerik.Web.UI.dll` ship with a hardcoded `PublicKeyToken` and its private key counterpart in several places within the code. Unless these strings are manually changed, the attacker can trivially obtain prior knowledge necessary to conduct this exploit.



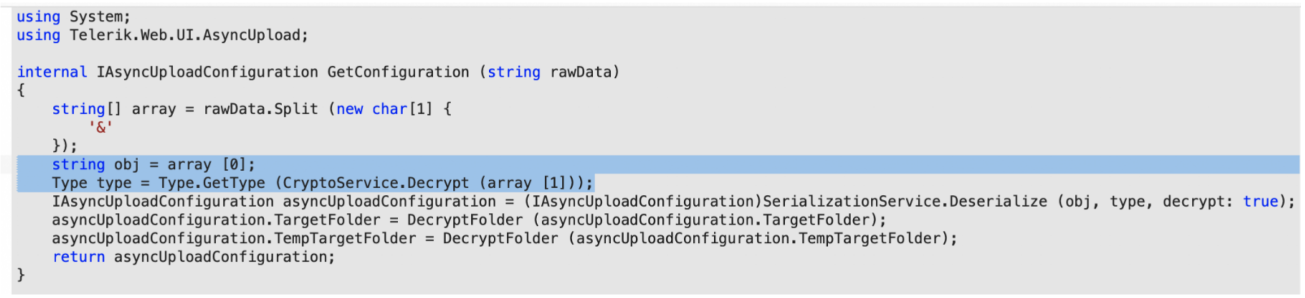
```
[TelerikToolboxCategory ("Telerik AJAX Upload Components")]
[RequiredScript (typeof(Core))]
[RequiredScript (typeof(jQueryPlugins))]
[ClientScriptResource ("Telerik.Web.UI.RadAsyncUpload", "Telerik.Web.UI.AsyncUpload.RadAsyncUploadScripts.js")]
[EmbeddedSkin ("Upload", typeof(RadAsyncUpload))]
[EmbeddedSkin ("Upload", "Default", typeof(RadAsyncUpload))]
[Designer ("Telerik.Web.Design.RadAsyncUploadDesigner, Telerik.Web.Design, Version=2012.3.1016.35, Culture=neutral, PublicKeyToken=121fae78165ba3d4")]
[ToolboxData ("<{0}:RadAsyncUpload runat=server></{0}:RadAsyncUpload>")]
[ToolboxBitmap (typeof(RadAsyncUpload), "Telerik.Web.UI.AsyncUpload.png")]
public class RadAsyncUpload : RadWebControl, ILocalizableControl
{
```

Older Telerik Web UI DLLs using a hardcoded `PublicKeyToken` at several places

Given the vulnerability isn't entirely new and improvements have been made to Telerik UI over the years to remediate older CVEs, the actual exploitation involves two key steps.

The first, is to trick the uploader into accepting a POST request (a "rawData" object, also referred to as "rauPOSTData") which looks legitimate but contains a carefully crafted DLL file of the attacker's choice rather than purely an "IAsyncUploadConfiguration" object that the deserializer is expecting. The second step involves influencing the application into deserializing objects of types *other than* IAsyncUploadConfiguration*,* such as DLLs.

NOTE: The `AsyncUploadConfiguration` class is simply the implementation of the `IAsyncUploadConfiguration` interface. I've used these names interchangeably.



```
using System;
using Telerik.Web.UI.AsyncUpload;

internal IAsyncUploadConfiguration GetConfiguration (string rawData)
{
    string[] array = rawData.Split (new char[1] {
        '&'
    });
    string obj = array [0];
    Type type = Type.GetType (CryptoService.Decrypt (array [1]));
    IAsyncUploadConfiguration asyncUploadConfiguration = (IAsyncUploadConfiguration)SerializationService.Deserialize (obj, type, decrypt: true);
    asyncUploadConfiguration.TargetFolder = DecryptFolder (asyncUploadConfiguration.TargetFolder);
    asyncUploadConfiguration.TempTargetFolder = DecryptFolder (asyncUploadConfiguration.TempTargetFolder);
    return asyncUploadConfiguration;
}
```

The "rawData" object represents the POST request. The '&' delimits the actual 'obj' (object) to be deserialized and its 'type' in the POST request. The 'type' parameter may be overridden to allow DLL deserialization, if the encryption keys are known to an attacker.

Having referred to the PoC, the exploit is conducted in the following steps:

1. The attacker first crafts a malicious POST request to the async upload file handler (WebResource.axd), specifying the type as the normal "Telerik.Web.UI.AsyncUploadConfiguration" which is what the deserializer expects. But in the AsyncUploadConfiguration JSON 'object' itself, the attacker provides a custom target directory path (where the file will be uploaded to on the server). Therefore, this is a fully valid AsyncUploadConfiguration object expected by the deserializer with the exception of a customized target upload path provided. Additionally, in the same request the attacker sneaks in the malicious DLL file. The DLL is simply uploaded in this step to the target directory (remote path) of the attacker's choice, and not deserialized.

2. The second step concerns the actual deserialization of the uploaded DLL. This time around, the attacker makes another request to the async file handler. Instead of sending a valid "AsyncUploadConfiguration" JSON object, however, the attacker sends a JSON object specifying the remote path (on the server) of the previously uploaded DLL and nothing else. The 'type' of the *object* is now specified as System.Configuration.Install.AssemblyInstaller (that of the DLL) with no other files uploaded in the request. Once the request is processed, the deserializer will attempt to load the malicious DLL present at the specified remote path, effectively resulting in remote code execution.

Therefore, with the knowledge of encryption keys to craft malicious requests and after running a successful two-step exploit, an attacker has gained the ability to execute arbitrary code on the enterprise server.

In malware campaigns involving the hacker group *Blue Mockingbird*, their choice of malice involves targeting enterprise IIS servers on a large scale, and turning them into Monero (XMR) cryptocurrency miners.

Remediation Guidance

It is important that any vulnerable IIS instances running Telerik UI components be immediately scanned, and patched for this vulnerability. Telerik's official advisory lists comprehensive scenarios in ways the vulnerability can be exploited and the appropriate remediation guidance.

Markus Wulftange of Code White GmbH has been credited with the discovery of the flaw and Paul Taylor (@bao7uo), for assisting with public disclosure and additional research.

Related Articles:

New IceApple exploit toolset deployed on Microsoft Exchange servers

U.S. Treasury sanctions Russian cryptocurrency mining companies

New malware targets serverless AWS Lambda with cryptominers

- Blue Mockingbird
- IIS
- Miners
- Monero

Ax Sharma

Ax Sharma is a Security Researcher and Tech Reporter. His works and expert analyses have frequently been featured by leading media outlets including Fortune, Business Insider, The Register, TechRepublic, etc. Ax's expertise lies in vulnerability research, malware analysis, and open source software. He's an active community member of the OWASP Foundation, Open Source Security Foundation (OpenSSF), and the British Association of Journalists (BAJ). Send any tips via email or Twitter DM.

- Previous Article
- Next Article

You may also like: