Friday the 13th: JSON Attacks

Alvaro Muñoz (@pwntester)
Oleksandr Mirosh

HPE Security
> whoarewe

- Alvaro Muñoz  
  - Security Research with HPE  
  🦉 @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
  • Is JSON/XML/<Put your favorite format here> any better?
  • Raise awareness for .NET deserialization vulnerabilities
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?
• 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 `java.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?
Object Reconstruction

• JSON libraries need to reconstruct objects by either:
  • Calling default constructor and using reflection to set field values
  • Calling default constructor and calling setters to set field values
  • Calling “special” constructors, type converters or callbacks
  • 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
  - `set_Source`
  - Arbitrary XAML load
  - Required to be able to work with setters of types derived from `IDictionary`
  - `set_(MethodName | ObjectInstance | ObjectType)`
  - Arbitrary Method Invocation
```csharp
if (!IsRefreshDeferred) 
    Refresh();

_initialLoadCalled = true;
BeginQuery();

QueryWorker(null);

data = InvokeMethodOnInstance(out e);

try 
{
    data = _objectType.InvokeMember(MethodName, 
    s_invokeMethodFlags, null, _objectInstance, parameters, 
    System.Globalization.CultureInfo.InvariantCulture);
} 
```
ObjectDataProvider

```json
 "ObjectInstance":{
   "MethodName": "Start"
   },
   "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
Gadgets: Java Edition

- `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

Arbitrary Getter call
- `org.antlr.stringtemplate.StringTemplate` (Java)
  - `toString`
  - Can be used to chain to other gadgets such as the infamous `TemplatesImpl.getOutputProperties()`
  - `set_DataMember`

XXE
- `System.Xml.XmlDocument/XmlDataDocument` (.NET < 4.5.2)
  - `set.InnerXml`
- `System.Data.DataViewManager` (.NET < 4.5.2)
  - `set_DataViewSettingCollectionString`
• Arbitrary Code Execution Requirements:
  1. Attacker can control type of reconstructed objects
     • Can specify Type
     • _type, $type, class, classname, javaClass, ...
     • Library loads and instantiate Type
  2. Library/GC will call methods on reconstructed objects
  3. There are gadget chains starting on method executed upon/after reconstruction
• 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
  2. Inspection of expected type
• Inspection of expected type’s object graph
  • Check assignability from provided type
  • In some cases it also create a whitelist of allowed types
• Vulnerable if
  • Expected type is user-controllable
  • Attacker can find injection member in object graph and no whitelist is applied
<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Type Name</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 + whitelist</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

```java
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
    • Type Resolver can be used to include this information.

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

- Performs a strict type graph inspection and whitelist creation.
- 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. Eg:

```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.
• It does not include Type discriminators unless **TypeNameHandling** setting other than **None** is used

• Performs an inspection of Expected Type’s 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,
    };

```csharp
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 JObjectReader(dynSaveOptions), typeof(SaveOptions));
    SaveWorkState = new SaveWorkState(this, entitiesArray);
}

public class SaveOptions {
    public bool AllowConcurrentSaves { get; set; }
    public object Tag { get; set; }
}
```
Demo 1: Breeze (CVE-2017-9424)
Similar Research

• 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
• 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
• Bridges to custom deserializer

```csharp
protected PSObject(SerializationInfo info, StreamingContext context)
{
    this.lockObject = new object();
    if (info == null)
    {
        throw PSTraceSource.NewArgumentNullException("info");
    }
    string source = info.GetValue("CliXml", typeof(string)) as string;
    if (source == null)
    {
        throw PSTraceSource.NewArgumentNullException("info");
    }
    PSObject obj2 = AsPSObject(PSSerializer.Deserialize(source));
    this.CommonInitialization(obj2.ImmediateBaseObject);
    CopyDeserializerFields(obj2, this);
}
```
```csharp
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;
            }
        }
    }
}
```
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.
private static PSConverter<object> FigureParseConversion(Type fromType, Type toType)
{
    ...

    else if (fromType.Equals(typeof(string)))
    {
        MethodInfo info = null;
        try
        {
            info = toType.GetMethod("Parse", bindingAttr, null, new Type[] { typeof(string), typeof(IFormatProvider) }, null);
        }
        ...
    }

```xml
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:System="clr-namepace:.System;assembly=mscorlib"
    xmlns:Diag="clr-namepace:System.Diagnostics;assembly=system">
    <ObjectDataProvider x:Key="LaunchCalc"
        ObjectType="{x:Type Diag:Process}"
        MethodName="Start">
        <ObjectDataProvider.MethodParameters>
            <System:String>calc</System:String>
        </ObjectDataProvider.MethodParameters>
    </ObjectDataProvider>
</ResourceDictionary>
```
# .NET Native Formatters

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Additional requirements</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinaryFormatter</td>
<td>Binary</td>
<td>No</td>
<td>ISerializable gadgets</td>
</tr>
<tr>
<td>SoapFormatter</td>
<td>SOAP XML</td>
<td>No</td>
<td>ISerializable gadgets</td>
</tr>
<tr>
<td>NetDataContractSerializer</td>
<td>XML</td>
<td>No</td>
<td>ISerializable gadgets</td>
</tr>
<tr>
<td>JavaScriptSerializer</td>
<td>JSON</td>
<td>Insecure TypeResolver</td>
<td>Setters gadgets</td>
</tr>
<tr>
<td>DataContractSerializer</td>
<td>XML</td>
<td>Control of expected Type</td>
<td>Setters gadgets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or knownTypes</td>
<td>Some ISerializable gadgets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or weak DataContractResolver</td>
<td></td>
</tr>
<tr>
<td>DataContractJsonSerializer</td>
<td>JSON</td>
<td>Control of expected Type</td>
<td>Setters gadgets</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or knownTypes</td>
<td>Some ISerializable gadgets</td>
</tr>
<tr>
<td>XmlSerializer</td>
<td>XML</td>
<td>Control of expected Type</td>
<td>Quite limited; does not work with interfaces</td>
</tr>
<tr>
<td>ObjectStateFormatter</td>
<td>Text, Binary</td>
<td>No</td>
<td>Uses BinaryFormatter internally;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TypeConverters gadgets</td>
</tr>
<tr>
<td>LosFormatter</td>
<td>Text, Binary</td>
<td>No</td>
<td>Uses ObjectStateFormatter internally</td>
</tr>
<tr>
<td>BinaryMessageFormatter</td>
<td>Binary</td>
<td>No</td>
<td>Uses BinaryFormatter internally</td>
</tr>
<tr>
<td>XmlMessageFormatter</td>
<td>XML</td>
<td>Control of expected Type</td>
<td>Uses XmlSerializer internally</td>
</tr>
</tbody>
</table>
Demo 2: NancyFX (CVE-2017-9785)

Install

```pm
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
• CSRF cookie

• Latest stable version used a `BinaryFormatter` serialized cookie (1.x)
  ```
  AAEAAAD/////AQABAAMBAw8UPkUCgAAAAkDAAAspLEeOrO0IgJBAAAA8DAAAAAgAAAAAt9dloO6qU2iUAuPUAtsq+UdOw5Qu1py8YhoCn5hv+PJCwAAAA
  AAABAAABAAAAAAAGAAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAA
  AAABAAABAAAAAAAGAAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAA
  AAABAAABAAAAAAAGAAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAAAAAAGAAA
  ```

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

• Pre-auth Remote Code Execution in both versions
Demo 2: NancyFX (CVE-2017-9785)
Generalizing the Attacks
• During unmarshaling, objects will need to be created and populated which normally mean calling setters or deserialization constructors.

• Arbitrary Code Execution Requirements:
  1. Attacker can control type to be instantiated upon deserialization
  2. Methods are called on the reconstructed objects
  3. Gadget space is big enough to find types we can chain to get RCE

• We can use our setter gadgets to attack most 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

```json
```

• 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.Diagnostic.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 = CBO0.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 (XmlNode 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 StringReader(xmlItem.InnerXml));

    // Use the Deserialized method to restore the object's state, and store it
    // in the HasTable
    hashTable.Add(key, xser.Deserialize(reader));
}
```
Wrap-Up
• **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