Trustwave®
Information Security & Compliance

FLEX, AMF 3 and BlazeDS: An Assessment

Jacob Carlson
Kevin Stadmeyer

Outline

I. FLEX
    I. Introduction
    II. ActionScript primer
    III. Security model
    IV. Communication
    V. Assessment tools
II. BlazeDS
    I. Introduction
    II. Security model
    III. Communication
    IV. Object instantiation and exposure
    V. Assessment tools
III. AMF3
    I. Introduction
    II. Data types
        I. Encodings
        II. Integer promotion
        III. Reference tables
        IV. Object traits
        V. Communication

Objectives

- Demystify
- Show developers how assessors will view your apps
- Help assessors do their jobs in this crazy moon world

Our Goals

- Show what happens on the bare metal
- Remove the abstractions
  - It's just two computers chattin' away
- Increase overall understanding
- Provide perspective

Flex Overview

- Execution
  - Executes within Flash ActionScript Virtual Machine
  - Flex 3 requires Flash Player 9
- MXML Files describe the layout and object interaction
- Magic Compiler Turns it into a .swf
  - Can be decompiled into “original” source
- Flash Player Is A Nicety Not Necessity
  - Subject to trivial decompilation
  - Normal JavaScript rules apply

Action Script Primer

- Object-Oriented Language
- Java-like syntax
- Developed to run hostile programs and in hostile environments
  - Your code executes in various sandboxes
- Restricts access to local & remote resources
- MXML is translated to ActionScript on compilation

Action Script Primer

- Adobe Example Code:

```
<mx:Script>
    <![CDATA[

            [Bindable]
            private var userName:String;

            [Bindable]
            private var roomName:String;

            private function joinRoom():void
            {
                  userName = connectPanel.userName;
                  roomName = connectPanel.roomName;
                  vs.selectedIndex = 1;
            }

    ]]>
</mx:Script>
```

Restricting Resource Access

- Flex Operates Under a Same-Domain Policy
  - Flex application can only access resources from the domain it was loaded from.
  - Flex applications must granted explicit permission to connect to ports lower then 1024
  - Exceptions are granted with a server-side crossdomain.xml file
  - Enforcement is implemented by FlashPlayer

Restricting Resource Access

- Crossdomain.xml

```
<xml version="1.0”>
<!-- http://www.trustwave.com/crossdomain.xml-->
<cross-domain-policy>
<allow-access-from domain="*.trustwave.com”/>
<allow-access-from domain=”trustwave.com" to-
ports=”4,8,15,16,23,42" />
</cross-domain-policy>
```

Flash Player Sandboxing

- Four Main Types
  - Remote
  - Local-with-filesystem
  - Local-with-networking
  - Local-trusted
  - avoidable with user permissions
- Environmental Information Restricted
  - Capabilities Class – read only
    - OS
    - MP3/Video Encoder
    - Screen Resolution
    - ETC.

Flash Player General Security

- Focuses on resources available to and from the flash player
  - URL's
  - Other SWF Files
  - Files
- Hierarchy of Permissions
  - admin settings
    - mms.cfg provides administrative control permissions
    - Intended for enterprise users
    - Won't affect yer mom
  - user settings
  - website settings
  - developer

Flash Player General Security

- Same domain policy
  - Based on DNS
    - Easily Avoidable
  - Wildcards no longer allowed
- Additional resource restrictions
  - Disk space constrained to 100k
    - If disk access is allowed at all
  - Outbound port filter
    - 20,21,23,25, etc
    - can still make a decent port scanner subject to cross domain

Flex Communication

- Mute applications aren't very useful
  - URLLoader
  - Socket
  - XML Socket

Flex Communication: URLLoader

- Surprisingly, this makes URL requests
- Subject to resource restriction
- Available Schemes
  - GET
  - POST
- Subject to port filtering

Flex Communication

- ActionScript Sockets
  - Allow essentially arbitrary network connections
  - Reads/writes binary data with no restrictions
  - Again, subject to resource restriction dependent on its sandbox.
  - Read, Write Methods: boolean, byte, bytes, double, float, int, multibyte, object, short, unsigned byte, unsigned int, unsigned short, utf, utf bytes.
  - Socket Policy Files, 843

Flex Communication (cont.)

- XML Sockets
  - Can maintain a continuous connection
    - Potential DOS via Socket Exhaustion
  - Each transmission is a complete xml doc
  - Again, subject to resource restriction dependent on its sandbox.
  - Close, Connect, and Send Methods
  - Commonly used to maintain realtime communications

Communications: Remoting Options

- Java, PHP, .Net, Ruby, Python
  - All have supported flex/amf interfaces
- RemoteObject can be used to directly invoke server objects
  - Flex handles serializations
  - No XML conversion
- Client server communications use channels
  - Standard (secured or not) AMF Channel
  - Polling
  - Real Time Streaming

Flex Communication (cont.)

- HTTPService
  - Can use GET, POST, HEAD, OPTIONS, PUT, TRACE, DELETE
  - Defaults to GET
  - ASCII
  - Headers

Flex Communication (cont.)

- WebService
  - Uses WSDL's
  - Text Format (ascii only)
  - Returns an Object, XML or e4x dependent on ResultFormat
  - WebService Object

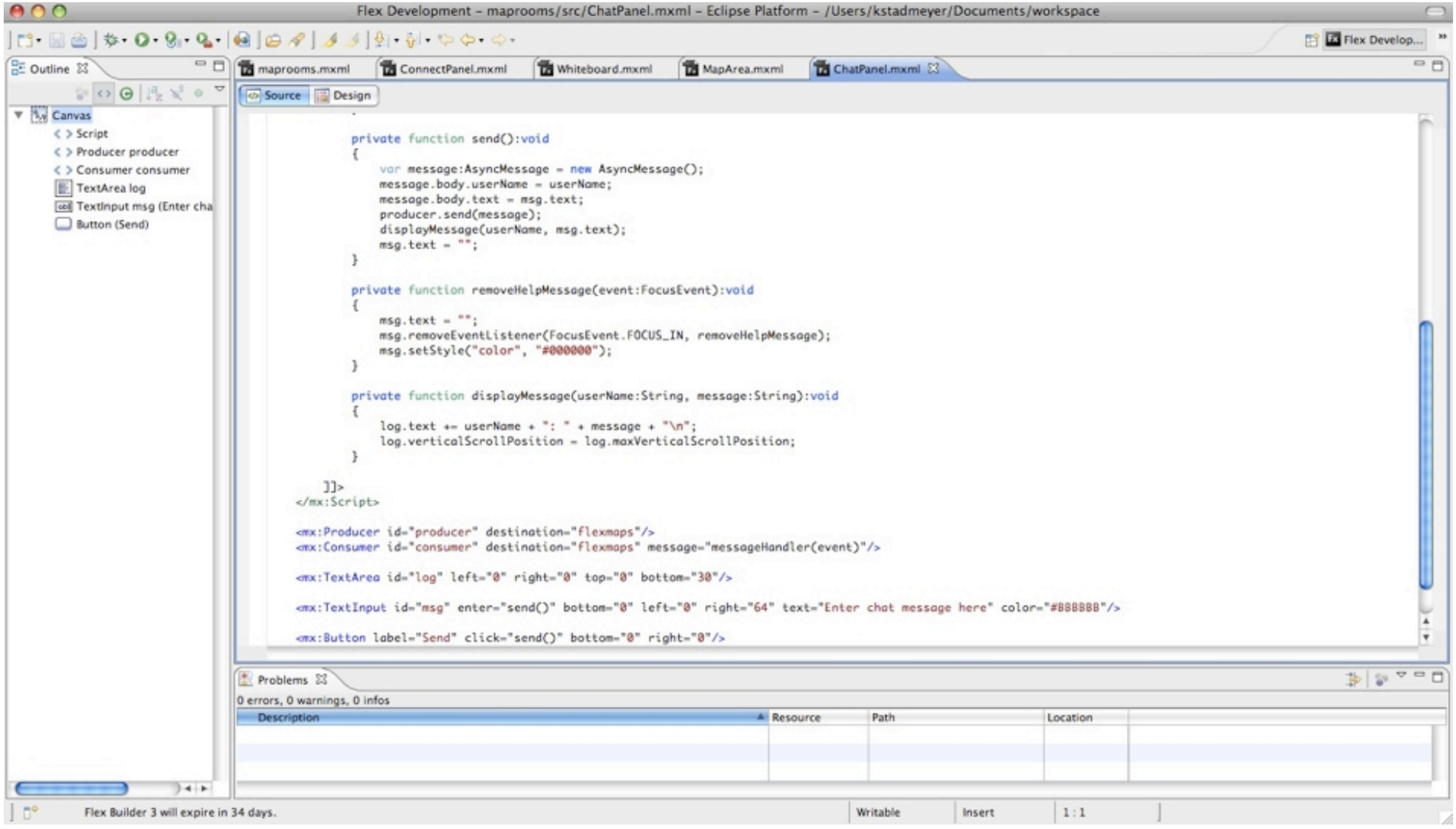
Flex Communication

- Remoting
  - What we concentrated on
  - Allows remote object calls
  - AMF3
  - Instantiates a RemoteObject object
  - if you instantiate a remote object, you have certain methods just because its a remote object and then you have the methods you get from the server
  - all properties are exposed

Assessment Tools

- Flash Decompilers
  - Most don't grok Flash 9/Flex
  - SoThink 4 and Later (cost $$)
- Debuggers
- Flash Player Debug Build
  - Part of the Flash SDK
- Sniffers
  - WireShark
- Proxies
  - Charles Proxy

Sample App In Eclipse

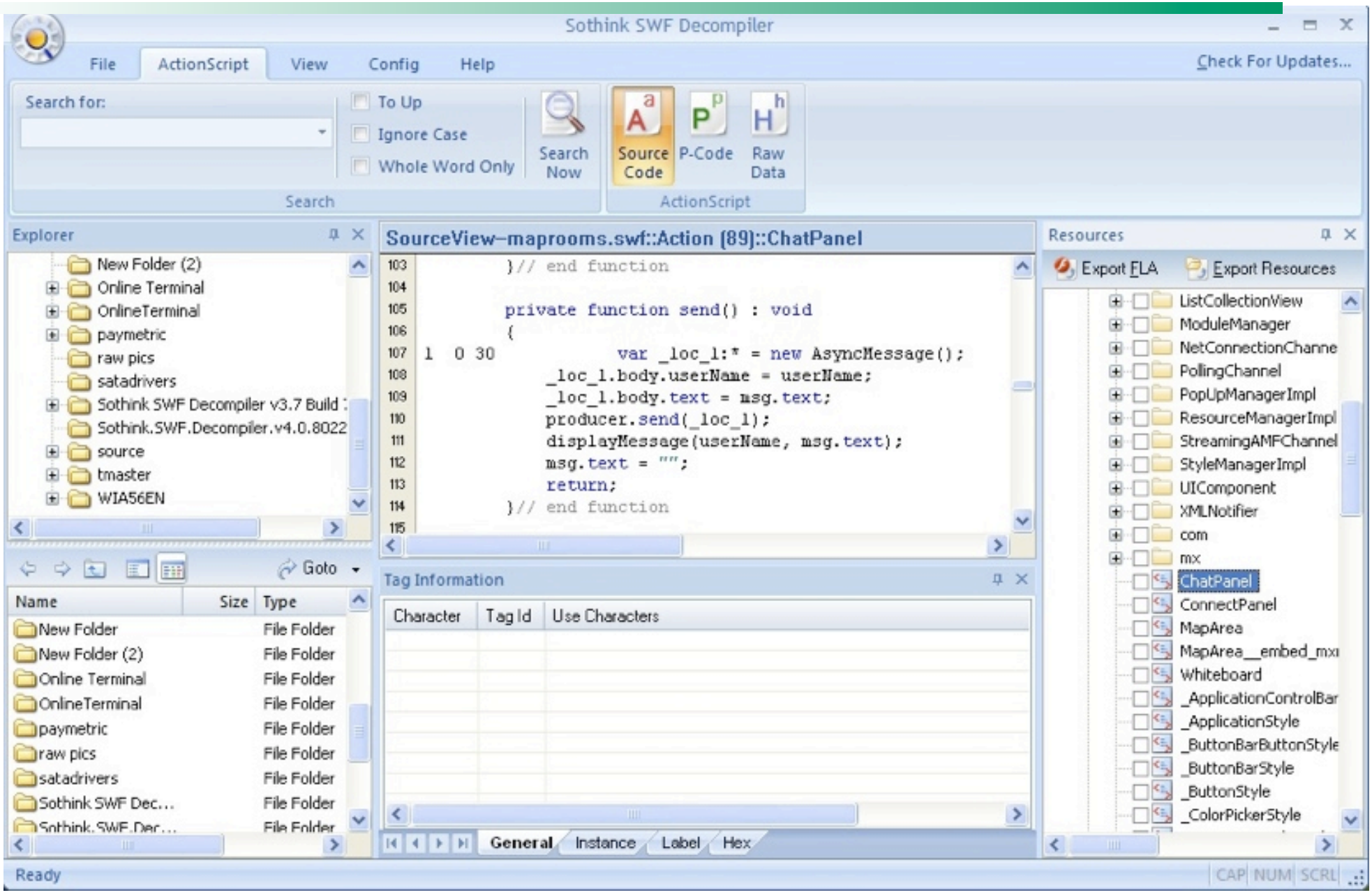


The Code

```
private function
send():void
{
     var message:AsyncMessage = new
       AsyncMessage();
    message.body.userName = userName;
    message.body.text = msg.text;
    producer.send(message);
    displayMessage(userName, msg.text);
    msg.text = "";
}

<mx:TextInput id="msg"
    enter="send()"
    bottom="0"
    left="0"
    right="64"
    text="Enter chat message here"
    color="#BBBBBB" />
```

In SoThink 4



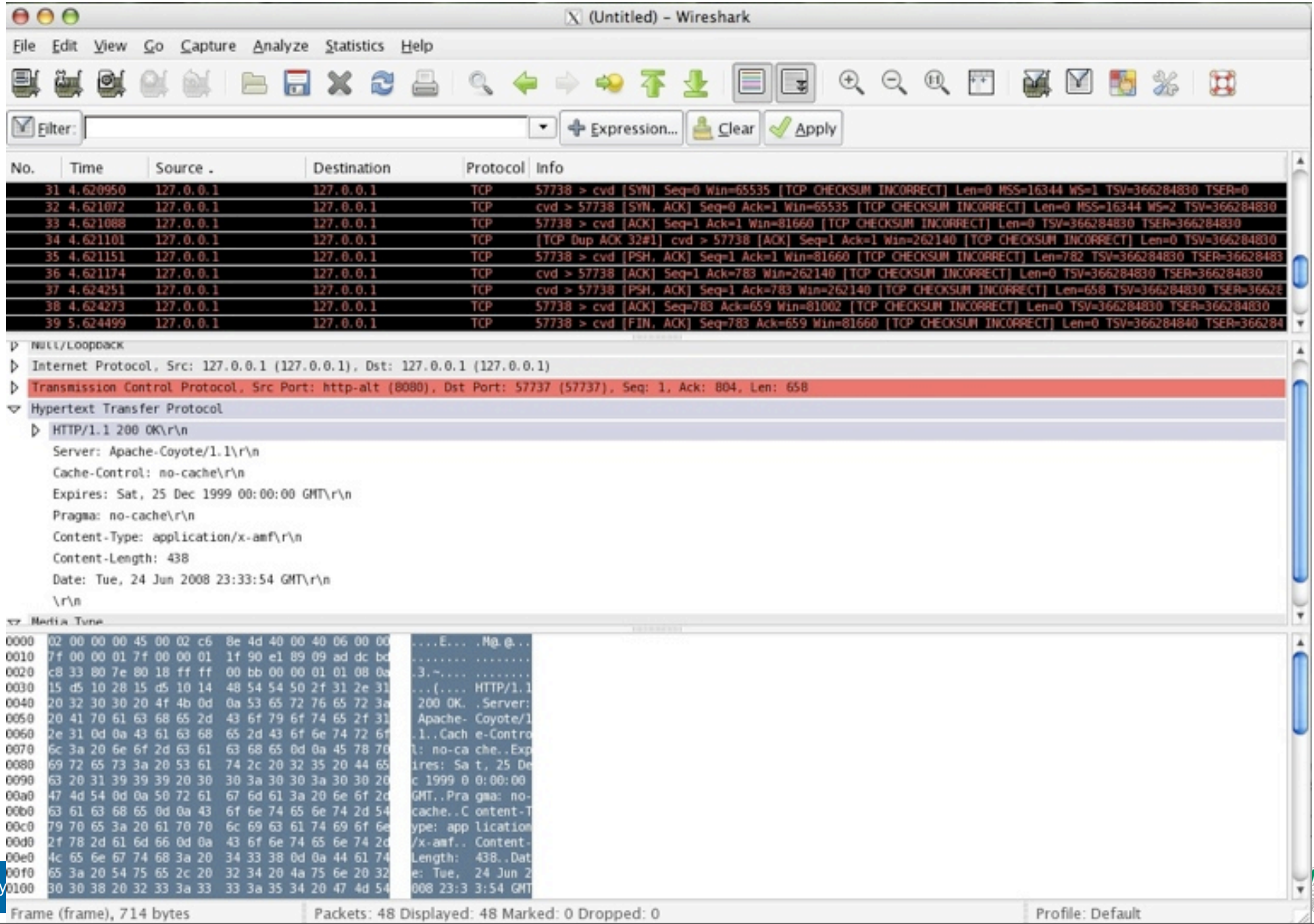
Trustwave®
Information Security & Compliance

Decompiled Code

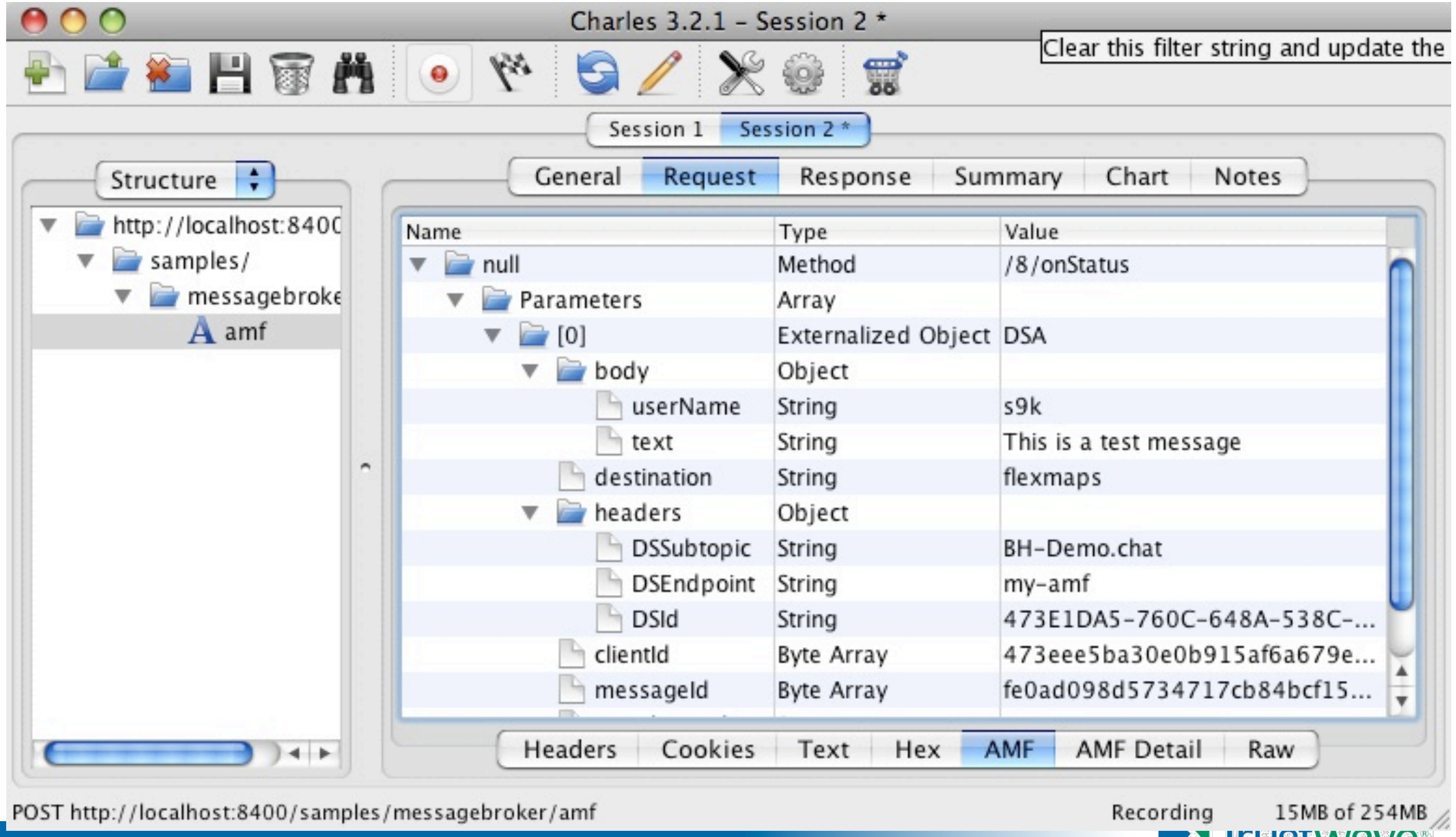
```
public function get msg() : TextInput
        {
            return this._108417msg;
        }

private function send() : void
        {
            1  0 30              var _loc_1:* = new AsyncMessage();
            _loc_1.body.userName = userName;
            _loc_1.body.text = msg.text;
            producer.send(_loc_1);
            displayMessage(userName, msg.text);
            msg.text = "";
            return;
        }
```

In WireShark



In Charles Proxy



BlazeDS

- Java-based Remoting/Messaging server
- Supports communication via AMF rather than traditional XML/SOAP methods
- Messaging component allows for publisher/subscribe communication
- Remoting component allows Flex applications to invoke methods on server-side objects

BlazeDS – Components

- Service
  - Remoting Or Messaging
- Destination
  - Remote Object or Message Topic
- Channel
  - Communication protocol
- Adapter
  - Destination-specific processing
- Security constraints

BlazeDS – Destinations

- Server-side class providing implementation- usually
- RPC Service Destinations
  - Server-side class to be accesses/instantiated remotely.
- Message Topic Destinations
  - Producers send messages to a messaging destination
  - Consumers subscribe to destinations
- Proxy Destinations
  - Web/HTTP Service destinations

BlazeDS – Sidebar on Proxies

- Flex applications can forward HTTP payloads through a BlazeDS server
- Destinations must be defined on server
  - Wildcards are supported, and encouraged
- Allows use of additional HTTP methods for mx:HTTPService
  - HEAD, OPTIONS, TRACE, and DELETE
- Requests aren't always controlled by Flash Player

BlazeDS – Channels

- Decouple endpoints from network and protocol specifics
- Client-side channels package messages for delivery across the network
- Server-side channels unpackage for delivery to Brokers
- Services can have multiple channels

BlazeDS – Adapters

- Code to handle requests prior to delivery to server-side objects
- Gets a message from the Channel to deliver to Destination
- Default adapters exist to handle most situations

BlazeDS – Security Constraints

- Provide Authentication and Authorization
- Authentication
  - Custom
  - Built-in
- Authorization
  - Role-based
- Method inclusion/exclusion

Security– Global Definition

```
<security>
    <security-constraint id="trusted-users">
        <auth-method>Custom</auth-method>
        <roles>
            <role>trustedUsers</role>
        </roles>
    </security-constraint>
    <login-command class="flex.messaging.security.TomcatLoginCommand"
        server="Tomcat"/>
</security>
```

Security– Destination Definition

```
<destination id="sampleRemoteObject">
    <security>
        <security-constraint ref="trusted-users" />
    </security>
    <properties>
        <source>flex.blackHat.sampleRemoteObject</source>
        <include-methods>
            <method name="myPublicMethod"/>
            <method name="mySensitiveMethod"
                security-constraint="admin-users"/>
        </include-methods>
    </properties>
</destination>
```

BlazeDS – Messaging Overview

- Asynchronous communication between producers and consumers
  - Producers send messages
  - Consumers accept messages
  - Clients and servers can be either, neither or both
- Built-in support for Java Message Service

BlazeDS – Message Service

- Server-side abstraction on top of AMF/RTMP
- Clients publish to a Message Service
  - Clients in this sense do not just mean Flex applications
- Message Service use Channels for communication
- Adapters handle message specifics
- Message Service delivers to a destination

BlazeDS – Remoting Overview

- Asynchronous call/response model
- Server objects are accessible without creating web services
- Server-side resources are proxied by BlazeDS
- Also provide access to WebServices and RemoteObjects

BlazeDS – Remoting Definition

```
<service id="remoting-service" class="flex.messaging.services.RemotingService">
    <adapters>
       <adapter-definition id="java-object" default=”true”
          class="flex.messaging.services.remoting.adapters.JavaAdapter"/>
    </adapters>
    <default-channels>
       <channel ref="my-amf" />
    </default-channels>
    <destination id="sampleRemoteObject">
       <properties>
          <source>flex.blackhat.sampleRemoteObject</source>
       </properties>
    </destination>
</service>
```

BlazeDS – Returning Objects

- List of object properties returned to client
- All public methods are accessible by default
  - Can be controlled via security contraints
- Methods and properties can return objects not named in remoting-config.xml

BlazeDS – Objects on the Wire

```
public classintegerDemo{
      private int charAsInt;
      private int ucharAsInt;
      private int shortAsInt;
      private int ushortAsInt;
      private int intAsInt;
      private int uintAsInt;

      public int getCharAsInt(){ return(charAsInt); }
      public void setCharAsInt(int x) { charAsInt = x;       }
      public int getUCharAsInt() {return(ucharAsInt); }
      public void setUCharAsInt(int x) { ucharAsInt = x; }
      public int getShortAsInt() { return(shortAsInt); }
      public void setShortAsInt(int x) { shortAsInt = x;     }
      public int getUShortAsInt() {         return(ushortAsInt); }
      public void setUShortAsInt(int x) {  ushortAsInt = x; }
      public int getIntAsInt()     { return(intAsInt);       }
      public void setIntAsInt(int x) { intAsInt = x; }
      public int getUIntAsInt() { return(uintAsInt); }
      public void setUIntAsInt(int x) { uintAsInt = x; }
}
```

BlazeDS – Objects on the Wire

```
00000470: 03 00 00 00 01 00 0B 2F - 35 2F 6F 6E 52 65 73 75 |       /5/onResu
00000480: 6C 74 00 00 FF FF FF FF - 11 0A 81 03 55 66 6C 65 |lt           Ufle
00000490: 78 2E 6D 65 73 73 61 67 - 69 6E 67 2E 6D 65 73 73 |x.messaging.mess
000004a0: 61 67 65 73 2E 41 63 6B - 6E 6F 77 6C 65 64 67 65 |ages.Acknowledge
000004b0: 4D 65 73 73 61 67 65 13 - 74 69 6D 65 73 74 61 6D |Message timestam
000004c0: 70 0F 68 65 61 64 65 72 - 73 09 62 6F 64 79 1B 63 |p headers body c
000004d0: 6F 72 72 65 6C 61 74 69 - 6F 6E 49 64 13 6D 65 73 |orrelationId mes
000004e0: 73 61 67 65 49 64 15 74 - 69 6D 65 54 6F 4C 69 76 |sageId timeToLiv
000004f0: 65 11 63 6C 69 65 6E 74 - 49 64 17 64 65 73 74 69 |e clientId desti
00000500: 6E 61 74 69 6F 6E 05 42 - 71 AC C1 4D 56 40 00 0A |nation Bq  MV@
00000510: 03 01 0A 63 29 62 6C 61 - 63 6B 68 61 74 2E 69 6E |   c)blackhat.in
00000520: 74 65 67 65 72 44 65 6D - 6F 13 55 49 6E 74 41 73 |tegerDemo UIntAs
00000530: 49 6E 74 11 69 6E 74 41 - 73 49 6E 74 15 55 43 68 |Int intAsInt UCh
00000540: 61 72 41 73 49 6E 74 13 - 63 68 61 72 41 73 49 6E |arAsInt charAsIn
00000550: 74 15 73 68 6F 72 74 41 - 73 49 6E 74 17 55 53 68 |t shortAsInt USh
00000560: 6F 72 74 41 73 49 6E 74 - 05 C1 E0 00 00 00 00 00 |ortAsInt
```

BlazeDS- Summary

- Unexpectedly large attack surface
  - Channels, adapters, etc.
- Each component is responsible for its own safety
- Classes must be defined carefully
  - Review extremely important
- Don’t return objects unless absolutely necessary

AMF

- AMF3 is the current version
- Free and Open
- Compact Binary Format
- Allows ActionScript object graph serialization
- Extension to AMF0
  - Adds new data types
  - Added support for sending traits and strings by reference

AMF –Data Types

- Denoted with a one byte marker
  - Simple Types
    - Integer- [1-4] bytes
    - Double- 8 bytes (IEEE-754 double)
    - True/False (actually two separate types)
  - Complex Types
    - String- Variable length
    - XMLDocument- Variable length
    - Array- Variable length
    - Object- Variable length
- Complex Objects can be sent by value or reference

AMF- Reference Tables

- References point to values
- Specifies index into a table
- Persist between requests
- Apply to
  - Objects
  - Arrays
  - Dates
  - XMLDocument
  - XML
  - Byte Arrays
  - Strings

AMF – Arrays (type 0x9)

- Strict (index/value) or Associative (key/value)
- Dense: No gaps between elements
- Sparse: At least 1 gap between elements
- Can contain references

AMF – Arrays (type 0x9)

- Array of 10 integers

```
09 15 01 04 00 04 14 04 28 04 3C 04 50 04 64 04   ........(.<.P.d
78 04 81 0C 04 81 20 04 81 34                     x..... ..4
```

- Array of 10 distinct strings

```
09 15 01 06 13 45 6C 65 6D 65 6E 74 20 30 06 13   .....Element 0
45 6C 65 6D 65 6E 74 20 31 06 13 45 6C 65 6D 65   ..Element 1..Ele
6E 74 20 32 06 13 45 6C 65 6D 65 6E 74 20 33 06   ment 2..Element
13 45 6C 65 6D 65 6E 74 20 34 06 13 45 6C 65 6D   3..Element 4..El
65 6E 74 20 35 06 13 45 6C 65 6D 65 6E 74 20 36   ement 5..Element
06 13 45 6C 65 6D 65 6E 74 20 37 06 13 45 6C 65   6..Element 7..El
6D 65 6E 74 20 38 06 13 45 6C 65 6D 65 6E 74 20   ement 8..Element
39                                                9
```

- Array of 10 duplicate strings

```
09 15 01 06 11 45 6C 65 6D 65 6E 74 20
```

AMF – Strings (type 0x6)

- Arrays of UTF-8 encoded byte arrays
- Strings cannot be longer than 2^28 – 1 (256M)
- String reference table cannot hold more than 2^28 – 1 unique strings
- The empty String is never sent by reference
- Not terminated

AMF – Strings (cont.)

- Preceded by length specifier
  - (strlen(string)) * 2) + 1
  - For UTF8-1 and UTF8-2 this is not the actual number of bytes
  - ‘const string 1’ becomes
    - `06 1D 63 6F 6E 73 74 20 73 74 72 69 6E 67 20 31`

AMF –Integers (type 0x4)

- Stored in network byte order
- Bits 31-29 used as flags
  - Max integer value is 0x3fffffff
  - Larger values are promoted to doubles
    - Demoted when decoded by recipient
- Variable-length
  - 1-4 bytes long
  - Top 3 bits still used as flags
    - In C terms:
      - char is sent as 1 byte
      - u_char and short sent as 2 bytes
      - u_short sent as 3 bytes
      - Int and u_int sent as 4 bytes

AMF – Objects (0x0a)

- Typed object
  - [0xa][size_of_object_name][object_name]
  - For each named property
    - [size_of_property_name][property_name]
  - For each named property
    - [property_type][property_value]

```
–          0A 63 29 62 6C - 61 63 6B 68 61 74 2E 69 |    c)blackhat.i
–6E 74 65 67 65 72 44 65 - 6D 6F 13 55 49 6E 74 41 |ntegerDemo UIntA
–73 49 6E 74 11 69 6E 74 - 41 73 49 6E 74 15 55 43 |sInt intAsInt UC
–68 61 72 41 73 49 6E 74 - 13 63 68 61 72 41 73 49 |harAsInt charAsI
–6E 74 15 73 68 6F 72 74 - 41 73 49 6E 74 17 55 53 |nt shortAsInt US
–68 6F 72 74 41 73 49 6E - 74 05 41 B0 00 00 00 00 |hortAsInt A
```

AMF – Implementation

- Length fields
- Staying synchronized
- Detecting bad offsets
- Detecting bad reference table indexes

Trustwave®

Information Security & Compliance

Questions?