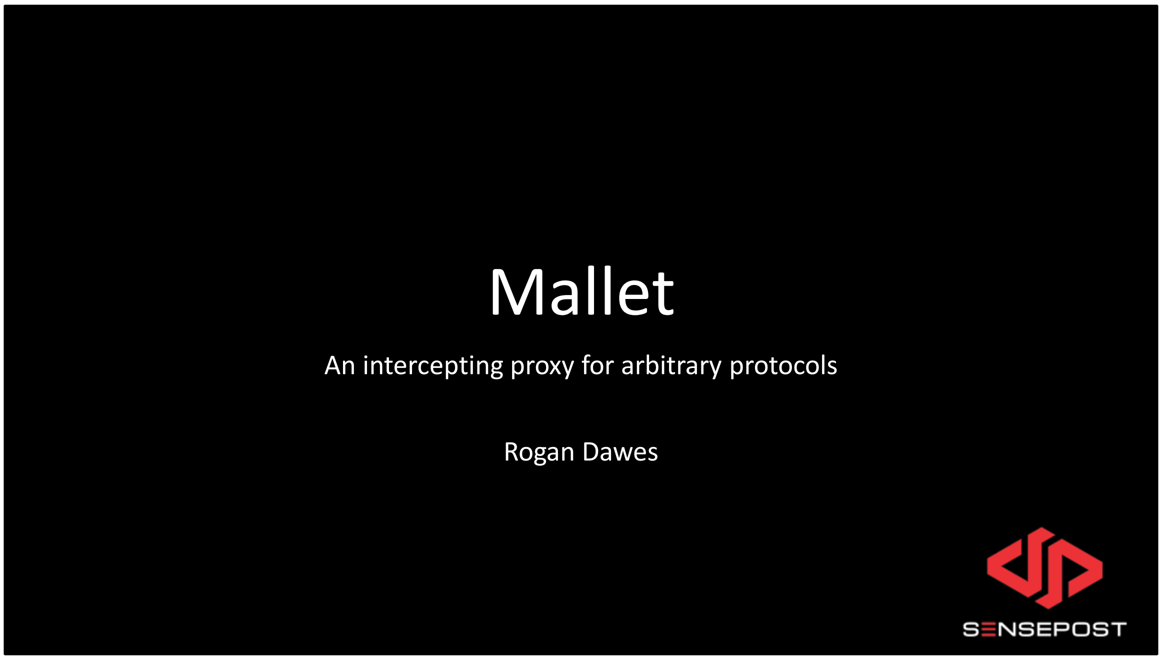


Mallet

An intercepting proxy for arbitrary protocols

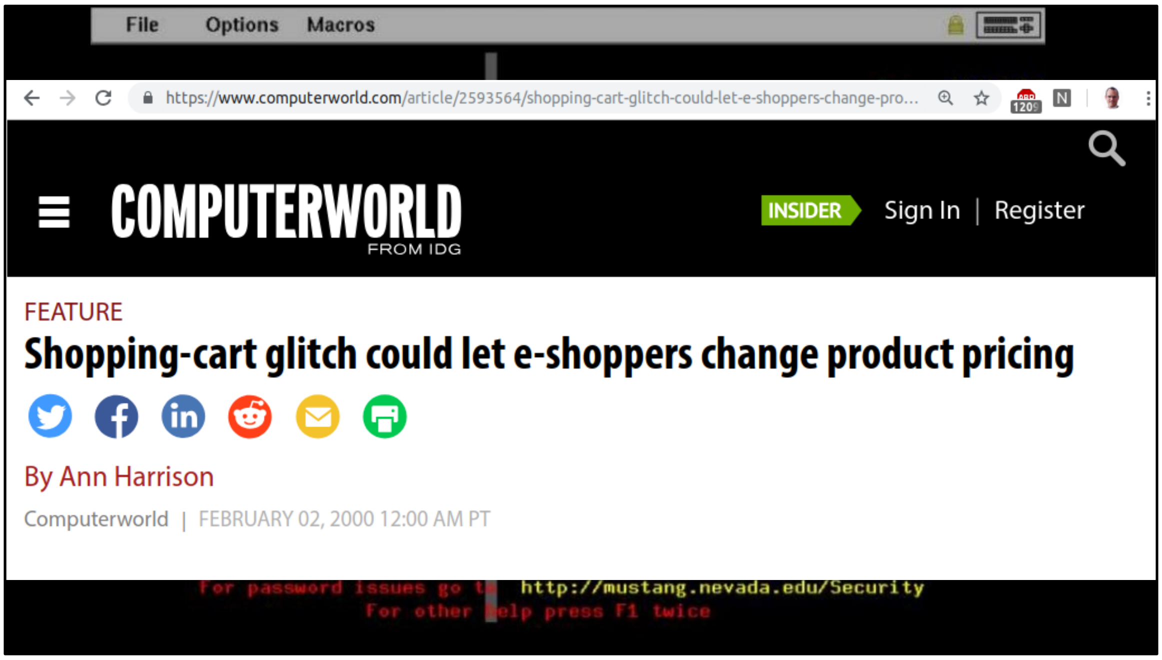
Rogan Dawes

Distributed applications (i.e. applications that operate on more than one computer) are often interesting from a security perspective, particularly when the communication between those nodes crosses a security boundary.
For instance, when part of the application executes on an untrusted computer, or transits an untrusted network.

Over the last 25 years or so, we have seen a shift from uncountable custom wire-level protocols, to applications making use of HTTP and HTTPS. HTTP IS the universal firewall traversal protocol, after all!

And along with the proliferation of HTTP-based applications has come security tooling to enable analysts to inspect and modify those HTTP messages in transit.

From Achilles, WebScarab, Paros, Charles Proxy, Fiddler, Burp and Zap, there are numerous options to choose from if you want to analyse applications communicating using HTTP. And of course, there are numerous automated web application scanners available to simulate a client and test the applications.


File
Options
Macros
https://www.computerworld.com/article/2593564/shopping-cart-glitch-could-let-e-shoppers-change-pro...
COMPUTERWORLD
FROM IDG
INSIDER
Sign In | Register

FEATURE

Shopping-cart glitch could let e-shoppers change product pricing

By Ann Harrison

Computerworld | FEBRUARY 02, 2000 12:00 AM PT

```
For password issues go to  http://mustang.nevada.edu/Security
         For other help press F1 twice
```

In 2000, ISS warned about shopping carts that were using hidden fields to manage pricing information, and how attackers could manipulate these hidden fields to change the prices of items on sale.

In 2014, Dominic White released a tool called Birp (Big Iron Recon & Pwnage), which exploited hidden fields in the TN3270 protocol., which was first documented in an RFC in 1994, after being in use for several years prior to that!

So, how are custom protocols different from HTTP?

On the one hand, we have a nice orderly protocol like HTTP, which is a simple, text-based (human readable), request-response protocol
It’s also stateless, allowing for requests to be captured on a single TCP connection, and sent to the server on a completely new connection that can be immediately torn down afterwards. For an intercepting proxy, this sort of capability is quite important! We’ll dig into this a bit later.

- Arbitrary protocols have many additional considerations, like sequence (who talks first), response ordering, connection/session state, data timeouts, message delimiters/framing, etc.

- As a result, most proxies for arbitrary protocols are once-off, hard-coded implementations for a specific instance, often not shared/sharable. Or else wrappers around socat!

Prior work

- OWASP Proxy (a library for writing proxies)
- Martin Holst Swende - Hatkit Proxy
- Intrepidus Group’s Mallory (2010!)
- jcrugzz/tcp-proxy
- ickerwx/tcpproxy
- praetorian-inc/Trudy
- Simone Margaritelli - Bettercap
- summitt/Burp-Non-HTTP-Extension

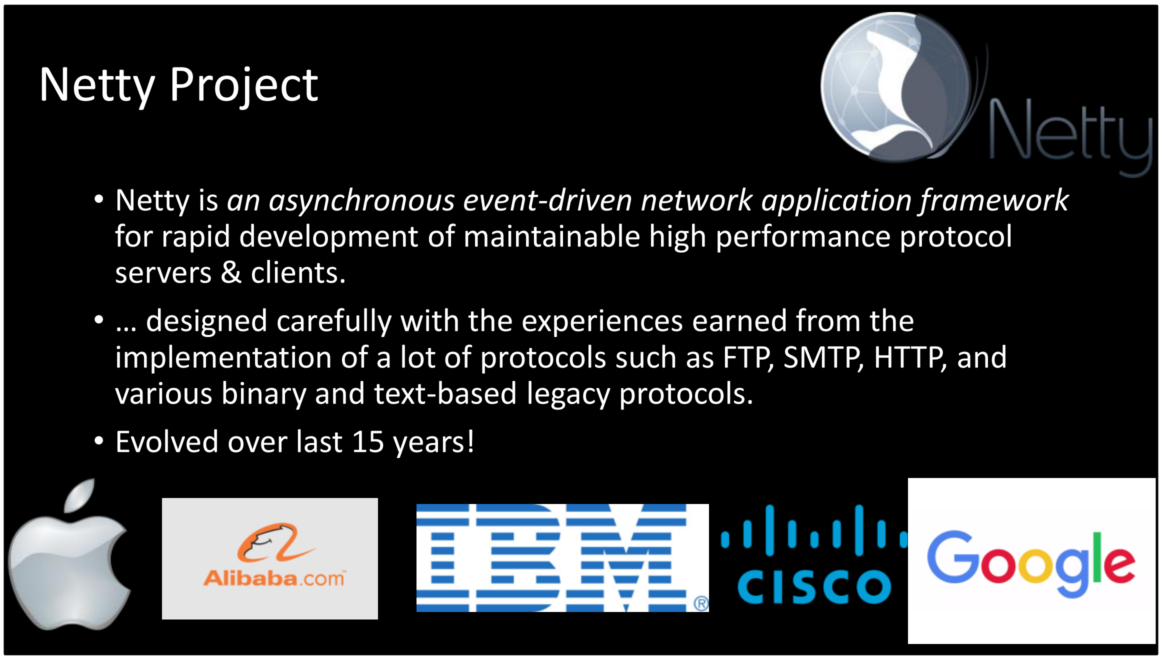
Common pattern is complex installation (Virtual machines, iptables rules, PPTP servers, etc), and limited protocol support (mostly HTTP/S), and 1-2 additional protocols at a demonstration level

The only conclusion I can come to is that making tooling to support analysis of arbitrary protocols is harder than it may seem initially!

With the background of the existing prior art, I had a couple of objectives for Mallet:

I wanted it to be as easy to use as possible, not requiring Virtual Machines, IPTables rules, and routing changes just to get traffic to the application.

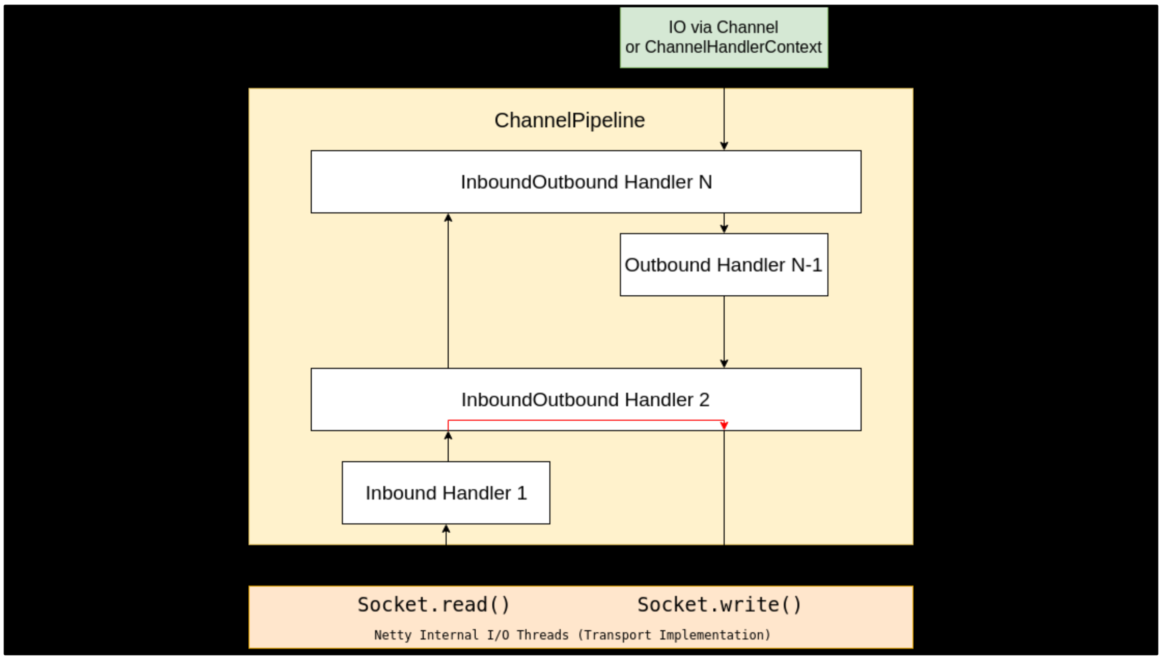
And I wanted there to be a large body of existing protocol support, so that you don't have to reinvent the wheel



Netty Project

- Netty is *an asynchronous event-driven network application framework* for rapid development of maintainable high performance protocol servers & clients.
- … designed carefully with the experiences earned from the implementation of a lot of protocols such as FTP, SMTP, HTTP, and various binary and text-based legacy protocols.
- Evolved over last 15 years!

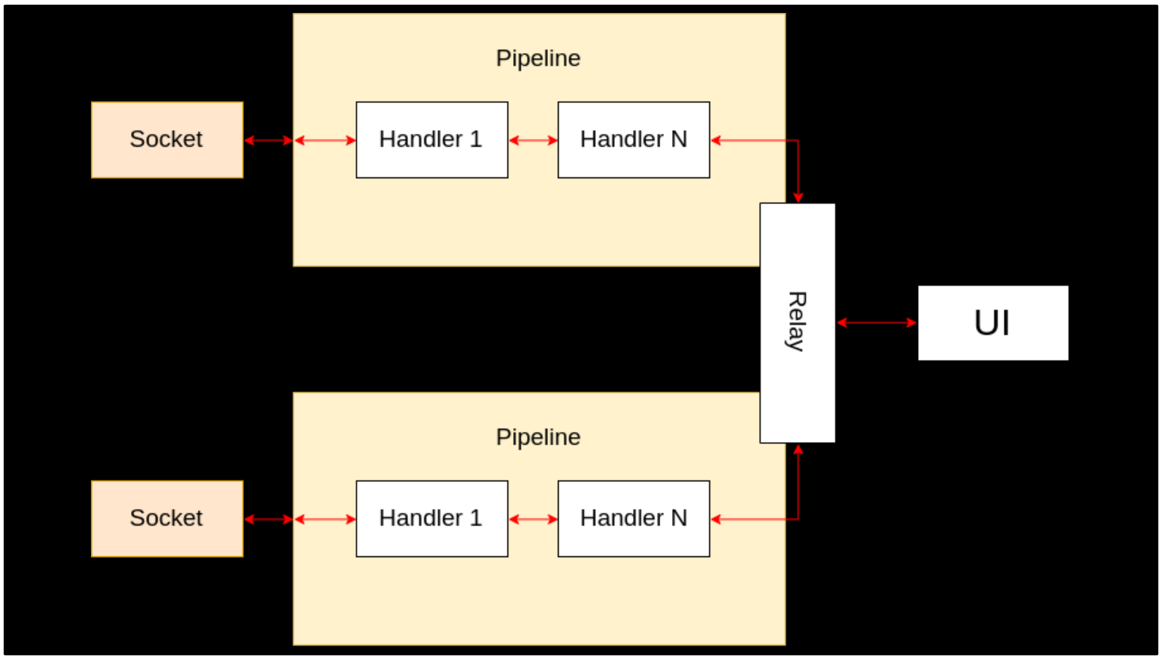
- A mature (2002!), high performance (10's of Petabytes/day at Apple, 400,000 instances as of 2016)
- In use by MANY large companies (https://netty.io/wiki/adopters.html)
- Lots of existing Protocol Decoders/Encoders, and body of knowledge/support for creating new ones!
- Great programming model supports "construction of pipelines" through composition, rather than by wrapping as was done in OWASP Proxy



One of the fundamental building blocks of the Netty framework is the ChannelPipeline. It is what defines the processing that occurs when a message is sent or received.

This illustrates some features of it, that it is composed of a number of handlers in sequence, that can choose to process incoming messages, or outgoing messages, or both.

Handlers can either process the incoming message and forward it, or respond to it directly.



This illustrates how to construct a proxy using two pipelines, by sharing a Relay handler between two pipelines.

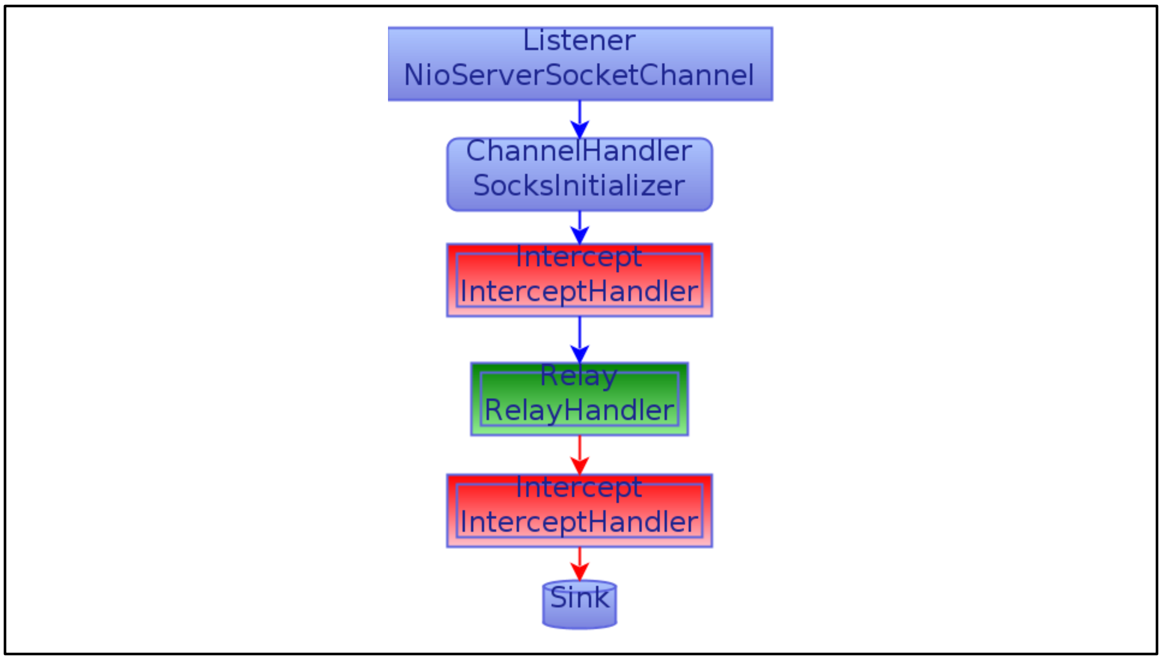
This is also responsible for making the messages available to the UI, where they can be viewed, edited and forwarded.

One of the ways Mallet makes it easy to intercept traffic is by making use of Netty's built-in Socks protocol support

That makes it trivial to intercept most TCP-based traffic, using tools such as tsocks, proxychains, etc, on Linux/OS X, and ProxyCap et al on Windows

Many applications even have built-in SOCKS proxy support. Most notably web browsers or other HTTP User-Agents, but other applications may also allow for upstream proxy configuration. In some cases, it can be configured at an Operating System level, and all programs making use of the OS-specific libraries will use the proxy automatically.

It is far simpler to configure tsocks or proxychains, and wrap the applications you are trying to intercept, than setting up routing and IPTables/PF rules!



This is an example of the default graph that is created when you start Mallet.

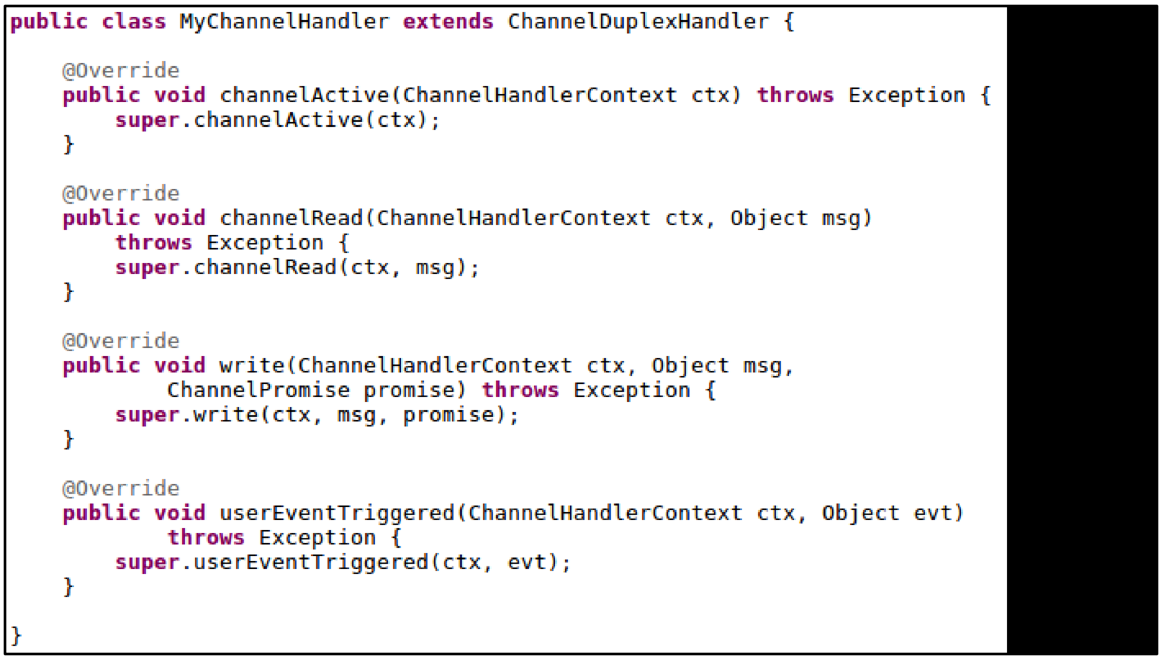
The Listener accepts incoming connections on the InetSocketAddress passed as a parameter, using the NioServerSocketChannel class.

The SocksInitializer aggregates a number of individual handlers that make up a functioning Socks Server.

The InterceptHandler records all events passing through it, and passes that information to the UI. Importantly, it only records events on its own Channel.

The RelayHandler is responsible for opening the outbound connection, and copying messages read on one channel as writes on the other channel.

And the Sink is just a marker for the end of the pipeline.



```
public class MyChannelHandler extends ChannelDuplexHandler {

    @Override
    public void channelActive(ChannelHandlerContext ctx) throws Exception {
        super.channelActive(ctx);
    }

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg)
        throws Exception {
        super.channelRead(ctx, msg);
    }

    @Override
    public void write(ChannelHandlerContext ctx, Object msg,
            ChannelPromise promise) throws Exception {
        super.write(ctx, msg, promise);
    }

    @Override
    public void userEventTriggered(ChannelHandlerContext ctx, Object evt)
            throws Exception {
        super.userEventTriggered(ctx, evt);
    }

}
```

This just illustrates the main methods that you may wish to override when writing a Handler.

There are others in the interface, but these are likely to be the most useful.

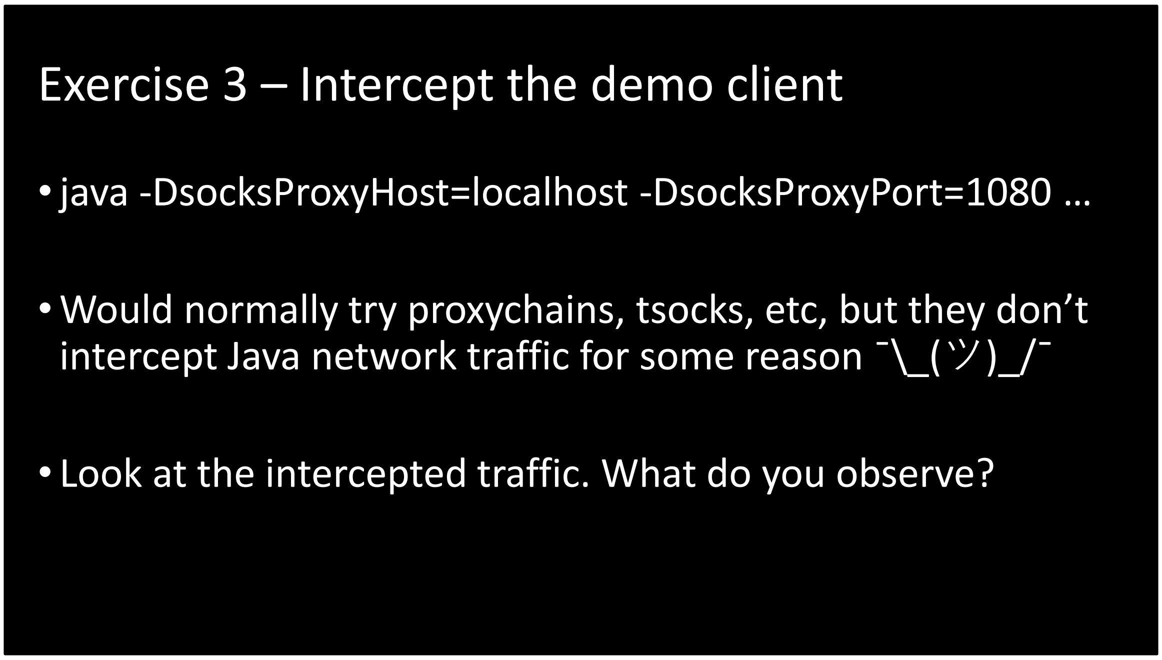
The ChannelHandlerContext can be thought of as a node in a linked list, having details of the previous and following handlers, and a number of methods for communicating with them, as well as a reference to the Channel that the handler is in.

Exercise 1 – Obtain Source and compile Mallet

- Prequisite: Java (1.8+) and Maven 3
- git clone https://github.com/SensePost/Mallet
- cd Mallet
- mvn package

Exercise 2 – Run Mallet and Demo apps

- java -jar target/mallet-1.0-SNAPSHOT.jar
- java -cp target/mallet-1.0-SNAPSHOT.jar demo.Server localhost 9999
- java -cp target/mallet-1.0-SNAPSHOT.jar demo.Client localhost 9999



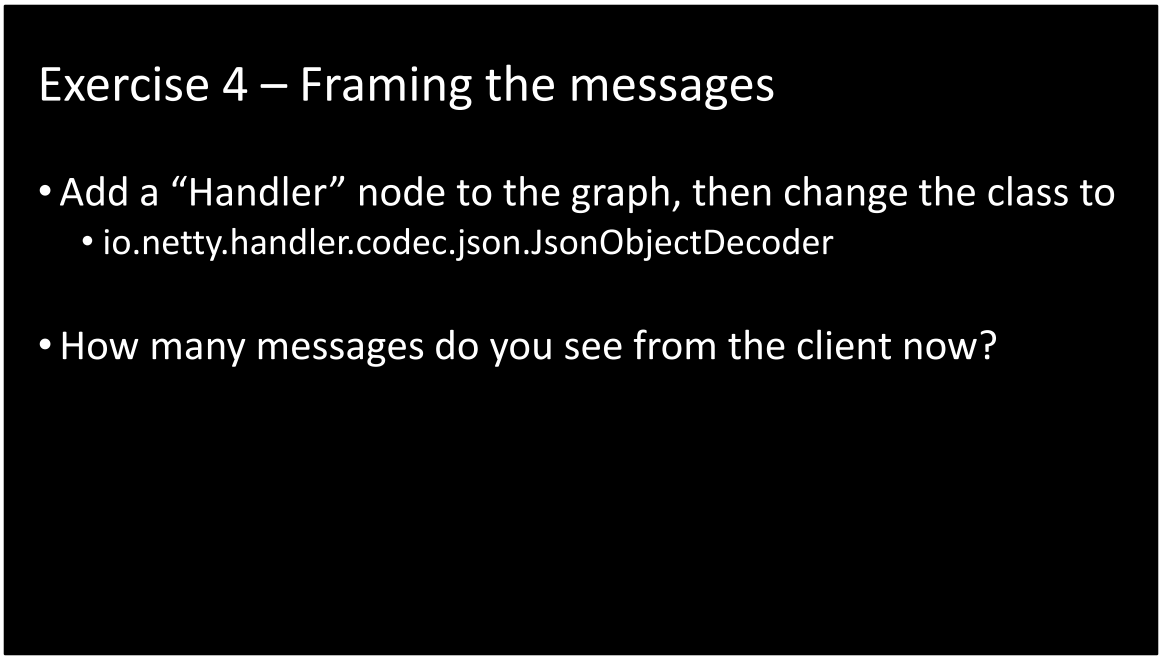
Exercise 3 – Intercept the demo client

- java -DsocksProxyHost=localhost -DsocksProxyPort=1080 …
- Would normally try proxychains, tsocks, etc, but they don’t intercept Java network traffic for some reason ¯_(ツ)_/¯
- Look at the intercepted traffic. What do you observe?

Fragmented message from the client to the server

Message Framing

- Messages can be framed in a number of ways
  - Prepended Length Field
  - Trailing “special character(s)”, such as CRLF, Null bytes, etc
  - Custom
- JSON messages end when the number of opening and closing braces match up
- XML messages end when the opening element is closed



Exercise 4 – Framing the messages

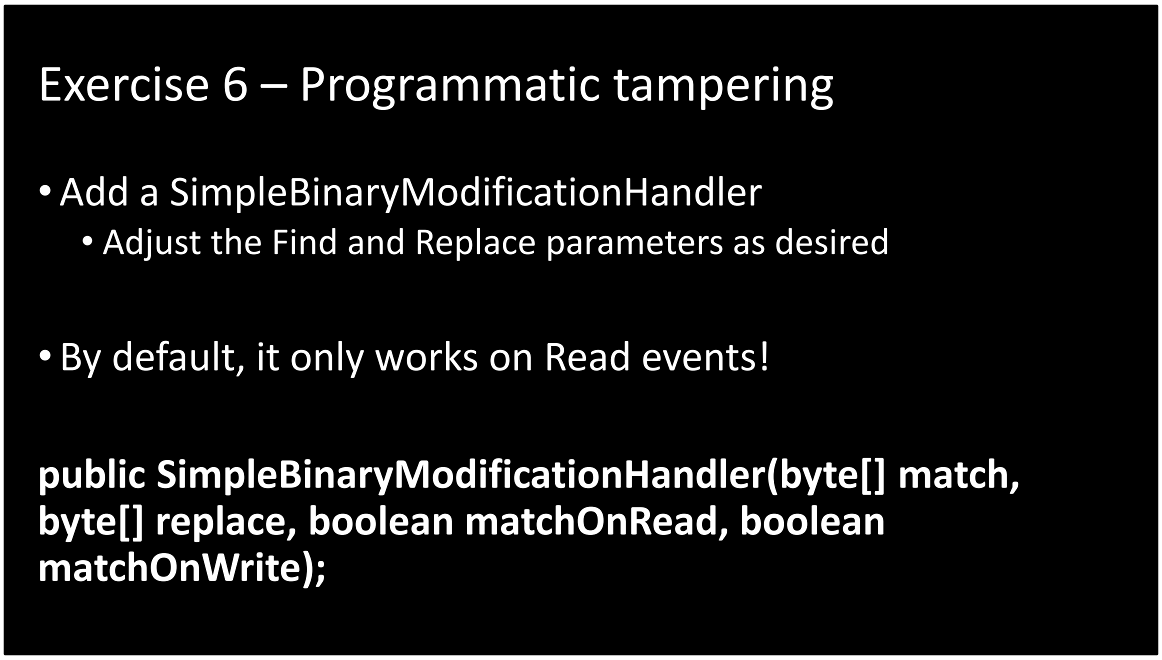
- Add a "Handler" node to the graph, then change the class to
  - io.netty.handler.codec.json.JsonObjectDecoder
- How many messages do you see from the client now?

Where do you think it should be added? After the SocksInitializer, as well as before the Sink. Is the second one strictly required? Not really, because the server happens to write all its responses in a single buffer. Are we likely to fail unexpectedly if we skip this? PROBABLY! Rather be sure!

Note that, while there is already a JsonObjectDecoder node in the protocols palette, you can use this method to add ANY netty handler to the graph, so long as you know the class name!

Exercise 5 – Intercepting and tampering

- Use the Intercept checkbox to suspend future connections
- Click on the connection to view pending events
- Select the latest event to allow more traffic until you see a “Read” event
- Select and modify
- Tampering with raw bytes is not much fun!

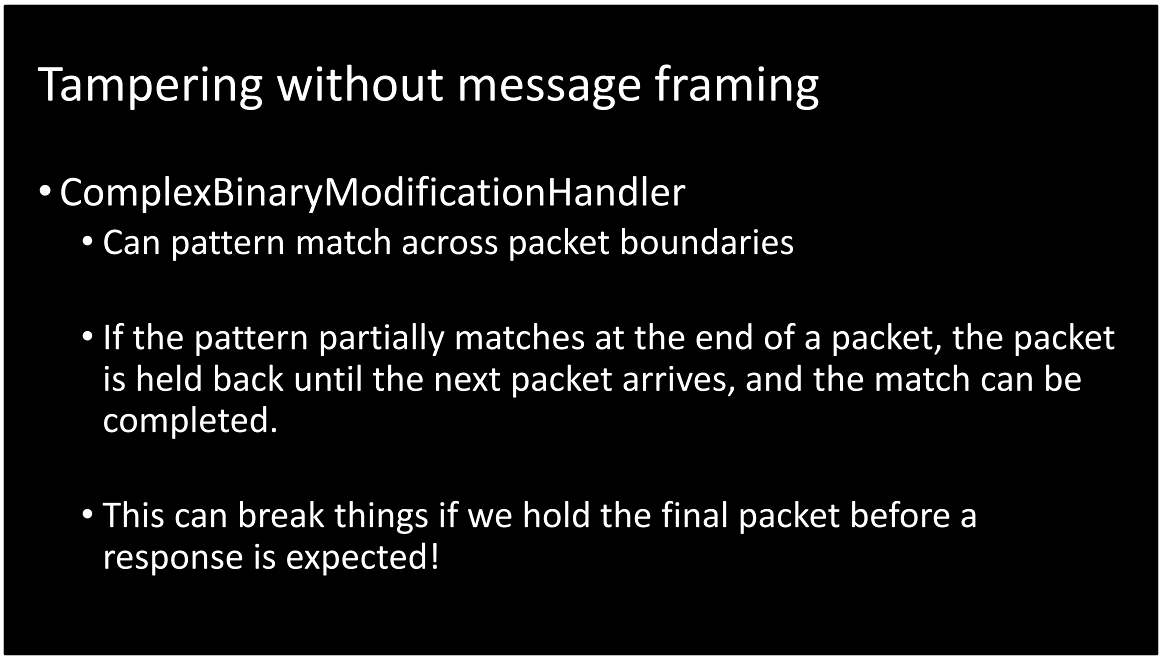


Exercise 6 – Programmatic tampering

- Add a SimpleBinaryModificationHandler
  - Adjust the Find and Replace parameters as desired

- By default, it only works on Read events!

public SimpleBinaryModificationHandler(byte[] match, byte[] replace, boolean matchOnRead, boolean matchOnWrite);



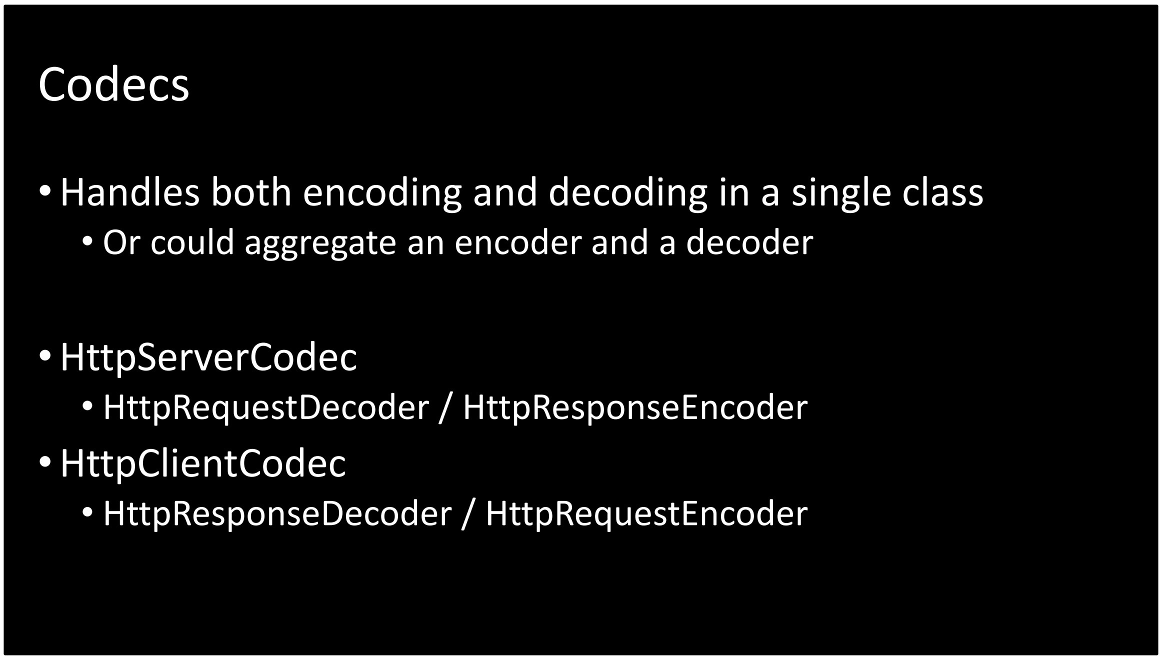
Tampering without message framing

- ComplexBinaryModificationHandler
  - Can pattern match across packet boundaries
  - If the pattern partially matches at the end of a packet, the packet is held back until the next packet arrives, and the match can be completed.
  - This can break things if we hold the final packet before a response is expected!

Decoders and Encoders

- Decoders convert from ByteBuf to Object
- Encoders convert from Object to ByteBuf
- Or could convert from one Class to another Class

- E.g. HttpRequestDecoder vs HttpRequestEncoder



Codecs

- Handles both encoding and decoding in a single class
  - Or could aggregate an encoder and a decoder

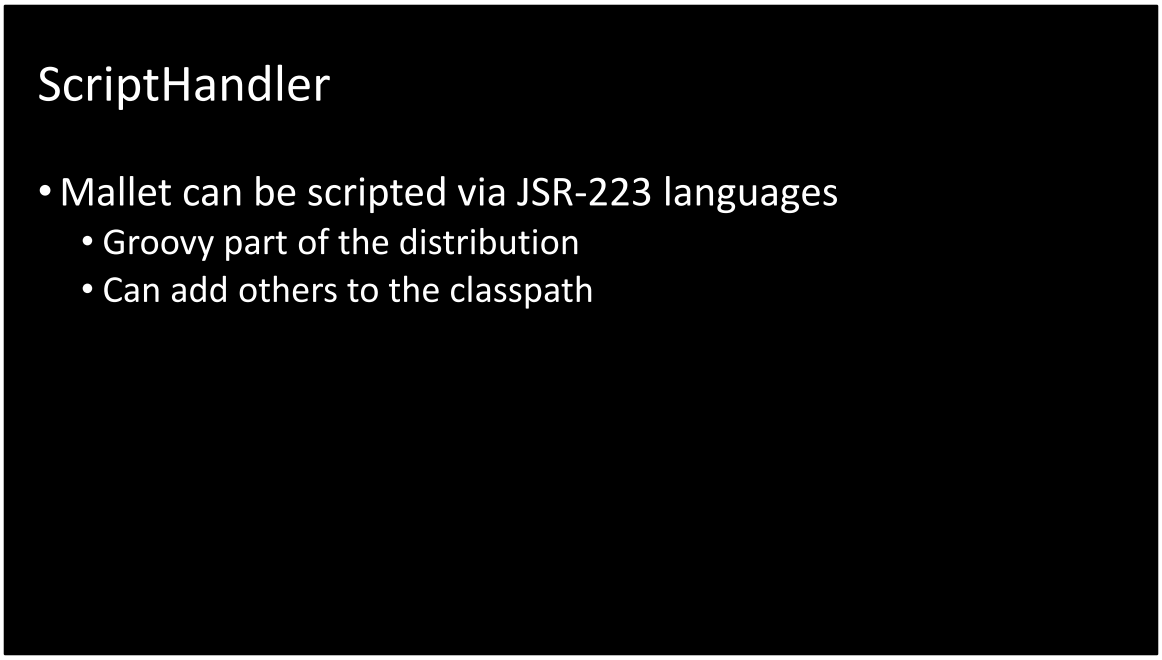
- HttpServerCodec
  - HttpRequestDecoder / HttpResponseEncoder
- HttpClientCodec
  - HttpResponseDecoder / HttpRequestEncoder

Can suggest one reason to link a HttpRequestEncoder and a HttpResponseDecoder in a single class?

Think HEAD requests. When the ResponseDecoder knows that the corresponding request was a HEAD, it should expect zero-byte content, even though the Content-Length header may say differently.

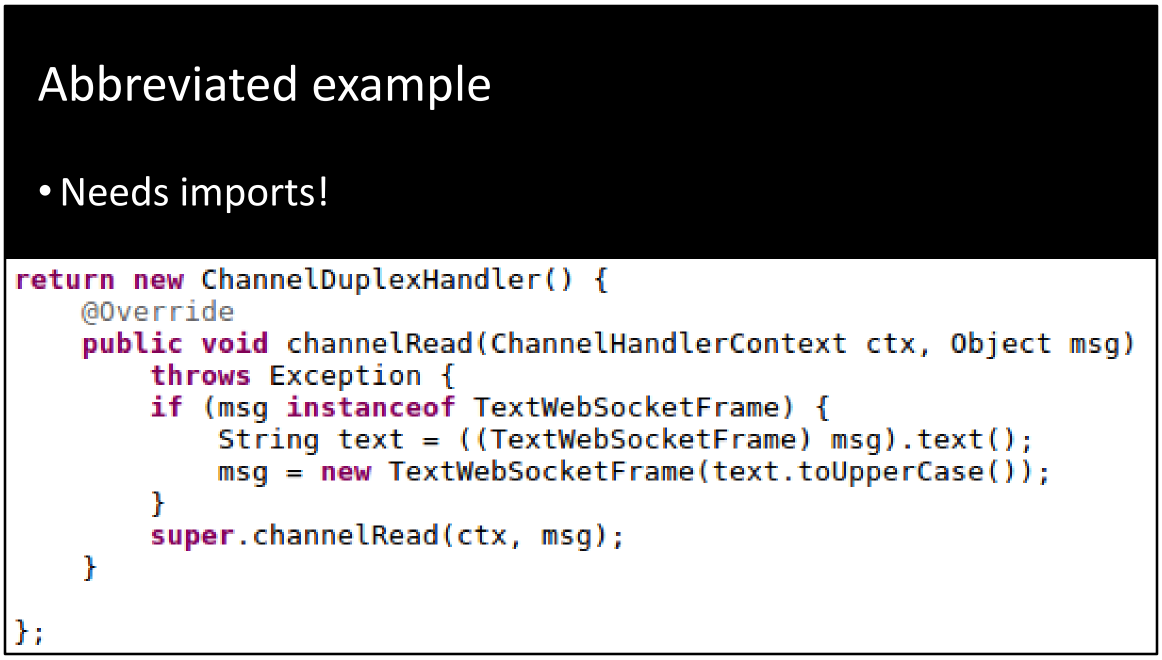
Exercise 7 – Working with Strings

- Add a StringDecoder / StringEncoder to the graph
  - Where?
- Intercept the messages



ScriptHandler

- Mallet can be scripted via JSR-223 languages
  - Groovy part of the distribution
  - Can add others to the classpath



Abbreviated example

- Needs imports!

```
return new ChannelDuplexHandler() {
    @Override
    public void channelRead(ChannelHandlerContext ctx, Object msg)
        throws Exception {
        if (msg instanceof TextWebSocketFrame) {
            String text = ((TextWebSocketFrame) msg).text();
            msg = new TextWebSocketFrame(text.toUpperCase());
        }
        super.channelRead(ctx, msg);
    }

};
```

Just an illustration of how little work it actually is to make some custom changes to a message, using a ScriptHandler

This makes the text in a TextWebSocketFrame uppercase.

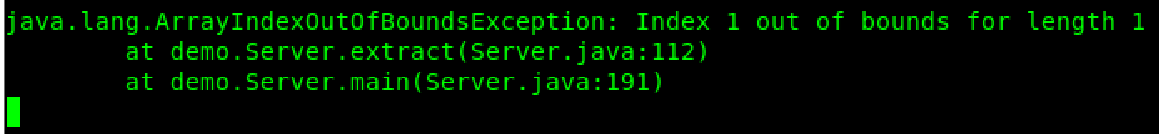
Exercise 8 – Decoding JSON to Object

- Load examples/json4.mxe
- Review both ScriptHandler nodes
- Generate messages
- Intercept and tamper

Importance of byte for byte matching

- Try removing the following line from the client ScriptHandler
  - objectMapper.enable(SerializationFeature.INDENT_OUTPUT);

- What happens? Why?



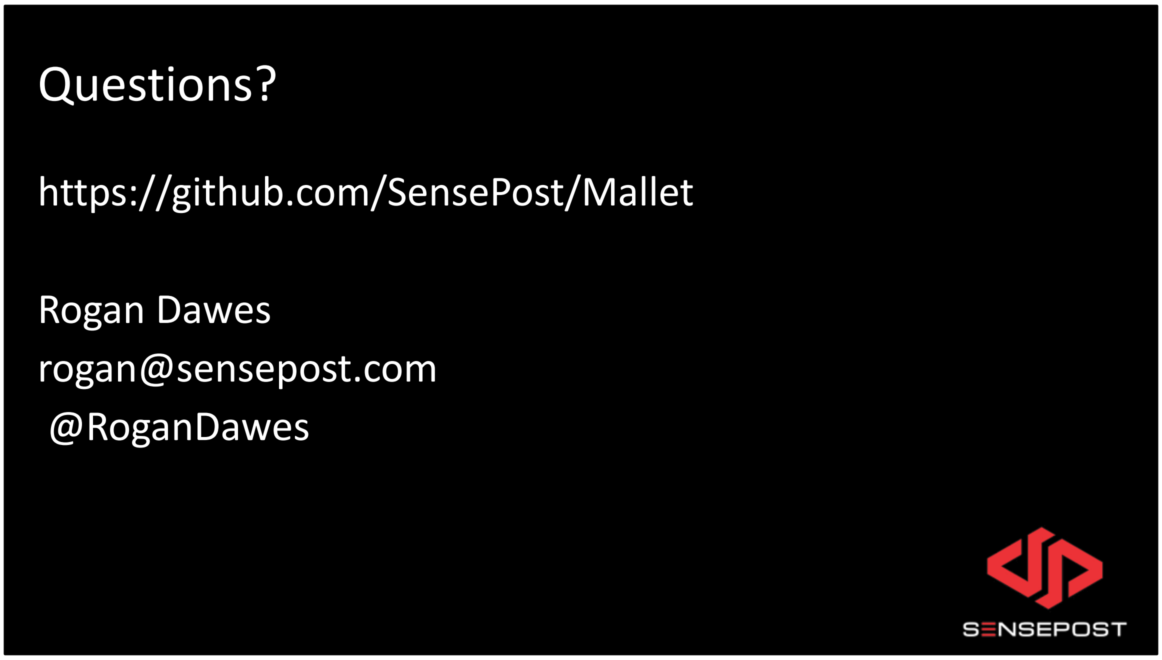
```
java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1
        at demo.Server.extract(Server.java:112)
        at demo.Server.main(Server.java:191)
```

Exercise 9 – Using LoggingHandler

- Select the Log tab, and enable logging
- Drag a LoggingHandler into the graph
- Generate a message

Exercise 10 – Custom modifications

- Using examples/json5.mxe, update the client ScriptHandler
  - Automatically update the value of the “name” field before it is sent to the server
  - Note that there are 2 client ScriptHandler nodes



Questions?

https://github.com/SensePost/Mallet

Rogan Dawes
rogan@sensepost.com
@RoganDawes

SENSEPOST