PREPARED BY
Luca Carettoni, Matasano Security
luca@matasano.com

AMF TESTING MADE EASY!

Monday, July 23, 2012

matasano
SECURITY

info@matasano.com
(888) 677-0666

39 West 14th St.
Suite 202
New York, NY
10011

53 W. Jackson
Suite 1850
Chicago, IL
60604

756 California St
Suite A
Mountain View, CA
94041

AMF TESTING MADE EASY!

Since its introduction in 2002, Action Message Format (AMF) has attracted the interest of developers and bug-hunters. Techniques and extensions for traditional web security tools have been developed to support this binary protocol. In spite of that, bug hunting on AMF-based applications is still a manual and time-consuming activity. Moreover, several new features of the latest specification, such as externalizable objects and variable length encoding schemes, limit the existing tools.

This research aimed at improving the current state of art, introducing a novel testing approach as well as a new tool named **Blazer**. Our automated gray-box testing technique allows security researchers to build custom AMF messages, generating dynamically objects from method signatures. The approach has been implemented in a Burp Suite plugin which currently supports Adobe BlazeDS, a well-known Java remoting technology. Using Blazer, testing AMF-based applications is easier and more robust. The tool consents to improve the coverage and the effectiveness of fuzzing efforts targeting complex applications.

This paper focuses on the newly introduced methodology and explain how to use the tool during gray-box testing of AMF-based applications. It is assumed that readers of this document have a basic knowledge and understanding of Adobe Flex and the AMF format.

The following summarizes the outline of this paper:

- **Technology overview**
  Brief overview of the technology context. Introduction to the AMF specification, Flex remoting and Adobe BlazeDS
- **State of art**
  Summary of the state of art, including existing tools and limitations of current techniques
- **Testing AMF-based applications**
  Description of a generic methodology for gray-box testing
- **Blazer**
  Introduction to Blazer, detailing core techniques and features
- **How to use Blazer**
  Two practical examples on how to use Blazer to test authorization and input validation flaws
- **Conclusion**
  Final considerations and possible future improvements

TECHNOLOGY OVERVIEW

The research outlined in this paper focuses on the latest AMF specification and Adobe BlazeDS/ LiveCycle Data Services, a widely used server-side Java implementation. Some of the technical details discussed in this paper are specific to Adobe's implementation. However, our approach and our tool can be applied on other Java server-side frameworks (e.g. Granite) and can be easily ported to other programming languages (RubyAMF, FluorineFX, amfPHP, etc.).

AMF Specification

As described in the Action Message Format (AMF) specification document [1], AMF is a compact binary format that is used to serialize ActionScript object graphs. ActionScript is an object -oriented programming language used within Adobe Flash applications. ActionScript object graphs are based on named properties in the form of key-value pairs.

AMF represents an efficient mechanism to save and retrieve the state of an application across sessions or to allow two endpoints to communicate through the exchange of strongly typed data. Nowadays, AMF is primarily used for accessing remote services and provide RPC (Remote Procedure Call) capabilities. Since its introduction, a standard Flex application can interact with complex back-end services and legacy applications. As a binary protocol with optimized objects compression, AMF allows faster data transfer comparing to traditional text-based protocols. Also, the native support of ActionScript objects serialization and deserialization improves the overall performance.

The first version of AMF, referred as AMF0, was officially released in 2002 with Macromedia Flash Player 6. Due to new data types and language features of ActionScript 3.0, Adobe released an updated version (AMF3) with Flash Player 9. Major improvements and changes include the possibility to send objects traits, string and other data type by reference. Also, AMF3 introduces variable length encoding schemes and support for "flash.utils.IExternalizable".

Adobe Flex Remoting

The combination of highly responsive client and server side components allow remote procedure invocation through Flex Remoting. This mechanism allows client-side applications to make asynchronous requests to remote services that process and return data. As long as the server-side technology has been implemented according to the Flex remoting specification, any Flex client or AIR application can communicate with remote services and inter-exchange data.

Adobe BlazeDS

In the effort of improving remoting and messaging capabilities of the Flex framework, Adobe released a open-source implementation named "BlazeDS" in addition to the commercial "Live-Cycle ES" version. BlazeDS core features include the RPC and the messaging services. In this context, remoting allows a Flex application to invoke directly methods of Java objects deployed within a traditional J2EE application server (e.g. Apache Tomcat). The server-side components of BlazeDS run in a standard web application container, in the form of Java classes, JAR libraries and XML-based configuration files (typically within WEB-INF/lib, WEB-INF/flex, WEB-INF/classes). As detailed in the Adobe BlazeDS Developer Guide [2], to send messages across the network, clients use the concept of channels. A channel encapsulates message formats, network protocols, and network behaviors to decouple them from services, destinations, and application code. A channel formats and translates messages into a network-specific form and delivers them to an endpoint on the server. Channels communicate with Java-based endpoints on the server. An endpoint unmarshals messages in a protocol-specific manner and then passes the data in generic Java form to the message broker. The message broker determines where to send messages and routes them to the appropriate service destination (Java class and method).

Supported AMF request/response types include:

- CommandMessage
- **RemotingMessage**
- AcknowledgeMessage
- ErrorMessage
- HTTPMessage / SOAPMessage

Each AMF remoting message can be uniquely identified by the following attributes:

- **endpoint** (e.g. *http://127.0.0.1:8080/myApp/messagebroker/amf*)
- **destination service** (e.g. *echoService*)
- **operation** and **parameters** (e.g. *String echo(String input)*)

Although software vulnerabilities in the BlazeDS framework have been uncovered in the past [3], this research focuses on software vulnerabilities at the application level only.

STATE OF ART

In recent years, Flex clients evolved from simple shiny user interfaces to large applications managing complex and sensitive data. As a result, we have observed an increase of interest for developers and breakers to evaluate the security posture of AMF-based applications. At OWASP AppSec 2007, Stefano di Paola presented one of the first research targeting the Flash framework [4]. During Blackhat USA 2008, a talk titled "Adobe Flex, Amf 3 And Blazeds An Assessment" [5] defined the technology stack as "an unexpectedly large attack surface" to emphasize the extend surface available to attackers. In 2009, Deblaze [6] was released by Jon Rose. The tool allows to enumerate remote methods through a combination of brute-forcing techniques and decompiling the client-side code. In 2010, Marcin Wielgoszewski presented one of the most comprehensive research targeting Flex-based applications [7]. In particular, he pointed out the need of building custom clients that would allow to define custom Java objects, in addition to primitive types. Starting from version 1.2.124 [8], the popular Burp Suite added support to visualize and tamper AMF requests and responses. Moreover, the Burp Scanner module was updated to place automatically attack payloads within string-based AMF values. Other tools have been also updated in order to support this technology, including Charles Proxy, WebScarab and Fiddler2 (AMFParser plugin). Particularly interesting is the idea implemented by Pinta [9], a cross-platform Adobe AIR application that allows to debug AMF calls.

A common methodology to perform black-box security testing against AMF applications include:

- Enumeration
    - Retrieving endpoints, destinations and operations from the traffic
    - Decompiling Flex application components
    - Brute-forcing endpoint, destination and operation names
- Traffic inspection and tampering
    - Using network packet analyzers
    - Using HTTP proxies

Although this approach can be used during black-box assessment with limited knowledge of the target, it has several disadvantages. Testing a large application is a time consuming task as it requires to invoke all application functionalities, observe the generated traffic and perform tampering. In case of operations using custom objects as arguments, reverse-engineering may be also required in order to build valid AMF messages. As a result, manual parameter tampering is usually limited to just few messages with primitive types and simple Java objects. Another drawback is related to “hidden” server-side functionalities that are impossible to uncover by decompiling the Flex client or observing the network traffic.

Although the current literature includes AMF testing techniques and tools, none of the previous research was focused on **coverage** and **automation**. In addition, most of the tools available have serious limitations while dealing with custom objects, which is a common practice in enterprise software. For instance, popular web security proxies do not properly handle complex AMF3 messages and they are not even able to display those custom objects thus tampering cannot be easily performed.

“Life is pain, highness. Anyone who tells you differently is selling something.” -W. Goldman

TESTING AMF-BASED APPLICATION

As outlined in the previous section, testing AMF-based applications is still a manual and time-consuming activity. If a security assessment is focused on coverage, the tester has to be able to generate valid AMF messages for all destinations. During several real-life engagements working with complex AMF-based applications, we have encountered enterprise-grade software with more than 500 remote invokable methods and more than 600 custom Java objects exchanged by the client and server.

Introducing a generic methodology for gray-box testing

Performing vulnerability research on such extended attack surface requires a new approach that would allow to overcome the limitations of current techniques. In particular, this research focuses on improving the coverage and the effectiveness of fuzzing efforts by designing and developing a tool, able to automatically generate valid AMF messages from operation signatures (e.g. Java methods in case of BlazeDS). Handling custom objects and other features of the AMF3 specification is crucial for auditing complex software and it was considered as requirement during the development of the tool.

The author assumes that the source code or the portable code (bytecode) for the application under scrutiny is available. During vulnerability research analysis, this is a realistic assumption as the tester has usually full control over the testing environment.

The ability of generating valid AMF messages for all endpoints, destinations and operations allows security researchers to cover the following test areas:

- **Authentication**
  Authentication and session management in BlazeDS applications is usually managed by the underline application server (e.g. Apache Tomcat) or web framework (e.g. Spring) via traditional session tokens. By generating valid messages and verifying that all operations require a valid cookie, except those specifically intended to be public, a tester can verify that no operations are exposed to unauthorized users.

- **Access control and authorization**
  By generating valid messages for all operations available to multiple low-privileges users as well as to administrators, a security researcher can detect horizontal and vertical escalation bugs. This allows the tester to uncover access control failures between users in the same tenancy and users between tenancies. In real-life situations, the tool should also be able to generate arbitrary operation parameters in order to test direct object references (DOR) bugs. For example, let's imagine a remote method for users management. The tool should be able to

build a custom Java object (e.g. "User") containing an attribute to identify the user id (e.g. "userID" as integer) and the user's roles (e.g. "Roles", another custom Java object).

- **Error handling and information disclosure**
  By sending properly formatted AMF messages with invalid content (e.g. an invalid "userID"), the tester can observe all possible server responses. This may lead to the discovery of Java stack traces or verbose error messages that may facilitate further attacks. It is important to note that we are targeting the application, thus the tool should not trigger errors or exceptions at the BlazeDS framework level.

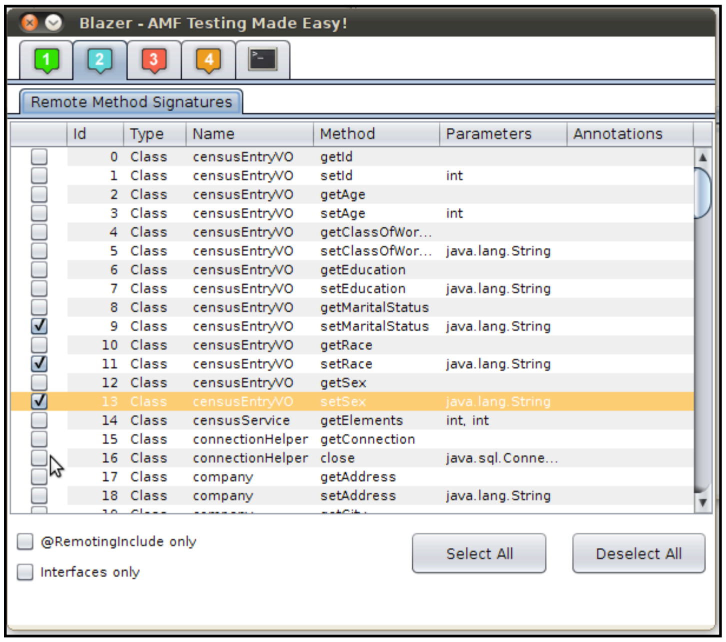
- **Input validation**
  Fuzzing with common attack patterns for traditional web application vulnerabilities (e.g. SQL injection, LDAP injection, etc.) allows to verify that all input validation failures result in input rejection or input sanitization. Thus, the tool should be able to tamper or mutate properly built AMF messages with malicious strings. Due to Java strong typing, injection would typically occur within String arguments. However, all data type are relevant. For instance, integer overflow bugs in Java application may affect the business logic and may have drastic consequences. Also, BlazeDS endpoints may expose native code through Java Native Interface (JNI) calls.

- **Output Encoding**
  Under some circumstances, it is important to verify that all untrusted data that are output need to be properly escaped for the specific application context. This task requires fuzzing and manual or semi-automatic evaluation of the resulting AMF responses. Although cross-site scripting and cross-site flashing affecting Flex applications have become less relevant thanks to the hardening introduced by recent Flash framework versions, these classes of vulnerabilities may still be relevant for particular use cases.

BLAZER

Considering the previously mentioned requirements, Blazer was designed and implemented to make AMF testing easy, and yet allows researchers to control fully the entire security testing process.

Blazer has been developed in Java as a Burp Suite plugin [10] and released under the GNU General Public License [11]. As it is highly integrated in a well-known testing suite, web security practitioners can start to use the tool with minimal setup in few seconds.



	Id	Type	Name	Method	Parameters	Annotations
☐	0	Class	censusEntryVO	getId		
☐	1	Class	censusEntryVO	setId	int	
☐	2	Class	censusEntryVO	getAge		
☐	3	Class	censusEntryVO	setAge	int	
☐	4	Class	censusEntryVO	getClassOfWor...		
☐	5	Class	censusEntryVO	setClassOfWor...	java.lang.String	
☐	6	Class	censusEntryVO	getEducation		
☐	7	Class	censusEntryVO	setEducation	java.lang.String	
☐	8	Class	censusEntryVO	getMaritalStatus		
✓	9	Class	censusEntryVO	setMaritalStatus	java.lang.String	
☐	10	Class	censusEntryVO	getRace		
✓	11	Class	censusEntryVO	setRace	java.lang.String	
☐	12	Class	censusEntryVO	getSex		
✓	13	Class	censusEntryVO	setSex	java.lang.String	
☐	14	Class	censusService	getElements	int, int	
☐	15	Class	connectionHelper	getConnection		
☐	16	Class	connectionHelper	close	java.sql.Conne...	
☐	17	Class	company	getAddress		
☐	18	Class	company	setAddress	java.lang.String	

☐ @RemotingInclude only

☐ Interfaces only

Select All   Deselect All

Blazer Burp Plugin Look and Feel

On Windows, it is possible to launch Burp and Blazer with the following command:

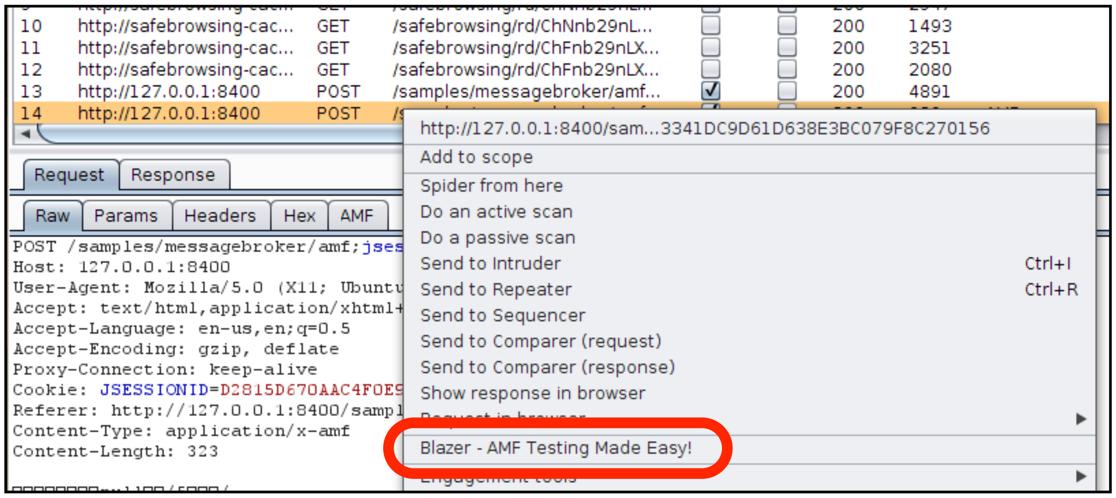
```
java -classpath burp.jar;Blazer.jar burp.StartBurp
```

On Linux and Mac OS X, use a colon character instead of the semi-colon as the classpath separator:

```
java -classpath burp.jar:Blazer.jar burp.StartBurp
```

Burp plugins are supported by both versions (free and professional) of the Burp Suite. All major operating systems (Windows, Mac, Linux) with standard Oracle JRE installed are supported by the current version of Blazer.

Upon launching Burp, it is possible to verify that Blazer was properly loaded by checking the "Alerts" notification tab. At this point, Blazer can be invoked by using a context menu available from within Burp tools.



```
10   http://safebrowsing-cac...   GET    /safebrowsing/rd/ChNnb29nL...          200   1493
11   http://safebrowsing-cac...   GET    /safebrowsing/rd/ChFnb29nLX...         200   3251
12   http://safebrowsing-cac...   GET    /safebrowsing/rd/ChFnb29nLX...         200   2080
13   http://127.0.0.1:8400        POST   /samples/messagebroker/amf...   ✓      200   4891
14   http://127.0.0.1:8400        POST
```

```
POST /samples/messagebroker/amf;jses
Host: 127.0.0.1:8400
User-Agent: Mozilla/5.0 (X11; Ubuntu
Accept: text/html,application/xhtml+
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive
Cookie: JSESSIONID=D2815D670AAC4F0E9
Referer: http://127.0.0.1:8400/sampl
Content-Type: application/x-amf
Content-Length: 323
```

Blazer Context Menu From Proxy History

Please note that the tool detects and verifies the presence of valid Flex's "RemotingMessage" objects within requests/responses captured by Burp.

Blazer can be fully configured and used from the plugin GUI. Traditional standard output and error streams provide further details during the automatic objects generation, messages creation and transmission.

At the top of Blazer's window, five tabs guide the user throughout the tool configuration:

1. Application Libraries
This allows to include all application artifacts, including application classes implementing the remote methods as well as application libraries. The current release of Blazer supports JAR files only. From the user's perspective, this is typically what is available in the application server webroot, under "/WEB-INF/".

2. **Remote Method Signatures**
Blazer automatically retrieves public method signatures from the application libraries. This tab allows to select all methods under scrutiny, filtering based on annotation or type.

3. **General Options and Data Pools**
Object generation and fuzzing can be precisely customized from the user interface. For in-

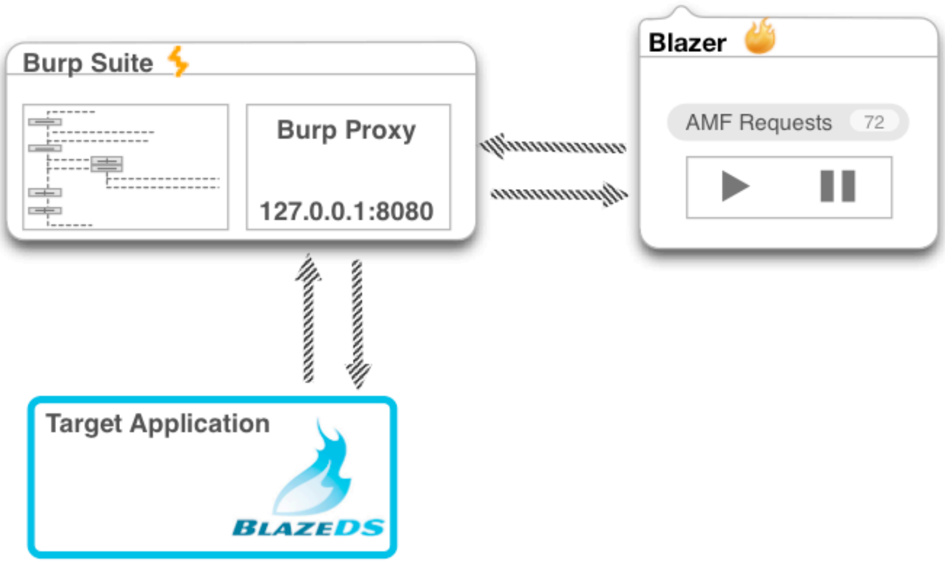
stance, users can select the number of threads, the total number of permutations for each iteration, common attack payloads and data pools containing “good” data.

4. **Status**
This tab allows to control the objects generation or fuzzing progress. It provides an overview of the total number of requests, permutations, overall time, time to finish and speed measured in requests/second. Also, this interface allows to pause, restore and stop current tasks.

5. **BeanShell**
Fuzzing is just the first step. Discovering and exploiting vulnerabilities requires troubleshooting and trial-and-error testing. In several cases, security professionals need an easy way to customize a specific AMF message and send it over the wire multiple times. Blazer embeds a BeanShell [12] console, automatically importing all required application libraries. Advanced users can build custom messages in few instructions by using Blazer internal methods.

Core techniques and architectural design

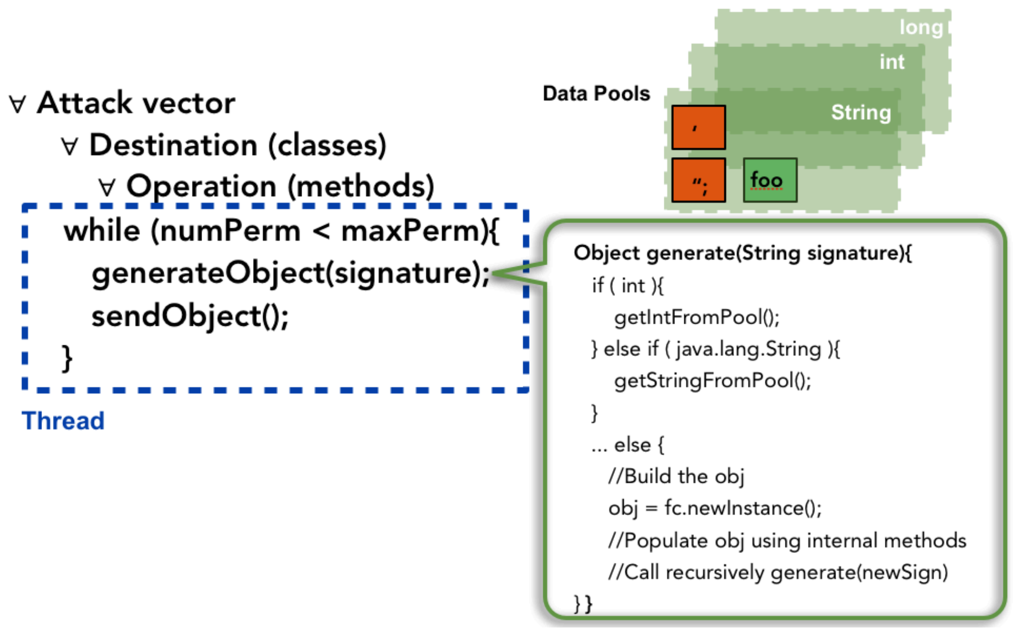
In a nutshell, Blazer is a custom AMF message generator with fuzzing capabilities. Access to the target application artifacts allows to generate automatically custom Java objects from method signatures via Java reflection and "best-fit" heuristics. Based on user-defined data pools, the tool automatically invokes objects' methods to populate each specific object instance with plausible data. This is internally referenced as "object generation" and allows to build valid AMF messages without preliminary knowledge of the application. In addition to that, fuzzing of string-based attributes for all encapsulated objects allow to easily pinpoint input-validation vulnerabilities. User-defined attack vector lists are employed to dynamically tamper string attributes of Java objects. Also, a specific setting allows to define the probability of using attack vectors versus "good" application data strings. In addition to that, users can also define the number of permutations during the object generation and fuzzing. This parameter is crucial for the entire analysis as it impacts directly the likelihood of generating semantically valid objects for the application under scrutiny. Incrementing the number of permutations increases the number of instances for a specific AMF method signature, populated with different data. Obviously, this parameter also increases the total number of requests and the overall time for the analysis.



Burp and Blazer Setup

By default, Blazer uses Burp Proxy to record requests and responses. This is particularly useful during the analysis of complex applications as the tester can benefit from the built-in tools available in Burp. Moreover, the "Filter by search term" functionality available in Burp Suite Professional allows to easily filter and remove irrelevant responses. Nevertheless, Blazer allows to specify an arbitrary proxy in order to use any third-party component to visualize and store the entire AMF communication.

A simplified heuristic used by Blazer is hereby presented:



```
∀ Attack vector
    ∀ Destination (classes)
        ∀ Operation (methods)
  while (numPerm < maxPerm){
      generateObject(signature);
      sendObject();
  }
```

```
Object generate(String signature){
  if ( int ){
      getIntFromPool();
  } else if ( java.lang.String ){
      getStringFromPool();
  }
  ... else {
    //Build the obj
    obj = fc.newInstance();
    //Populate obj using internal methods
    //Call recursively generate(newSign)
}}
```

Blazer Simplified Heuristic

For all interested readers, intrigued by the technical details of the object generation and the primitive type pools, it is suggested to read the source code of Blazer's core classes: *Object-Generator*, *MessageGenerator*, *TaskManager* and *MessageTask*.

At the architectural level, Blazer has three main components:

- a packet generator, based on Adobe AMF OpenSource libraries [13]
- an object generator that allows to build valid application objects using "best-fit" heuristics
- a lightweight fuzzing infrastructure that allows to generate attack vectors, insert payloads within objects, manage multiple threads and monitor the progress

HOW TO USE BLAZER

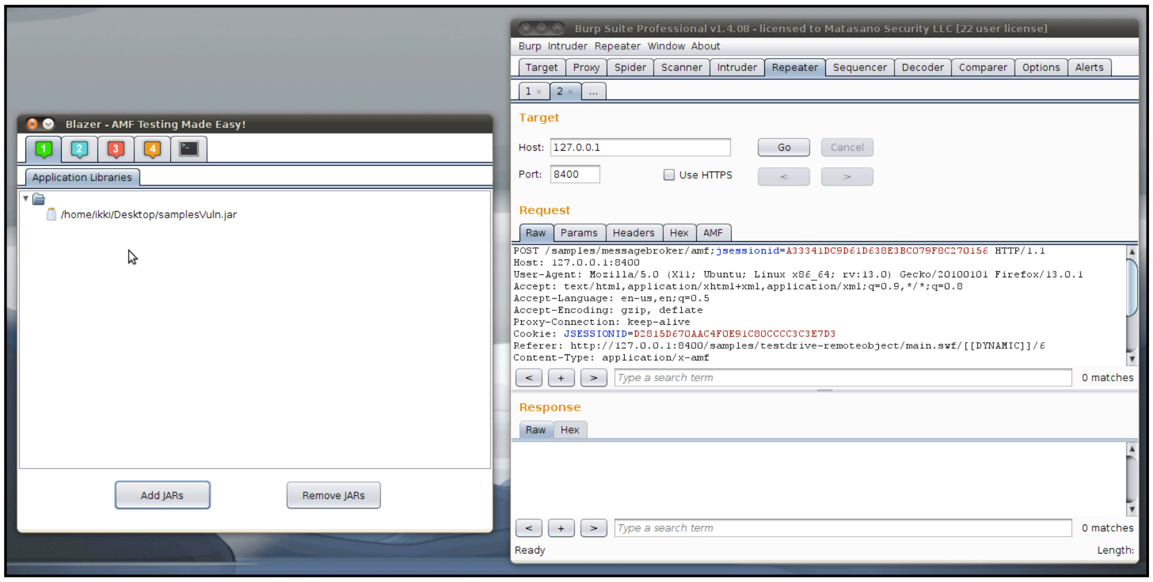
The following chapter illustrates the use of Blazer for two standard activities, typically performed during a security assessment. As usual, the best way to understand how the tool works is to experiment. Go and download Blazer from www.matasano.com !

Test case #1 - Testing Remote Methods Authorization

Frequently, it is important to evaluate the authorization mechanism and the level of privileges required for accessing remote methods. Let's imagine an application where session management and user permissions are based on standard session tokens. In this situation, Blazer can be extremely useful to evaluate whether a specific session has access to remote methods.

Testing with Blazer requires literally just few clicks. Upon instrumenting the browser to proxy all requests through Burp, intercept a valid AMF request for the remote BlazeDS-based application. Users can now start Blazer by selecting "Blazer - AMF Testing Made Easy!" from the context menu (e.g. right click from a Burp repeater tab).

Clicking "Add JARs", users can import all application resources.

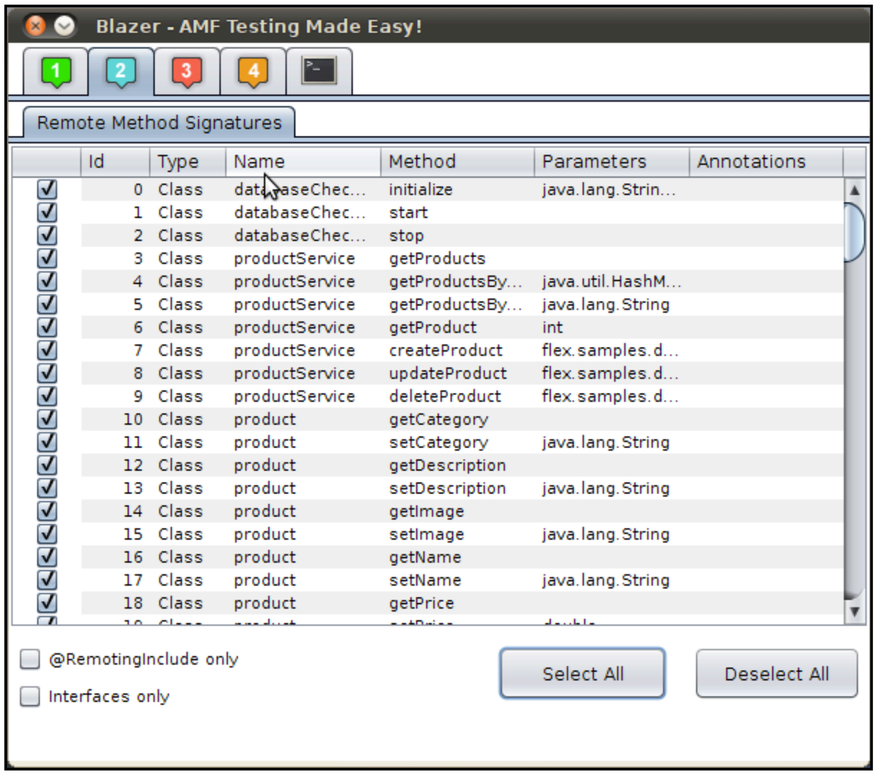

Blazer - AMF Testing Made Easy!
Application Libraries
/home/ikki/Desktop/samplesVuln.jar
Add JARs
Remove JARs
Burp Suite Professional v1.4.08 - licensed to Matasano Security LLC [22 user license]
Burp Intruder Repeater Window About
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Options Alerts
Target
Host: 127.0.0.1
Port: 8400
Use HTTPS
Go
Cancel
Request
Raw Params Headers Hex AMF
Type a search term
0 matches
Response
Raw Hex
Ready
Length:

```
POST /samples/messagebroker/amf;jsessionid=A33341DC9D61D638E3BC079F8C270156 HTTP/1.1
Host: 127.0.0.1:8400
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:13.0) Gecko/20100101 Firefox/13.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive
Cookie: JSESSIONID=D2815D670AAC4F0E91C80CCCC3C3E7D3
Referer: http://127.0.0.1:8400/samples/testdrive-remoteobject/main.swf/[[DYNAMIC]]/6
Content-Type: application/x-amf
```

Blazer - Step 1

In this example, "samplesVuln.jar" contains all classes including the exposed remote methods as well as all custom Java objects exchanged by the application.

Switching to tab "2", Blazer automatically unpacks and analyzes the libraries in order to display all public methods. Filtering can be used to identify specific methods. For instance, whenever annotation is used, users can quickly identify all exposed methods by displaying "@RemotingInclude" methods only.

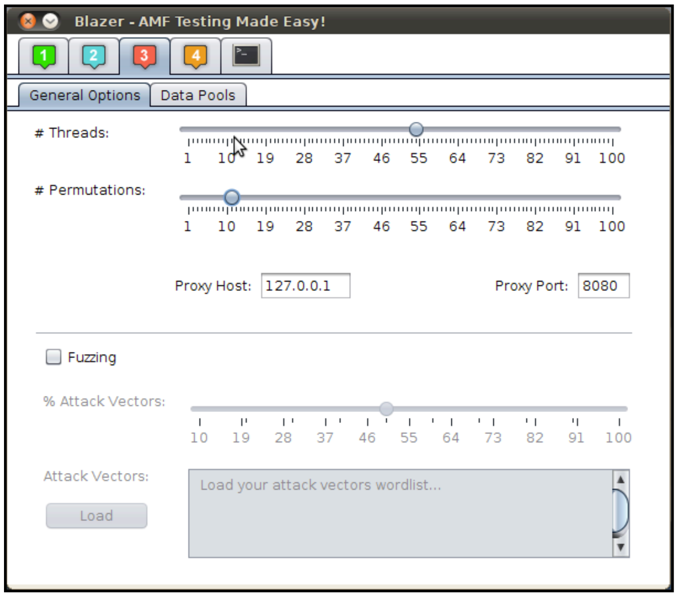


	Id	Type	Name	Method	Parameters	Annotations
✓	0	Class	databaseChec...	initialize	java.lang.Strin...	
✓	1	Class	databaseChec...	start		
✓	2	Class	databaseChec...	stop		
✓	3	Class	productService	getProducts		
✓	4	Class	productService	getProductsBy...	java.util.HashM...	
✓	5	Class	productService	getProductsBy...	java.lang.String	
✓	6	Class	productService	getProduct	int	
✓	7	Class	productService	createProduct	flex.samples.d...	
✓	8	Class	productService	updateProduct	flex.samples.d...	
✓	9	Class	productService	deleteProduct	flex.samples.d...	
✓	10	Class	product	getCategory		
✓	11	Class	product	setCategory	java.lang.String	
✓	12	Class	product	getDescription		
✓	13	Class	product	setDescription	java.lang.String	
✓	14	Class	product	getImage		
✓	15	Class	product	setImage	java.lang.String	
✓	16	Class	product	getName		
✓	17	Class	product	setName	java.lang.String	
✓	18	Class	product	getPrice		
✓	19	Class	product	setPrice	double	

Blazer - Step 2

For this test case we simply select all methods, even though some of those operations are not remotely invokable.

In step "3", users can configure multiple parameters used by Blazer. As we are testing access control mechanisms, we just want to generate valid AMF requests for all methods. Thus, we simply increase the number of threads and the number of permutations. The latter should be properly tuned, depending on the target application.

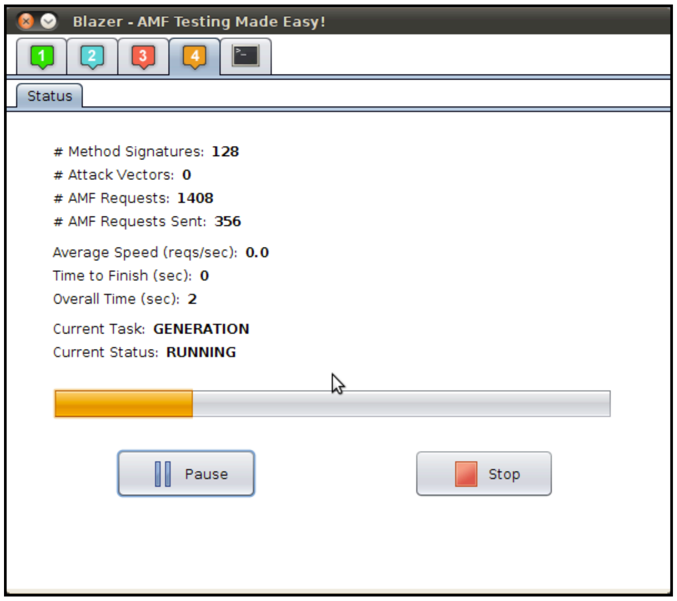


Proxy Host: 127.0.0.1

Proxy Port: 8080

Blazer - Step 3

Finally, users can launch the object generation by clicking on "Start". As all requests and responses are stored within Burp Proxy history, it is possible to identify easily accessible methods by filtering the raw responses.



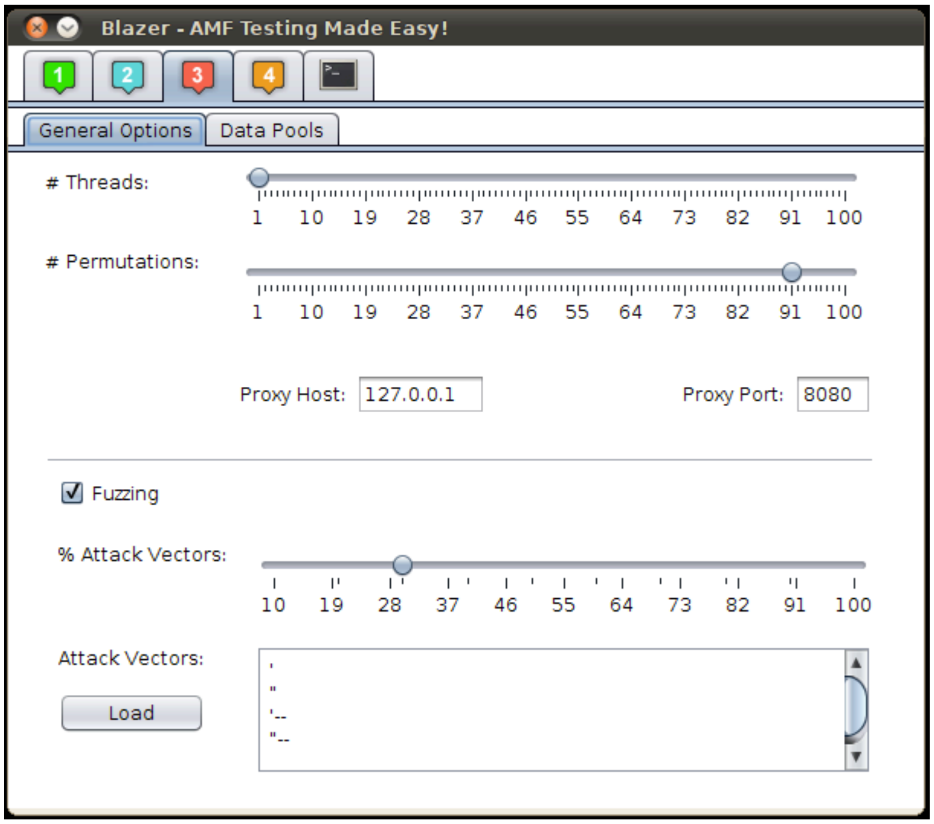
```
# Method Signatures: 128
# Attack Vectors: 0
# AMF Requests: 1408
# AMF Requests Sent: 356

Average Speed (reqs/sec): 0.0
Time to Finish (sec): 0
Overall Time (sec): 2

Current Task: GENERATION
Current Status: RUNNING
```

Blazer - Step 4

For example, it is possible to search for specific keywords (e.g. "No destination with id") and remove all invalid destinations. In just few minutes, users can identify all exposed methods for a specific user session. Changing the session token and repeating previous steps allows to test access control for different users and different set of privileges.

Test case #2 - Testing SQL Injection

The following demonstrates how to use Blazer in order to detect a common web application vulnerability: SQL Injection. Detecting input validation vulnerabilities within AMF messages re-quire to build properly formatted AMF requests containing common attack patterns.

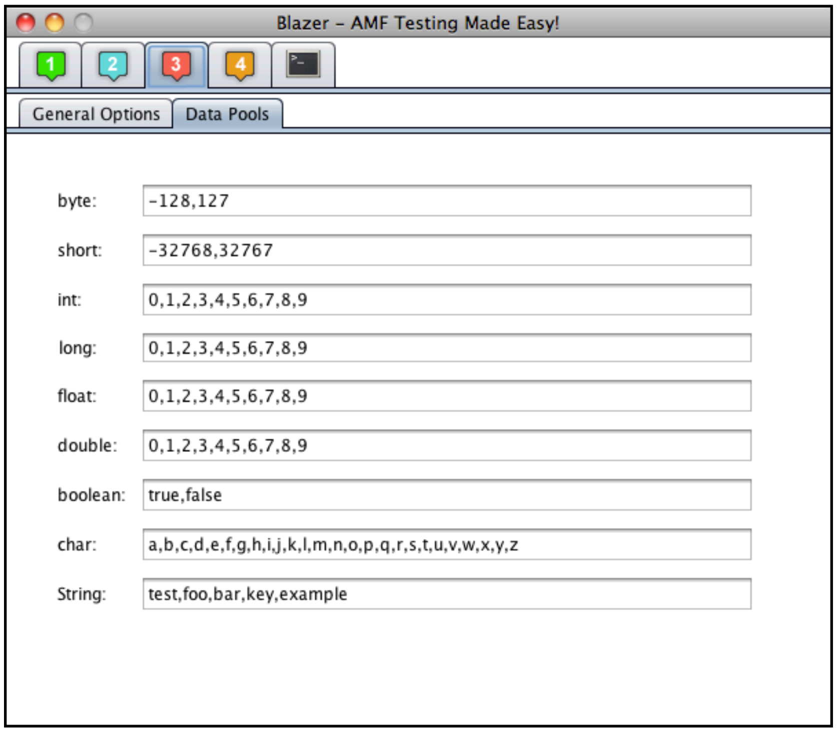


Threads: 1
Permutations: 91
Proxy Host: 127.0.0.1
Proxy Port: 8080
[✓] Fuzzing
% Attack Vectors: 28
Attack Vectors:

```
'
"
'--
"--
```

Tuning Blazer Configuration

Blazer can be easily configured with a custom attack vectors wordlist by loading it from a file or just typing in common attack patterns. In addition, users can control the mechanism used by Blazer to build objects with string attributes. This tuning allows to unbalance "good" and "bad" user-supplied input. In this example, Blazer will send "good data" 70% of the times and just 30% of the strings will contain actual attack vectors.



byte: -128,127

short: -32768,32767

int: 0,1,2,3,4,5,6,7,8,9

long: 0,1,2,3,4,5,6,7,8,9

float: 0,1,2,3,4,5,6,7,8,9

double: 0,1,2,3,4,5,6,7,8,9

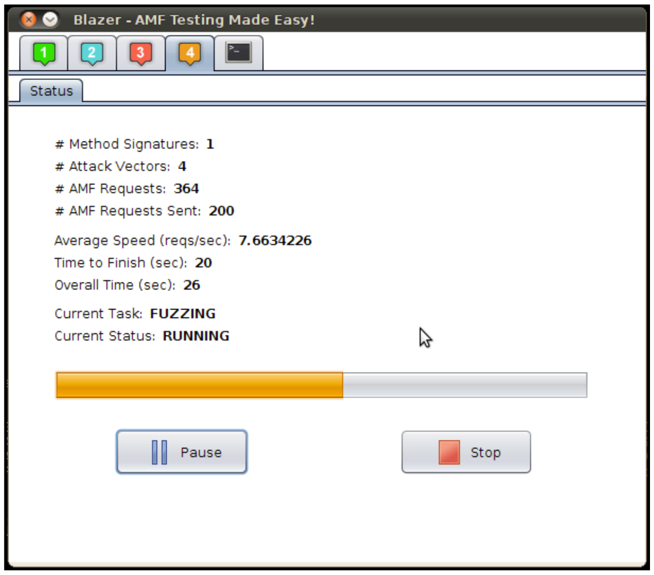
boolean: true,false

char: a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z

String: test,foo,bar,key,example

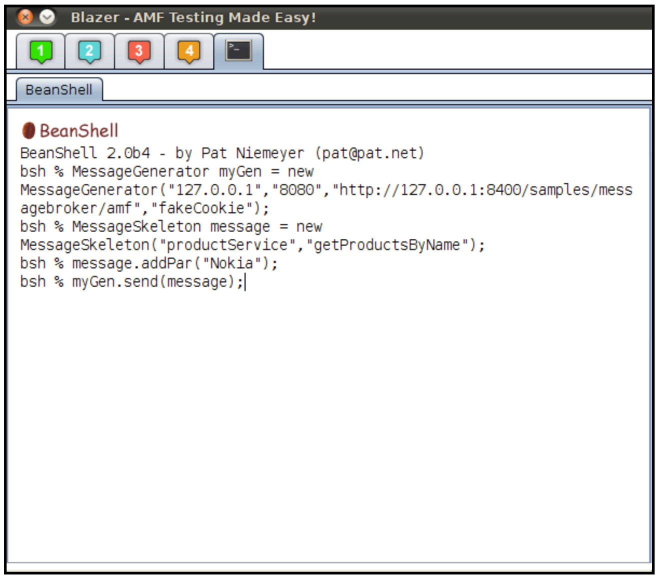
Data Pools Configuration

The "Data Pools" tab allows to define input that is considered by the application as "good" input and can be used by Blazer to build syntactically and semantically valid objects. By default, pools include all Java primitive types with predefined values. In-depth fuzzing requires to build valid AMF messages with attack vectors for detecting input validation vulnerabilities in strategic attributes only. By carefully tuning Blazer's configuration, users can achieve good results during the automatic objects generation.



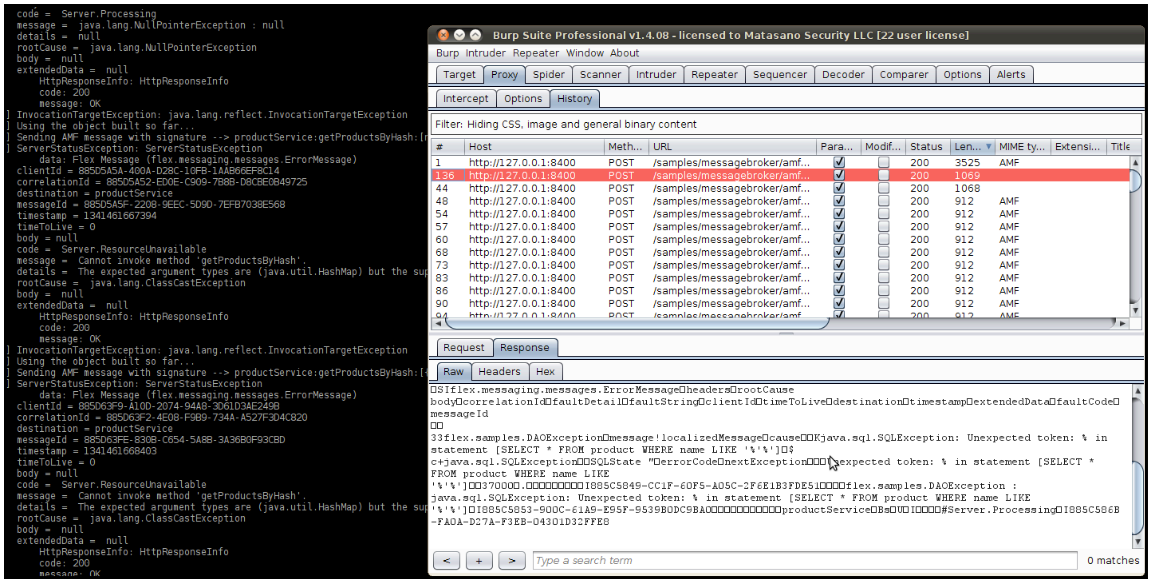
```
# Method Signatures: 1
# Attack Vectors: 4
# AMF Requests: 364
# AMF Requests Sent: 200

Average Speed (reqs/sec): 7.6634226
Time to Finish (sec): 20
Overall Time (sec): 26

Current Task: FUZZING
Current Status: RUNNING
```

Automatic and Manual Testing (BeanShell) with Blazer



```
BeanShell 2.0b4 - by Pat Niemeyer (pat@pat.net)
bsh % MessageGenerator myGen = new
MessageGenerator("127.0.0.1","8080","http://127.0.0.1:8400/samples/mess
agebroker/amf","fakeCookie");
bsh % MessageSkeleton message = new
MessageSkeleton("productService","getProductsByName");
bsh % message.addPar("Nokia");
bsh % myGen.send(message);
```

As in the previous test case, users can finally review all AMF requests and responses from within Burp. Sorting and filtering allows security testers to identify traditional error messages, typically associated with SQL injection vulnerabilities. Advanced users can also select the "console" tab in order to invoke BeanShell and programmatically use Blazer's inner classes to customize their exploit and create powerful proof-of-concepts.



```
  code =  Server.Processing
  message =  java.lang.NullPointerException : null
  details =  null
  rootCause =  java.lang.NullPointerException
  body =  null
  extendedData =  null
      HttpResponseInfo: HttpResponseInfo
      code: 200
      message: OK
] InvocationTargetException: java.lang.reflect.InvocationTargetException
] Using the object built so far...
] Sending AMF message with signature --> productService:getProductsByHash:[
] ServerStatusException: ServerStatusException
      data: Flex Message (flex.messaging.messages.ErrorMessage)
  clientId = 885D5A5A-400A-D28C-10FB-1AAB66EF8C14
  correlationId = 885D5A52-ED0E-C909-7B8B-D8CBE0B49725
  destination = productService
  messageId = 885D5A5F-2208-9EEC-5D9D-7EFB7038E568
  timestamp = 1341461667394
  timeToLive = 0
  body = null
  code =  Server.ResourceUnavailable
  message =  Cannot invoke method 'getProductsByHash'.
  details =  The expected argument types are (java.util.HashMap) but the su
  rootCause =  java.lang.ClassCastException
  body =  null
  extendedData =  null
      HttpResponseInfo: HttpResponseInfo
      code: 200
      message: OK
] InvocationTargetException: java.lang.reflect.InvocationTargetException
] Using the object built so far...
] Sending AMF message with signature --> productService:getProductsByHash:[
] ServerStatusException: ServerStatusException
      data: Flex Message (flex.messaging.messages.ErrorMessage)
  clientId = 885D63F9-A10D-2074-94A8-3D61D3AE249B
  correlationId = 885D63F2-4E08-F9B9-734A-A527F3D4C820
  destination = productService
  messageId = 885D63FE-830B-C654-5A8B-3A36B0F93CBD
  timestamp = 1341461668403
  timeToLive = 0
  body = null
  code =  Server.ResourceUnavailable
  message =  Cannot invoke method 'getProductsByHash'.
  details =  The expected argument types are (java.util.HashMap) but the su
  rootCause =  java.lang.ClassCastException
  body =  null
  extendedData =  null
      HttpResponseInfo: HttpResponseInfo
      code: 200
      message: OK
```

Burp Suite Professional v1.4.08 - licensed to Matasano Security LLC [22 user license]
Burp Intruder Repeater Window About
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Options Alerts
Intercept Options History

Filter: Hiding CSS, image and general binary content

#	Host	Meth...	URL	Para...	Modif...	Status	Len...	MIME ty...	Extensi...	Title
1	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	3525	AMF		
136	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	1069			
44	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	1068			
48	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	912	AMF		
54	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	912	AMF		
57	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	912	AMF		
60	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	912	AMF		
68	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	912	AMF		
73	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	912	AMF		
83	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	912	AMF		
86	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	912	AMF		
90	http://127.0.0.1:8400	POST	/samples/messagebroker/amf...	✓	☐	200	912	AMF		
94	http://127.0.0.1:8400	POST	/samples/messagebroker/amf	✓	☐	200	912	AMF		

```
□SIflex.messaging.messages.ErrorMessage□headers□rootCause
body□correlationId□faultDetail□faultString□clientId□timeToLive□destination□timestamp□extendedData□faultCode□
messageId
□□
33flex.samples.DAOException□message!localizedMessage□cause□□Kjava.sql.SQLException: Unexpected token: % in
statement [SELECT * FROM product WHERE name LIKE '%'%']□$
c+java.sql.SQLException□□SQLState "□errorCode□nextException□□□Unexpected token: % in statement [SELECT *
FROM product WHERE name LIKE
'%'%']□□37000□.□□□□□□□□□I885C5849-CC1F-60F5-A05C-2F6E1B3FDE51□□□□flex.samples.DAOException :
java.sql.SQLException: Unexpected token: % in statement [SELECT * FROM product WHERE name LIKE
'%'%']□I885C5853-900C-61A9-E95F-9539B0DC9BA0□□□□□□□□□□□productService□Bs□U□I□□□□#Server.Processing□I885C586B
-FA0A-D27A-F3EB-04301D32FFE8
```

Detecting SQLi in AMF endpoints with Burp and Blazer

CONCLUSION

Our newly introduced approach has been proven to increase the coverage and the effectiveness of AMF security testing. Blazer has been used to find real-life vulnerabilities including direct object reference bugs, authentication flaws, business logic abuses, SQL injections and other critical bugs during several complex engagements.

Unlike previous research, our effort was focused on coverage and automation. We strongly believe that our approach reshapes the concept of AMF fuzzing. Security testers can now generate valid AMF messages in just a few clicks. Also, Blazer fuzzing capabilities allow to perform tampering and to detect vulnerabilities in a very time-efficient manner. Finally, taking advantage of Blazer internal methods, a user can easily construct custom AMF messages from within the tool with BeanShell. Also, Java, JRuby and other programming languages can import Blazer as a library in order to quickly build complex proof-of-concepts.

Although the current tool has been designed specifically for BlazeDS-based application, Blazer can be used on other Java server-side frameworks and the approach can be easily ported to other programming languages and technologies.

Upcoming tool improvements include the possibility to import recursively source code and classes from specified directories. Sand-boxing via Java security manager configurations will be also provided to avoid dangerous methods execution while generating custom objects.

In future, the following features may be implemented:

- In addition to BeanShell, it will be also useful to extend the "console" tab in order to support jRuby or other scripting languages
- Having an embedded utility for visualizing complex AMF messages will be also beneficial while monitoring the objects generation process
- Auto-selection of remote method signatures from entries already present in Burp History may further speed-up the testing activity
- Auto-save of all Blazer's configuration parameters would allow to easily tamper HTTP requests (e.g. changing the session token), without having to reconfigure Blazer

REFERENCES

1. AMF 3 Specification, Adobe Systems Inc.
   http://download.macromedia.com/pub/labs/amf/amf3_spec_121207.pdf

2. Adobe BlazeDS Developer Guide, Adobe Systems Inc.
   http://livedocs.adobe.com/blazeds/1/blazeds_devguide/index.html

3. AMF arbitrary code execution, Wouter Coekaerts
   http://wouter.coekaerts.be/2011/amf-arbitrary-code-execution

4. Testing Flash Applications, Stefano di Paola
   http://www.owasp.org/images/8/8c/OWASPAppSec2007Milan_TestingFlashApplications.ppt

5. Adobe Flex, AMF 3 and BlazeDS: An Assessment, Jacob Karlson and Kevin Stadmeyer
   http://www.blackhat.com/presentations/bh-usa-08/Carlson_Stadmeyer/BlackHat-Flex-Carlson_Stadmeyer_vSubmit1.pdf

6. Deblaze, Jon Rose
   http://deblaze-tool.appspot.com/

7. Pentesting Adobe Flex Applications, Marcin Wielgoszewski
   http://blog.gdssecurity.com/storage/presentations/OWASP_NYNJMetro_Pentesting_Flex.pdf

8. Burp Suite v1.2.14 Release Note, PortSwigger Ltd.
   http://releases.portswigger.net/2009/08/v1214.html

9. AMF service test and debug utility, Pinta project members
   http://code.google.com/p/pinta/

10. Burp Suite Extender, PortSwigger Ltd.
    http://portswigger.net/burp/extender/

11. Licenses, Free Software Foundation, Inc.
    http://www.gnu.org/licenses/

12.BeanShell, BeanShell developers
    http://www.beanshell.org/

13. BlazeDS Java AMF Client, Adobe Systems Inc.
    http://sourceforge.net/adobe/blazeds/wiki/Java%20AMF%20Client/

Corporate Office:
39 W. 14th Street
Suite 202
New York, NY 10011

Midwest:
53 W. Jackson Blvd
Suite 1850
Chicago, IL 60604

West Coast:
756 California St
Suite A
Mountain View, CA
94041

www.matasano.com
info@matasano.com
888-677-0666

ABOUT THE AUTHOR

Luca Carettoni is a senior security consultant for Matasano Security with over 7 years experience as a computer security researcher. His professional expertise includes black box security testing, web application security, vulnerability research and source code analysis.

Prior to Matasano, Luca worked at The Royal Bank of Scotland as a penetration testing specialist where he performed security audits against several online banking systems worldwide. In the past years, Luca has been an active participant in the security community and a member of the Open Web Application Security Project (OWASP). Luca holds a Master's Degree in Computer Engineering from the Politecnico di Milano university.

ABOUT MATASANO

Matasano is an independent security research and development firm. We work with vendors and enterprises to pinpoint and eradicate security flaws, using penetration testing, reverse engineering, and source code review.

Since 1994, Matasano researchers have had founding roles in the first security research labs, discovered new classes of vulnerabilities, secured operating systems, and shipped large software projects. We’ve been behind some of the first breaks in SAN technology, virtualization, and financial protocols.

Our work has been featured in Network World, eWeek, Forbes, Macworld, Wired, and the Washington Post, and at conferences ranging from Black Hat to Gartner. Our practice focus areas include storage, virtualization, financial, middleware, software protection, kernel security, AJAX/Web 2.0, enterprise technology, security product testing, 802.11 and authentication protocols, RF, VOIP and telephony, Windows Vista, Mac OS X, and embedded platforms.