Automated Security Testing of Oracle Forms Applications

GIAC (GWAPT) Gold Certification

Author: Bálint Varga-Perke, vpbalint@silentsignal.hu
Advisor: Rick Wanner
Accepted: May 15, 2015

Abstract

Oracle Forms, a component of Oracle Fusion Middleware is a technology to efficiently build browser-based enterprise applications. In order to support multiple transport methods Forms has its own binary message format that is meant to provide serialization and additional security for the platform. Unfortunately this proprietary format renders conventional security testing tools unusable. Reverse engineering methods will be employed to reveal the format of the protocol messages and to analyze the cryptographic protections in use. It will be shown that the proprietary encryption and key exchange schemes can be attacked in multiple ways. New tools will be presented which can be used to exploit these weaknesses and allow existing security testing software to be used against Oracle Forms applications. Based on the observations deployment best practices will also be described to help mitigate the discussed problems.

1. Introduction

To keep up with the increasing rate of web application attacks (Imperva, 2014) a wide variety of automated security testing tools have been developed (OWASP, 2014). These tools rely on interface and protocol standards so they can be used against various applications.

But what if some of these well-known interfaces are replaced with proprietary alternatives? How does such a change affect the work of a penetration tester and the opportunities of an attacker? This paper explores these questions through the example of Oracle Forms.

Oracle Forms, a component of Oracle Fusion Middleware is a technology to rapidly develop browser-based enterprise applications (Oracle Corporation). The framework is implemented in Java, client-side components run as applets inside the browser. Instead of using standard HTTP requests Oracle Forms implements a proprietary binary protocol that can be used over raw TCP channels or embedded in HTTP. The protocol is encrypted which renders conventional testing tools useless, since they are unable to read or modify messages in plain text.

Oracle Forms is based on the Java applet technology to display client-side content. Applets used to provide attractive features long before they were incorporated in web browsers (Schuh, 2013). Today the majority of these once unique features became part of web browsers (Microsoft Corporation, 2009.). Also the popularity of applets is declining because of security concerns (Mimoso, 2013.). Despite these trends Oracle is dedicated to continue to support Forms (Oracle Corporation, 2012.). Moreover, migration to the more modern Application Development Framework (Oracle Corporation, 2014.) is not supported by the vendor (Oracle Corporation, 2012.).

To support the testing of Oracle Forms applications this paper gives detailed description of Oracle's propriatery protocol. Multiple vulnerabilities of the encryption scheme are shown which can be exploited by passive and active network attackers. Based on the analysis the possibilities and challenges of automated testing are assessed. Proof of concept software is presented that demonstrate attacks, and can be used directly or as a base of more advanced tools for security testing Oracle Forms applications.

At the time of writing the latest version of Oracle Forms is 11g, all findings are based on this version. Source code of the presented tools are available at http://github.com/v-p-b/oracle_forms/.

2. Security Analysis of Oracle Forms

2.1. The Oracle Forms Protocol

To assess the security of a web application framework (and the applications built on it) detailed understanding of its communication protocol is required. Oracle Forms is written in Java so the byte-code of its components can be easily decompiled. This way a "white box" approach (Girish Janardhanudu, 2005) can be taken for protocol analysis.

The main source of information was the decompiled byte-code of the `frmall.jar` archive. The `frmall.jar` archive contains the framework code that is to be loaded client side when using Oracle Forms. The code snippets in the following subsections are taken from this archive. The decompilation was done using the JAD decompiler (Java Decompilers Online, 2015) via the following command (Linux BASH):

```
unzip frmall.jar; find oracle/ -name '*.class' -execdir jad {} \;
```

Among the decompiled pseudo-code the encrypted communication was intercepted and analyzed with Burp Suite Professional (PortSwigger Ltd.).

2.1.1. Encryption and Key Exchange

Since Oracle Forms supports plain text communication channels the framework provides additional encryption in the application layer (Oracle Corporation, 2009). The documentation provided by the vendor doesn't specify the level of protection this encryption layer is meant to provide. The FAQ only states that the encryption scheme is "not as strong as the SSL standard" (Oracle Corporation, 2009). An older version of the documentation states that 40-bit RC4 encryption is used when communicating over plain text channel (Oracle Corporation, 2009). RC4 is a symmetric stream cipher designed by Ron Rivest in 1987 (Ron L. Rivest, 2014) and later became part of important protocols such as the SSL and TLS protocol families (IETF, 2008.). Later the cipher was found to be vulnerable against multiple cryptographic attacks. An overview of these attacks is presented in section 2.2.1.

Aside from the cryptographic strength of the employed cipher, the security of a protocol is also dependent on how the algorithm is used. The *oracle.forms.net* package contains the classes related to network communication. Communication is supported over raw TCP sockets or wrapped inside HTTP or HTTPS protocols.

Connection classes (*HTTPConnection*, *SocketConnection*) are responsible for establishing communication channels between the Oracle Forms client and server. This task includes cryptographic negotiations and proxy support. Key exchange is performed by these classes in the following way (taken from the *HTTPConnection* class):

```
dataoutputstream.writeInt(NEG_SEND); // NEG_SEND = 0x47446179
int i;
dataoutputstream.writeInt(i = (new Random()).nextInt());
dataoutputstream.flush();
int k = datainputstream.readInt();
int j = datainputstream.readInt();
if(k == NEG_RESPONSE) // NEG_RESPONSE = 0x4d617465
{
    byte abyte0[] = new byte[5];
    abyte0[0] = (byte)(i >> 8);
    abyte0[1] = (byte)(j >> 4);
    abyte0[2] = -82;
    abyte0[3] = (byte)(i >> 16);
    abyte0[4] = (byte)(j >> 12);
    if(mUseNativeHTTP)
        mHNs.setEncryptKey(abyte0);
    else
        mHs.setEncryptKey(abyte0);
}
```

The client first sends the *NEG_SEND* constant (equivalent to the ASCII "GDay" string) to the server along with a random integer. Then it waits for the server to send the *NEG_RESPONSE* constant (equivalent to the ASCII "Mate" string) and another integer.

The two integers are then used to construct the 5 byte (40-bit) long *abyte* byte array that is passed to the *setEncryptKey()* methods of two *Stream* objects.

Stream objects are either instances of the *EncryptedInputStream* and *EncryptedOutputStream* classes or wrappers around these classes (in case of *HTTPConnection*). These classes perform the actual encryption and decryption of the data streams. Review of the pseudo-code confirms that the cipher in use is indeed RC4 and that the key of the cipher is the one passed to the *setEncryptKey()* methods (the relevant pseudo-code is included in the Appendix). This observation confirms that although there are 64 bits of key material exchanged on protocol initiation and the key length of RC4 is 40 bits, **the effective key length of the encryption is only 32 bits** since the third byte of the key is always -82 (0xAE).

After the key is set encryption and decryption is performed byte-by-byte on the data streams, no message authentication or integrity checking takes place. The structure of the underlying data is determined by the proprietary message format of Oracle Forms.

2.1.2. Message Format

Source code review revealed that Oracle Forms implements a custom serialization format to transmit different Java objects over the network. The serialization is implemented in the *Message* class of the *oracle.java.forms.engine* package. A *Message* represents a series of *Properties* representing Java basic types and objects. *Message*s can be nested: a *Message* object can hold other *Message* objects as its *Properties*.

Messages start with a variable sized header that identifies the type of the *Message*. *Message* types can describe standard CRUD (Create, Read, Update, Delete) functionality and there are also 4 protocol specific message types: two types to indicate “delta” messages and two types to indicate “client” messages. These message types aren’t further investigated as they don’t affect the data representation format of the protocol.

After the header the *Properties* of the *Message* are serialized sequentially. A serialized property starts with a 2 or 3 byte prefix (header) that indicates its type. After the prefix the determining parameters of the objects follow as an unaligned byte stream. The serialization formats of the serializable types are summarized in the following table:

1. Table Object serialization formats

Type	Property Type Header	Representation
Boolean (true)	0x5000	N/A
Boolean (false)	0x6000	N/A
Integer (0)	0x1000	N/A
Integer (0-255)	0x2000	Integer value as 1 byte
Integer (255-65535)	0x3000	Integer value as 2 bytes
Integer (other)	0x0000	Value as 4 bytes
String	0x4000	1 byte identifier (see description below) Length: 2 bytes UTF-8 string buffer
String reference	0x9000	1 byte identifier 1 byte new identifier (see description below)
Byte	0x7000	Byte value
null	0x8000	N/A
String[]	0xE00002	Array length: 1 byte UTF-8 Strings with 2 byte length prefixes
Float	0xE00005	Float value
Date	0xE00006	Timestamp represented on 8 bytes
byte[]	0xE00007 0xE0000F	Array length: 1 byte Array elements

Message	0xE00008	Serialized Message
Rectangle	0xE0000A	Coordinate X: 2 bytes Coordinate Y: 2 bytes Width: 2 bytes Height: 2 bytes
Point	0xC000 0xA000	Coordinate X: 1 or 2 bytes Coordinate Y:1 or 2 bytes
Character	0xE00004	UTF-8 character (2 bytes)
DeleteMask	0xD000	2-byte identifier

Boolean values, *null* and the *Integer* value 0 is encoded inside the property header. *Integers* bigger than 0 are "optimized", so that they acquire as many bytes as their value require. All numbers are big endian.

String objects can be cached and then referenced by later *Properties*. As *String* objects are encountered, they are stored in a *String* array indexed with the 1-byte identifier following the type prefix. Later *Properties* can reference previous *Strings* by using the 0x9000 type prefix and supplying the *String* identifier. *String* references also cause the value of the *String* cache replaced with another element (pointed by the second byte after the type prefix) of the cache array.

Messages end with the byte value -16 (0xF0). A single HTTP message can hold multiple *Messages*. The end of the *Message* sequence is indicated by the 0xF001 byte pair.

To help interpreting decrypted *Messages* a simple test program (`MessageTester.java`) was created. The program takes the hexadecimal representation of a message as an argument and parses it using the packages of `frmall.jar`. The decompiled source of the *oracle.forms.engine.Message* class can be provided to standard debugging tools to allow step-by-step tracing of the parser. A decrypted *Message* stream is provided below as an example:

```
10000b50adf010002450aef0100006a089038402bcf0f001
```

The stream consists of three *Messages*: two boolean values and a *Point* object. *Message* headers (blue) are three bytes long. In case of boolean values the type identifier prefix (orange) encodes the value (0x5000 - *true*) itself. The coordinate values of the *Point* are represented as two 2-byte values (green). In this case the Point is at the (900,700) coordinate (0x384,0x2bc). *Message* terminators (0xF0) are marked red. The final *Message* is indicated by the 0xF001 sequence (black).

2.2. Attacks on Cryptography

2.2.1. Overview of RC4 weaknesses

Because of its cryptographic weaknesses the prohibition of use of RC4 in TLS protocols is proposed by IETF in RFC7465 (Popov, 2015.). That decision was induced by the results of AlFardan et al. showing that plaintext recovery is possible under realistic circumstances (Nadhem AlFardan, 2013.). The attacks proposed by this research"require a fixed plaintext to be RC4-encrypted and transmitted many times in succession" and "large amounts of ciphertext" to be recorded. The first requirement is met because the initialization messages of Oracle Forms are mostly static (see section 2.2.3). The fulfillment of the second requirement is dependent on the particular target. It's also worth noting that the attacks were developed against the 128-bit version of the cipher while Oracle Forms uses the weakest, 40-bit version. This may reduce the complexity of attacks significantly.

While the known cryptanalytic attacks seem applicable to Oracle Forms their implementation is out of the scope of this paper. The following sections provide more effective ways to break the security of the communication exploiting the naive use of cryptography.

2.2.2. Passive Network Attacks

In a passive network attack the attacker intercepts the entire communication between the Oracle Forms client and the server. In this case the key exchange scheme described in 2.1.1 can be trivially attacked: The attacker intercepts both the server- and client-supplied components of the key and constructs the RC4 key that can be used to decrypt the whole encrypted data stream. This approach is implemented in the **OracleFormsTester** Burp Suite extension accompanying this document.

This attack against the key exchange renders encryption useless because a passive attacker can obtain the key. If no other protections (like TLS or IPsec) are implemented the security of communication is equivalent to plain HTTP.

2.2.3. Active Network Attacks

During an active network attack the attacker can intercept and modify any traffic passing between the Oracle Forms server and client. One of the countermeasures against these attacks is mutual authentication that prevents the attacker from acting as a man in the middle. Oracle Forms doesn't implement any authentication scheme at the network level, the only requirement for a man-in-the-middle attack is the knowledge of the protocol of the framework.

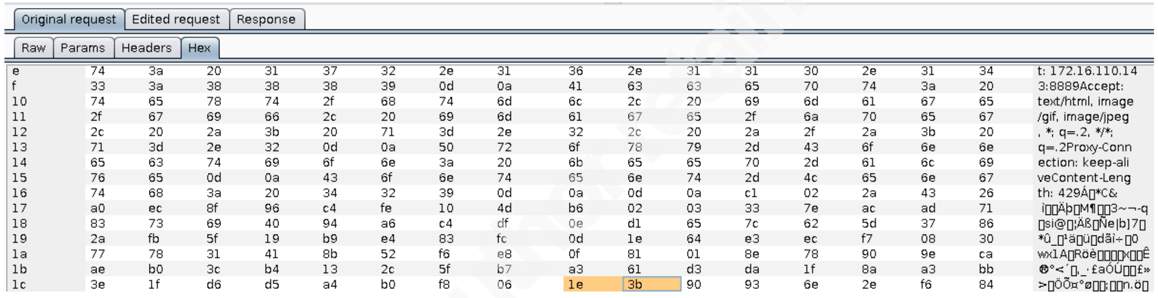
Another important countermeasure against active attacks is integrity checking that is especially important when using stream ciphers like RC4 to avoid bit-flipping attacks (David LeBlank, 2002). In a bit-flipping attack the attacker takes advantage of the general structure of stream ciphers to modify bits in the decrypted plaintext by flipping bits at the same position in the ciphertext. Since Oracle Forms doesn't implement integrity protection bit-flipping attacks can be trivially mounted. For the bit-flipping to be meaningful the attacker must predict the position of the data to be corrupted.

In case of Oracle Forms there is an obvious opportunity for exploitation during the standard applet initialization phase. In this phase the client sends an Oracle configuration string to the server. This string usually contains the name or IP address of the Oracle server to connect to. The offset of the server location substring is predictable because the format of the configuration string is known and it is always sent at the start of the first client request.

The following screenshots show how an encrypted request was edited using Burp Suite. The configuration string of the original request was:

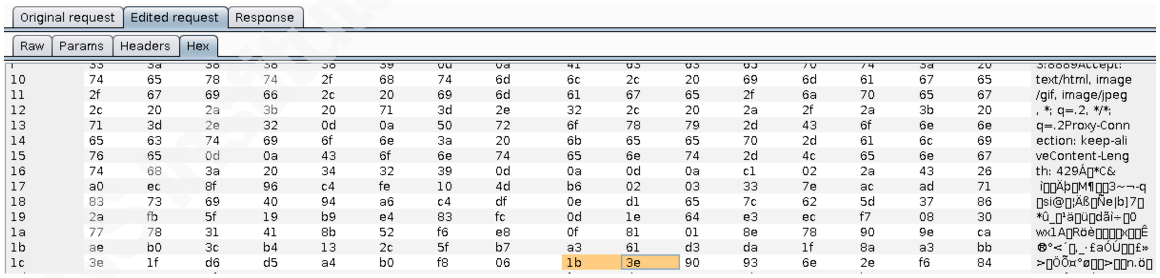
```
server escapeParams=true module=tuto_forms.fmx
userid=SYSTEM/oracle@127.0.0.1/XE  sso_userid=%20
sso_formsid=%25OID_FORMSID%25 sso_subDN= sso_usrDN= debug=no
host= port= buffer_records=no              debug_messages=no
array=no obr=no query_only=no quiet=yes render=no record=
tracegroup= log= term=
```

The original configuration pointed to the Oracle database at 127.0.0.1. As a demonstration the bits of 2nd and 3rd bytes of the IP address were flipped to make up the "72" substring:


Original request | Edited request | Response
Raw | Params | Headers | Hex

```
e    74 3a 20 31 37 32 2e 31 36 2e 31 31 30 2e 31 34  t: 172.16.110.14
f    33 3a 38 38 38 39 0d 0a 41 63 63 65 70 74 3a 20  3:8889Accept:
10   74 65 78 74 2f 68 74 6d 6c 2c 20 69 6d 61 67 65  text/html, image
11   2f 67 69 66 2c 20 69 6d 61 67 65 2f 6a 70 65 67  /gif, image/jpeg
12   2c 20 2a 3b 20 71 3d 2e 32 2c 20 2a 2f 2a 3b 20  , *; q=.2, */*;
13   71 3d 2e 32 0d 0a 50 72 6f 78 79 2d 43 6f 6e 6e  q=.2Proxy-Conn
14   65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69  ection: keep-ali
15   76 65 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67  veContent-Leng
16   74 68 3a 20 34 32 39 0d 0a 0d 0a c1 02 2a 43 26  th: 429Á*C&
17   a0 ec 8f 96 c4 fe 10 4d b6 02 03 33 7e ac ad 71   ìÄþM¶3~¬-q
18   83 73 69 40 94 a6 c4 df 0e d1 65 7c 62 5d 37 86  si@¦ÄßÑe|b]7
19   2a fb 5f 19 b9 e4 83 fc 0d 1e 64 e3 ec f7 08 30  *û_¹äüdãì÷0
1a   77 78 31 41 8b 52 f6 e8 0f 81 01 8e 78 90 9e ca  wx1ARöèxÊ
1b   ae b0 3c b4 13 2c 5f b7 a3 61 d3 da 1f 8a a3 bb  ®°<´,_·£aÓÚ£»
1c   3e 1f d6 d5 a4 b0 f8 06 1e 3b 90 93 6e 2e f6 84  >ÖÕ¤°ø;n.ö
```

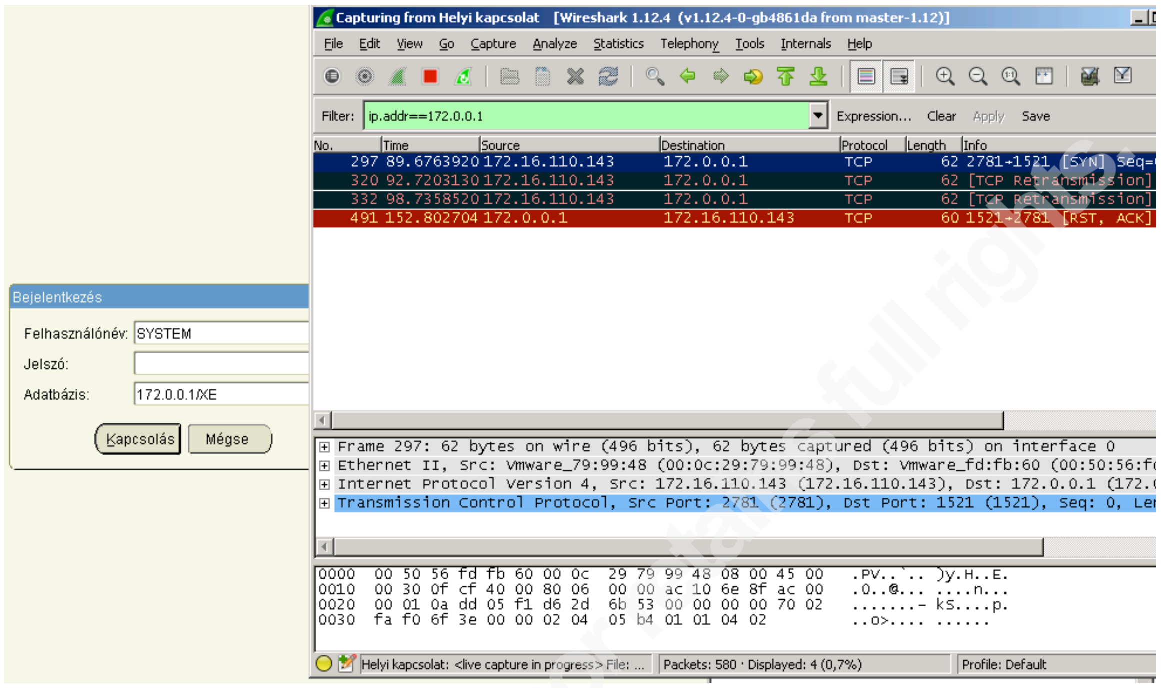
1. Figure Original encrypted request containing database connection information


Original request | Edited request | Response
Raw | Params | Headers | Hex

```
10   74 65 78 74 2f 68 74 6d 6c 2c 20 69 6d 61 67 65  text/html, image
11   2f 67 69 66 2c 20 69 6d 61 67 65 2f 6a 70 65 67  /gif, image/jpeg
12   2c 20 2a 3b 20 71 3d 2e 32 2c 20 2a 2f 2a 3b 20  , *; q=.2, */*;
13   71 3d 2e 32 0d 0a 50 72 6f 78 79 2d 43 6f 6e 6e  q=.2Proxy-Conn
14   65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69  ection: keep-ali
15   76 65 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67  veContent-Leng
16   74 68 3a 20 34 32 39 0d 0a 0d 0a c1 02 2a 43 26  th: 429Á*C&
17   a0 ec 8f 96 c4 fe 10 4d b6 02 03 33 7e ac ad 71   ìÄþM¶3~¬-q
18   83 73 69 40 94 a6 c4 df 0e d1 65 7c 62 5d 37 86  si@¦ÄßÑe|b]7
19   2a fb 5f 19 b9 e4 83 fc 0d 1e 64 e3 ec f7 08 30  *û_¹äüdãì÷0
1a   77 78 31 41 8b 52 f6 e8 0f 81 01 8e 78 90 9e ca  wx1ARöèxÊ
1b   ae b0 3c b4 13 2c 5f b7 a3 61 d3 da 1f 8a a3 bb  ®°<´,_·£aÓÚ£»
1c   3e 1f d6 d5 a4 b0 f8 06 1b 3e 90 93 6e 2e f6 84  >ÖÕ¤°ø>n.ö
```

2. Figure Edited encrypted request containing database connection information

After the edited request was forwarded to the server connection attempts to the 172.0.0.1 address were registered by the Wireshark network analyzer. Since this host didn't exist the connection timed out and the client reported an error. After the error was dismissed the client displayed a login window containing the modified IP address.


Bejelentkezés
Kapcsolás
Mégse
Capturing from Helyi kapcsolat [Wireshark 1.12.4 (v1.12.4-0-gb4861da from master-1.12)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

Felhasználónév: SYSTEM
Jelszó:
Adatbázis: 172.0.0.1/XE

Filter: ip.addr==172.0.0.1

```
No.  Time        Source          Destination     Protocol Length Info
 297 89.6763920  172.16.110.143  172.0.0.1       TCP      62 2781→1521 [SYN] Seq=
 320 92.7203130  172.16.110.143  172.0.0.1       TCP      62 [TCP Retransmission]
 332 98.7358520  172.16.110.143  172.0.0.1       TCP      62 [TCP Retransmission]
 491 152.802704  172.0.0.1       172.16.110.143  TCP      60 1521→2781 [RST, ACK]
```

```
⊞ Frame 297: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
⊞ Ethernet II, Src: Vmware_79:99:48 (00:0c:29:79:99:48), Dst: Vmware_fd:fb:60 (00:50:56:f
⊞ Internet Protocol Version 4, Src: 172.16.110.143 (172.16.110.143), Dst: 172.0.0.1 (172.
⊞ Transmission Control Protocol, Src Port: 2781 (2781), Dst Port: 1521 (1521), Seq: 0, Le
```

```
0000  00 50 56 fd fb 60 00 0c  29 79 99 48 08 00 45 00   .PV..`.. )y.H..E.
0010  00 30 0f cf 40 00 80 06  00 00 ac 10 6e 8f ac 00   .0..@... ....n...
0020  00 01 0a dd 05 f1 d6 2d  6b 53 00 00 00 00 70 02   .......- kS....p.
0030  fa f0 6f 3e 00 00 02 04  05 b4 01 01 04 02         ..o>.... ......
```

Helyi kapcsolat: <live capture in progress> File: ...
Packets: 580 · Displayed: 4 (0,7%)
Profile: Default

3. Figure Results of bit-flipping at client and server side

This proof-of-concept attack demonstrates that **the encryption scheme is vulnerable to bit-flipping attacks** independently from the strength of the key-exchange protocol phase.

2.2.4. Insecure password handling

In section 2.2.3 the database user and the password were configured server side, but the modified packet containing the same information was sent by the client. This means that **Oracle Forms sends the configured database credentials to its clients**. This creates a security weakness that allows unauthenticated attackers to obtain the configured password. The credentials are sent to the client in response to a request similar to the following:

GET
/forms/lservlet;jsessionid=*sessionid*?ifcmd=getinfo&ifhost=*hostname*&ifip=192.168.1.1 HTTP/1.0

The response contains the credentials encoded with the DEFLATE algorithm (Deutsch, 1996). The response body can be decoded using the *zlib* library (Akira, 2010):

```
python -c "import zlib,sys;print \
repr(zlib.decompress(sys.stdin.read()))" < creds.bin
```

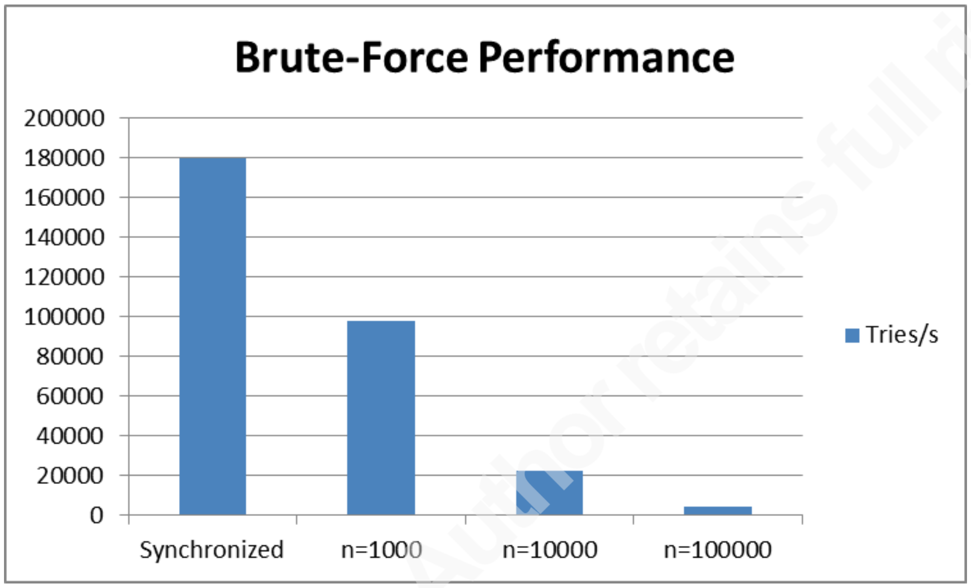
Further research is required to assess if other configuration parameters sent by the client can be used to conduct attacks.

2.2.5. Brute-force

If an attacker can intercept only a subset of encrypted HTTP messages (e.g. when performing an ARP poisoning attack (Jeff King, 2010)) she can't rely on the weak key exchange scheme as the required key material is not necessarily included in the captured packets.

Since the key size of the protocol is effectively 32-bits (see 2.1.1) brute-force attacks against the captured ciphertext can be practical. As a proof of concept a primitive Java program (`OracleFormsBruteForce.java`) was built that decrypts a given ciphertext with every possible key and detects if the resulting plaintext is a valid Oracle Forms *Message*. This program is suboptimal as it is written in a high level language, runs only a single thread, decrypts the whole message every time, etc. Still, the program performs around 180.000 tries per second on a dual core Intel Core i5-520M (2.4 GHz) CPU. At this speed a valid key is found from the 32-bit key-space in 3.3 hours on average.

The average attack speed could probably be reduced to minutes on average hardware but further optimizations were not made as the approach has serious practical limitations: The assumption that the attacker doesn't intercept the communication from the beginning implies that the attacker doesn't know which part of the stream she is intercepting. To measure the effect of this uncertainty a modified version of the brute-force program was built that predicts parts of the keystream assuming that the intercepted ciphertext begins with a known 3-byte header (see 2.1.2). RC4 works by generating a pseudo-random byte stream (the keystream) that is XOR-ed with the bytes of the plaintext. To predict part of the matching keystream the first 3 bytes of the ciphertext is XOR-ed with the presumed plaintext header bytes. Then an *n* byte long keystream is generated in which the program looks for the predicted byte triplet for every possible key. It is assumed that the attacker knows which part of the cipher stream she intercepted with *n* byte precision. If the program finds a match it tries to use the keystream from the matching offset to decrypt the whole ciphertext. If the decryption results in a valid protocol message the key (and the matching offset) is found. The following diagram shows how the performance of brute-force search is impacted as *n* grows:



4. Figure Brute-force performance in unsynchronized state

The diagram shows that loosing synchronization with the stream cipher has serious impact on the performance of a brute-force attack. Such an attack may be feasible if the average sessions are relatively short or if the attacker can obtain additional information about the state of the cipher.

It's worth noting that this is not just a limitation for the attacker: In case of network problems legitimate Oracle Forms peers can easily lose sync.

2.3. Test Automation

Testing is a fundamental part of any software development process, that helps eliminate a wide range of errors from simple usability issues to critical vulnerabilities. Although manual testing can't be avoided test automation is fundamental to maintain large software projects. In terms of security it is crucial to be able to effectively run a high number of test cases (like generic patterns of injection attacks) in a black-box manner.

In the case of Oracle Forms this effort is hindered by several factors: The application protocol is undocumented, the communication is encrypted and the testing tools provided by the vendor are integrated strongly with its proprietary tools (Oracle Corporation, 2009.).

The results of reverse engineering described in section 2.1 makes it possible to create tools which can interact with Oracle Forms applications without having access to the server-side components or source code. A Proof of Concept extension, **OracleFormsTester** was built for Burp Suite Professional that demonstrates the practical application of the revealed details and also the challenges of the automated testing.

2.3.1. Test software design

The main automated testing component of Burp Suite is the Scanner that works by taking HTTP requests issued by the browser and replays them with modified content. The modifications are based on a large set of rules which define several payloads to be inserted as part of HTTP (or REST, etc.) parameters. To achieve similar result in case of Oracle Forms the testing software has to:

1. Intercept the traffic of the Oracle Forms applet
2. Decrypt the intercepted request
3. Identify potential insertion points (*Message* parsing)
4. Insert payload (*Message* serialization)
5. Encrypt and resend the modified request

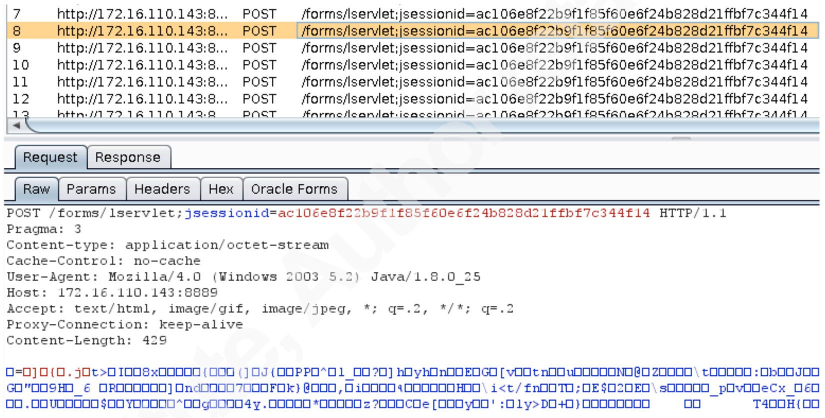
Sample traffic can be easily intercepted by setting the proxy server used by the Java runtime to the address of an intercepting proxy (like Burp Suite). In the case of a raw TCP channel the routing table of the client can be modified to run traffic through an intercepting network node.

While both cryptographic operations and *Message* handling tasks may seem challenging the Java technology allows easy code reuse: During normal operation the client-side part of the Oracle Forms framework (`frmall.jar`) is sent to the client for use as part of the applet. The downloaded archive contains all code that is required to parse and create valid protocol messages. The archive can be saved and reused in custom programs by simply adding it to the Java classpath.

2.3.2. Decryption and Encryption

Decryption and encryption is implemented in the *EncryptedInputStream* and *EncryptedOutputStream* classes of the *oracle.forms.net* package. The classes are inherited from "filter" stream classes (*FilterInputStream*, *FilterOutputStream*) which can serve as a transformation layer for other basic streams (like data or byte streams). Since a standard cipher is employed, independent implementations can also be used without modification. The encryption key can be trivially derived from the intercepted traffic.

The following screenshots demonstrate successful decryption of Oracle Forms *Message*s through the Message Editor Tab introduced by **OracleFormsTester**:

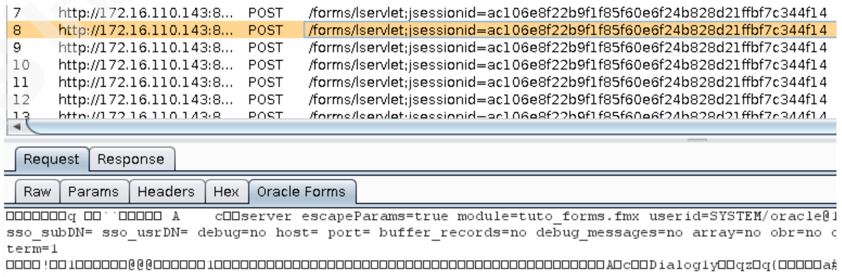


```
7    http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
8    http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
9    http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
10   http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
11   http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
12   http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
```

```
POST /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14 HTTP/1.1
Pragma: 3
Content-type: application/octet-stream
Cache-Control: no-cache
User-Agent: Mozilla/4.0 (Windows 2003 5.2) Java/1.8.0_25
Host: 172.16.110.143:8889
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Proxy-Connection: keep-alive
Content-Length: 429

□=□]□{□.j□t>□I□□8x□□□□□{□□□(]□J{□□PP□^□1_□□?□]h□yh□n□□E□G□[v□□tn□□u□□□□□N□@□Z□□□□\t□□□□□:□b□□J□□
G□"□□9H□_6 □R□□□□□□]□nd□□□□7□□□F□k}@□□□,□i□□□□□□□□□□H□□\i<t/fn□□T□;□E$□2□E□\s□□□□□_p□v□□eCx_□6□
□□.□□U□□□□□$□□Y□□□□□^□□g□□□□4y.□□□□□*□□□□□z?□□□C□e[□□□y□□':□1y>D□+□}□□□□□□□□□    □□      T4□□H{□□
```

5. Figure Intercepted request (encrypted)



```
7    http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
8    http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
9    http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
10   http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
11   http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
12   http://172.16.110.143:8...   POST   /forms/lservlet;jsessionid=ac106e8f22b9f1f85f60e6f24b828d21ffbf7c344f14
```

```
□□□□□□□q □□``□□□□□ A    c□□server escapeParams=true module=tuto_forms.fmx userid=SYSTEM/oracle@
sso_subDN= sso_usrDN= debug=no host= port= buffer_records=no debug_messages=no array=no obr=no
term=1
□□□□!□□1□□□□□□□@@@□□□□□□1□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□A□c□□Dialog1y□□qz□q{□□□□□a#
```

6. Figure Decrypted body of intercepted request

Both the HTTP protocol and the *Message* format are stateless: valid messages can be constructed independently from each other. Encryption however requires the internal states of the ciphers at client and server side to be synchronized. This means that issuing new messages independently from the client makes the applet unable to communicate with the server. Also, corrupted messages (like heartbeats) sent by the client can desynchronize the testing tool. Both problems can be solved by shutting down the client as the testing starts. A more advanced solution is to block the traffic of the client while the tests run and then setting the internal state of the cipher appropriately via a debugger.

In case of **OracleFormsTester** every decrypted request body and the corresponding cipher state are stored in a *HashMap* indexed by the hash of the encrypted body. This way plaintext messages can be found for every intercepted ciphertext. The recoded plaintext messages are also used to select all *String* properties as Scanner insertion points. When the Scanner runs the *Messages* are deserialized (see section 2.3.3), the insertion points are replaced with the upcoming payload then the *Message* is serialized and encrypted again.

2.3.3. Message parsing and serialization

Message parsing and serialization are performed by the *writeDetails()* and *readDetails()* methods of the *Message* class. These implementations can be easily reused, but successful message parsing requires taking care of two important details. First, the *readDetails()* method should be invoked with a valid *String* array as its second argument to support *String* caching (see section 2.1.2). Second, when decrypting *Messages* the *EncryptedInputStream(InputStream, boolean)* constructor should be used with the second argument set to *true* to initialize deciphering objects. This is important because the single argument constructor declares only a 7800 bytes long internal buffer for the stream while the two argument version declares 8192 bytes. Since some initial protocol messages usually exceed 7800 bytes they can't be read in a single run with the smaller buffer.

Successful Message parsing can be confirmed in the debug output of the **OrcleFormsTester** extension:

```
Oracle Forms Tester loaded
Found GDay!
Found GDay! f1dbf270
Found Mate! 00000390
RC4 Key: f239aedb00
[...]
Encrypted Request: 77d4f3cdc03b22d2beb9ed71
506fd22b5a9b50757516f1b341423f546127b524d4189c15afa244dabefe1e6d
Body: 12 byte(s)
Property 0: 137 Type: 9
--- Value: java.awt.Point[x=900,y=700]
```

```
Message OK
100006a089038402bcf0f001
```

2.4. Deployment Best Practices

The attacks presented in section 2.2 showed that despite the encryption the security of the Oracle Forms protocol is nearly equivalent to a plain text channel. The transferred data should be protected with standard technologies like TLS or IPsec.

Oracle Forms shouldn't be configured with database access credentials since this information is sent to the clients as described in section 2.2.4. The *userid* parameter should be set empty in the `formsweb.cfg` configuration file (Oracle Corporation, 2006). Database passwords of existing installations with similar configuration should be considered compromised.

3. Conclusion

This paper gave detailed overview on the communication protocol of Oracle Forms and presented tools and methods to support automated testing. Multiple attacks were presented which prove that the protective features of the framework give a false sense of security as they can be circumvented easily.

Although Oracle Forms is meant to be a tool for Rapid Application Development (Oracle Corporation, 2009) it hinders testing – one of the most important steps of development – considerably. While test automation is possible, complex tools must be built to support Oracle Forms. This property doesn't only affect the security of Oracle Forms based applications but the general quality of them too. A symptom of this inadequacy is that this research revealed fundamental security weaknesses which would likely have been found and mitigated in time in case of a more open technology.

The presented results confirm the opinion that Oracle Forms is an obsolete technology and applications should be replaced (Berg, 2013). Hopefully the provided information will help assessing the quality of the live applications and migrating them to more modern platforms. Further research is required to determine if the design of Oracle Forms introduce security risks that may affect a wider range of applications.

References

Akira. (2010). *Deflate command line tool*. Retrieved from Stack Overflow: http://stackoverflow.com/a/8326032

Berg, E. (2013., February 25.). *The Oracle Forms Dilemma - part 1.* Retrieved from http://whywebsphere.com/2013/02/25/the-oracle-forms-dilemma-part-1/

David LeBlank, M. H. (2002). *Writing Secure Code (2nd Edition) (Developer Best Practices).* Redmond: Microsoft Press.

Deutsch, L. P. (1996). *DEFLATE Compressed Data Format Specification version 1.3*. Retrieved from https://tools.ietf.org/html/rfc1951

Girish Janardhanudu, K. v. (2005., September 26.). *Build Security In.* Retrieved from White Box Testing: https://buildsecurityin.us-cert.gov/articles/best-practices/white-box-testing/white-box-testing

IETF. (2008., August). *The Transport Layer Security (TLS) Protocol Version 1.2.* Retrieved from http://people.csail.mit.edu/rivest/pubs/RS14.pdf

Imperva. (2014., October). *Web Application Attack Report #5.* Retrieved from http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed5.pdf

Java Decompilers Online. (2015.). Retrieved March 15., 2015., from Java Decompilers Online: http://www.javadecompilers.com/

Jeff King, K. L. (2010). *ARP Poisoning (Man-in-the-Middle) Attack and Mitigation Techniques.* Retrieved from http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11_603839.pdf

Microsoft Corporation. (2009.). *Get ready for plug-in free browsing.* Retrieved from https://msdn.microsoft.com/en-us/library/ie/hh968248%28v=vs.85%29.aspx

Mimoso, M. (2013.). *Java's Loosing Security Legacy*. Retrieved from Threat Post: https://threatpost.com/javas-losing-security-legacy/102176

Nadhem AlFardan, D. J. (2013., August). *On the Security of RC4 in TLS.* 22nd USENIX Security Symposium, Washington, D.C.

Oracle Corporation. (2000). *Security Considerations.* Retrieved from Oracle Forms Server Release 6i: Deploying Forms Applications to the Web with Oracle Internet

Application Server: http://docs.oracle.com/cd/A97338_01/doc/forms.6i/a83591/chap10.htm

Oracle Corporation. (2006.). *Configuration Files*. Forrás: Oracle Application Server Forms Services Deployment Guide 10g Release 2 (10.1.2) : https://docs.oracle.com/cd/B14099_19/web.1012/b14032/basics002.htm

Oracle Corporation. (2009, October). *Frequently Asked Questions for Oracle Forms 11g.* Retrieved from http://www.oracle.com/technetwork/developer-tools/forms/documentation/faq-2-130064.pdf

Oracle Corporation. (2009, June). *Oracle Forms Services & Oracle Forms Developer 11g Technical Overview.* Retrieved from http://www.oracle.com/technetwork/developer-tools/forms/overview/technical-overview-130127.pdf

Oracle Corporation. (2009.). *Using the Oracle Forms Functional Test Module.* Retrieved from Oracle Application Testing Suite OpenScript User's Guide: https://docs.oracle.com/cd/E25291_01/doc.900/e15488/opscrpt_using_offt_module.htm

Oracle Corporation. (2012., March). *Oracle Application Development Tools Statement of Direction: Oracle Forms, Oracle Reports and Oracle Designer.* Retrieved from http://www.oracle.com/technetwork/issue-archive/2010/toolssod-3-129969.pdf

Oracle Corporation. (2014.). *Oracle Application Development Framework.* Retrieved 2015., from http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html

Oracle Corporation. (2015). *Java Documentation.* Forrás: Connection and Invocation Details: https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/conninv.html

Oracle Forms. (n.d.). Retrieved 03 15, 2015, from http://www.oracle.com/technetwork/developer-tools/forms/overview/index-098877.html

OWASP. (2014.). *Appendix A: Testing Tools.* Forrás: OWASP Testing Guide v4: https://www.owasp.org/index.php/Appendix_A:_Testing_Tools

Popov, A. (2015.). *Prohibiting RC4 Cipher Suites*. Forrás: https://tools.ietf.org/html/rfc7465

PortSwigger Ltd. (n.d.). *Burp Suite.* Retrieved March 15., 2015., from Portswigger: http://portswigger.net/burp/

Roland, G. (2009). *Migrating Oracle Forms to Fusion: Myth or Magic Bullet?* Retrieved from http://www.oracle.com/technetwork/developer-tools/forms/documentation/formsmigration-133693.pdf

Ron L. Rivest, J. S. (2014, October 27.). *Spritz - a spongy RC4-like stream cipher and hash function.* Retrieved from http://people.csail.mit.edu/rivest/pubs/RS14.pdf

Schuh, J. (2013). *Saying Goodbye to Our Old Friend NPAPI*. Retrieved from Chromium Blog: https://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html

Appendix

Decompiled RC4 pseudo-code

Relevant methods decompiled from *oracle.forms.net.EncryptedInputStream*:

```
    public synchronized void setEncryptKey(byte abyte0[])
    {
        if(abyte0 == null || abyte0.length == 0 || abyte0.length
> 256)
            throw new RuntimeException();
        mSeedBuffer = new int[256];
        mI = mJ = 0;
        for(int i = 0; i < 256; i++)
            mSeedBuffer[i] = i;

        int l;
        int k = l = 0;
        for(int j = 0; j < 256; j++)
        {
            l = (l + (abyte0[k] & 0xff) + mSeedBuffer[j]) % 256;
            int i1 = mSeedBuffer[j];
            mSeedBuffer[j] = mSeedBuffer[l];
            mSeedBuffer[l] = i1;
            k = (k + 1) % abyte0.length;
        }

    }

    public synchronized int read()
        throws IOException
    {
        if(mPos == mLength)
        {
            fill();
            if(mPos == mLength)
```

```
                return -1;
        }
        return mBuf[mPos++] & 0xff;
    }

    public synchronized int read(byte abyte0[], int i, int j)
        throws IOException
    {
        if(mPos + j > mLength)
        {
            fill();
            if(mPos >= mLength)
                return -1;
        }
        int k = mLength - mPos;
        k = k >= j ? j : k;
        System.arraycopy(mBuf, mPos, abyte0, i, k);
        mPos += k;
        return k;
    }
    private void fill()
        throws IOException
    {
        int i;
        if(mPos == mLength)
        {
            mLength = 0;
            i = 0;
        } else
        {
            mLength = mLength - mPos;
            System.arraycopy(mBuf, mPos, mBuf, 0, mLength);
            i = mLength;
        }
        int j = mIstream.read(mBuf, i, mBuf.length - i);
        if(j > 0)
        {
            mLength += j;
            mBytesCount += j;
        }
        mPos = 0;
        int ai[] = mSeedBuffer;
        if(ai != null)
        {
            int k = mI;
            int l = mJ;
            for(int i1 = i; i1 < mLength; i1++)
            {
                k = (k + 1) % 256;
                l = (ai[k] + l) % 256;
                int j1 = ai[k];
                ai[k] = ai[l];
                ai[l] = j1;
                mBuf[i1] ^= ai[(ai[k] + j1) % 256];
            }

            mI = k;
            mJ = l;
        }
    }
```