Reply Challenge 2023 CTF

3rd Floor CSIS

Blake Ryan
Brian Riordan
Euan Bourke
Milan Kovacs

Table of Contents

Coding

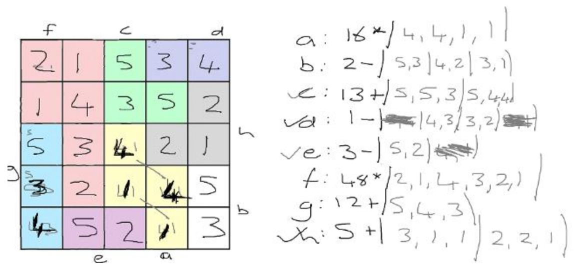
We sort of ignored the coding challenges at the beginning of the event and focused on the other categories.

Coding100

At 5pm (1.5hr before deadline), having experience solving sudokus, we decided to just do them manually, which on reflection was not the best idea and a script could have been written in much shorter time. On top of that we did them in paint of all things.

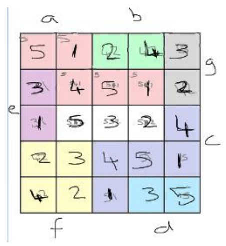
Surprisingly we still managed to get 3 of the levels done in the remaining 1.5hr.

Level 1:



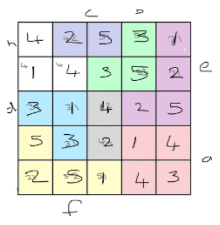
Answer: 2153414352534213214545213

Level 2:



Answer: 5124334512153242345142135

Level 3:



Answer: 4253114352314255321425143

Unfortunately, we didn't make it past this.

Web

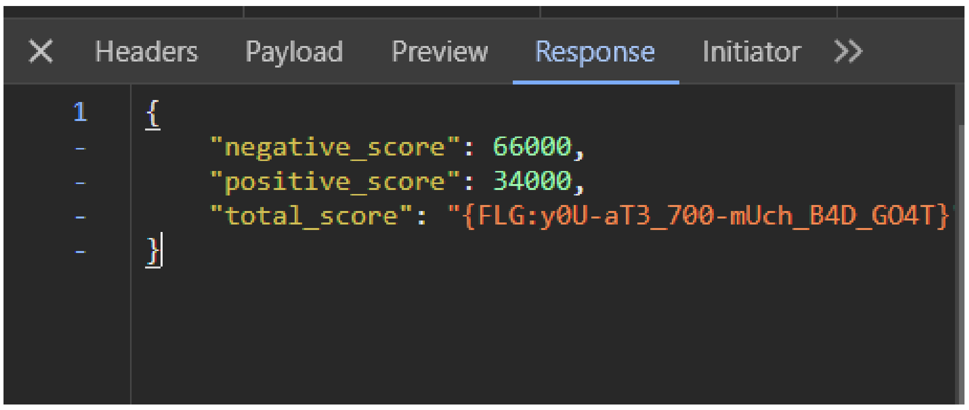
General

Web100

We were brought to a webpage that had a minigame on it. You would use your keyboard to position an image across the bottom of the screen to catch some falling images that spawned randomly above. You would gain or lose points for catching good or bad images, 2000 and –2000 respectively. The goal of the game was to amass a score of 65,536, which immediately set off alarms as this would be impossible with increments of +-2000.

Our knowledge of computer hardware let us figure out the trick, 65.536 is the upper bound for storing 16-bit integers, so we figured the best course of action would be to get the score to roughly 64,000 – 66,000 and see if some boundaries are hit. We found it easier to run JavaScript locally on the page rather than actually play the game, as this would take too much time. Calling a function update() with either red or green as the args let us change the score in an easier manner.

After trying various combinations to hit the 66,000 boundary, we found one that works, going completely red/negative until about –66,000 at which point the counter resets to positive 33,000, and the continuing to send green/positive scores until we hit 66,000 positive, at which point we assume some logic with the storing/conversion of 16 bit ints on the server changed the score to the target, which is 65,536.



```
{
    "negative_score": 66000,
    "positive_score": 34000,
    "total_score": "{FLG:y0U-aT3_700-mUch_B4D_GO4T}
}
```

Crypto

Crypto100

This problem had a URL to a website, which has given information on a pair of numbers (N, e) and a ciphered text in decimal, alongside with an endpoint on the bottom.

When you hit the endpoint with the payload:
{
“cipher”: “some number”
}

It returns:

```
GOT IT!! This is your plain text -->
```

And when you send the encrypted message it complains:

Do you want to make me angry?? YOU DON'T DECRYPT THE FLAG!!

After researching on how the RSA encryption algorithm works, here is a boiler plate on from a site:

Choose p = 3 and q = 11
Compute n = p * q = 3 * 11 = 33
Compute ϕ(n) = (p - 1) * (q - 1) = 2 * 10 = 20
Choose e such that 1 < e < ϕ(n) and e and ϕ (n) are coprime. Let e = 7

Compute a value for d such that (d * e) % ϕ(n) = 1. One solution is d = 3 [(3 * 7) % 20 = 1]

Public key is (e, n) => (7, 33)
Private key is (d, n) => (3, 33)

The encryption of m = 2 is c = 27 % 33 = 29
The decryption of c = 29 is m = 293 % 33 = 2

Given, we have (N, e) and in the algorithm:

```
# p and q are primes
# N = p * q
# phi(N) = p-1 * q-1
# e < phi(N)
# decrypted ^ e % phi(N) = encrypted
# encrypted ^ d  % phi(N) = decrypted
```

We were thinking that in order to decrypt it, the ciphered text is either encrypted via the public key N or the private key d. When we tried to decrypt it via the Public key N, we didn’t get far, so we were thinking we need to break N down to its two factors, however since it is very large, computing it would have taken us a really long time. We knew this was not the approach, and this is as far as we have gotten.

Binary

The binary challenges were quite difficult. We were not sure how to proceed or find hints on where to begin

Bin100

This problem gave us a compiled file called ‘steeluoter’ and a zip called flag.zip. We knew we had to find password somehow.

We downloaded a decompiler called Ghidra. Importing the file, Ghidra returned the assembly code along with the function calls and function definitions of the assembly in C.

After scanning it, we weren’t sure what we were looking for so it felt like a need in a haystack.

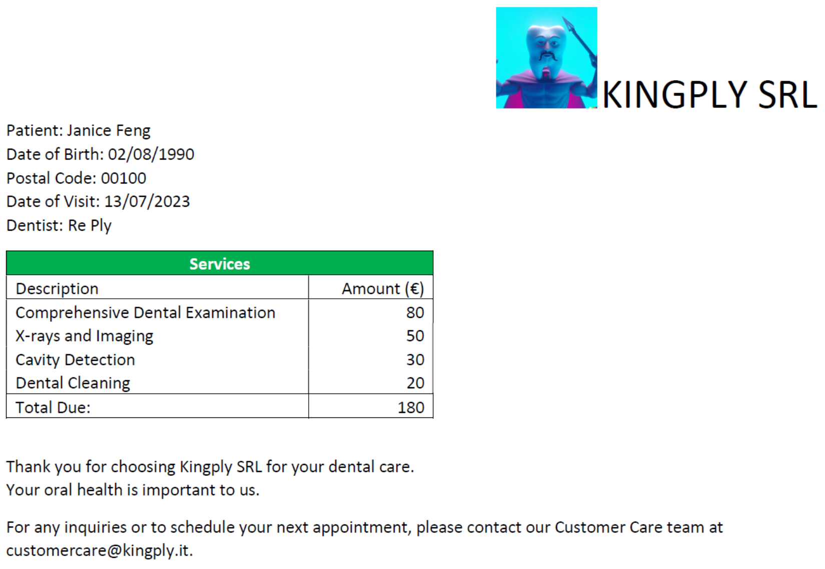
We didn’t consider adding or changing the file extension.

Miscellaneous

This was a very interesting topic and had fun connecting the dots.

Misc100

After unzipping the folder, we were given an export of an email with some attachments. The attachments were in base64 and it also supplied the media type. Converting the first attachment using base64 to image, we were given this:



KINGPLY SRL

Patient: Janice Feng
Date of Birth: 02/08/1990
Postal Code: 00100
Date of Visit: 13/07/2023
Dentist: Re Ply

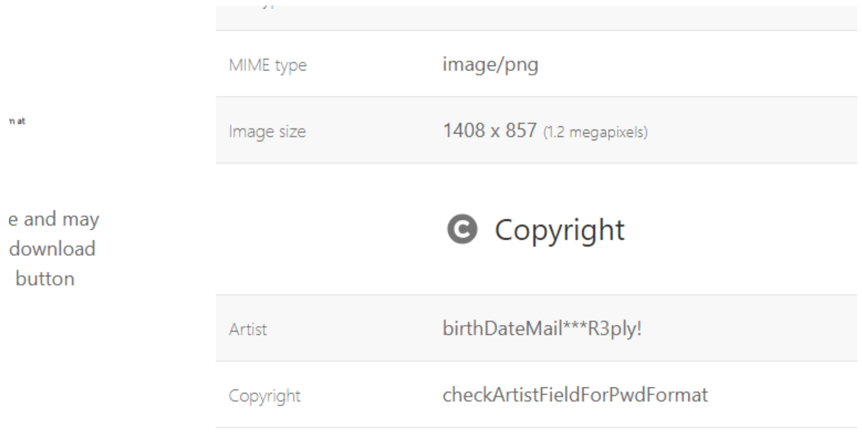
Services	
Description	Amount (€)
Comprehensive Dental Examination	80
X-rays and Imaging	50
Cavity Detection	30
Dental Cleaning	20
Total Due:	180

Thank you for choosing Kingply SRL for your dental care.
Your oral health is important to us.

For any inquiries or to schedule your next appointment, please contact our Customer Care team at customercare@kingply.it.

The second attachment was an file, so a base64 to file got us a zip folder which was password protected.

Importing the image into https://exiftool.org/ we were given the following:



n at

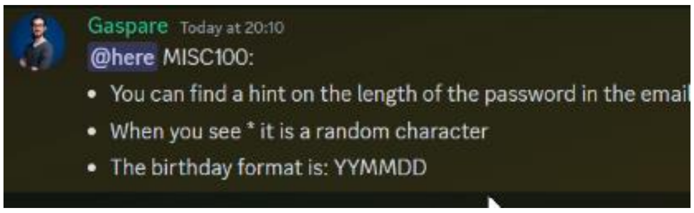
e and may
download
button

MIME type	image/png
Image size	1408 x 857 (1.2 megapixels)

Copyright

Artist	birthDateMail***R3ply!
Copyright	checkArtistFieldForPwdFormat

The Artist field suggested that is the password format which was later confirmed by a hint announced on the Discord server:


Gaspare Today at 20:10
@here MISC100:

- You can find a hint on the length of the password in the email
- When you see * it is a random character
- The birthday format is: YYMMDD

The length of the password was in the bottom of the email, in the same format as the artist field on the top:

_________________________________*R3ply!*

31 characters before R3ply! And we knew the date format took 6 and the 3 asterisk so we were left with 22 characters. Parsing the file and splitting on spaces and removing full stops and commas we were given the ***jfeng@veryrealmail.com*** which matched up with the mail field. Now we needed to find the three random characters. First we tried to write a python script using all alphanumeric characters which failed to open.

Using JohnTheRipper and giving it the format and the information we were given, it cracked it instantly. The three random characters were !@#. Unzipping the file with the given password we were given the flag.

Misc200

Upon opening the zip folder we were given a .pcapng file along with an executable. The executable took in a string called password so we needed to find within the pcap file.

Importing the file into Wireshark, it showed us a sequence of packets. Some about the IP address calls, some about certain UDP calls, but there were two streams of TCP calls which contained JSON data. Once we found a packet which was part of the stream, right click and you follow the TCP stream, we were given the following two JSONs:

TCP stream 1:

Password: c0nn3cti0n_p4ssw0rdZOMBIE78WAITING&READY00WAITZOMBIE78WAITING&READY

TCP stream 2:

**Password:
c0nn3cti0n_p4ssw0rdZOMBIE05WAITING&READY00WAITZOMBIE05WAITING&READY00WAITZOMBIE05WAITING&READY00WAITZOMBIE05WAITING&READYEXPORT{**

"iv": "ddromu4CFhvHG+HE70Bbvw==",

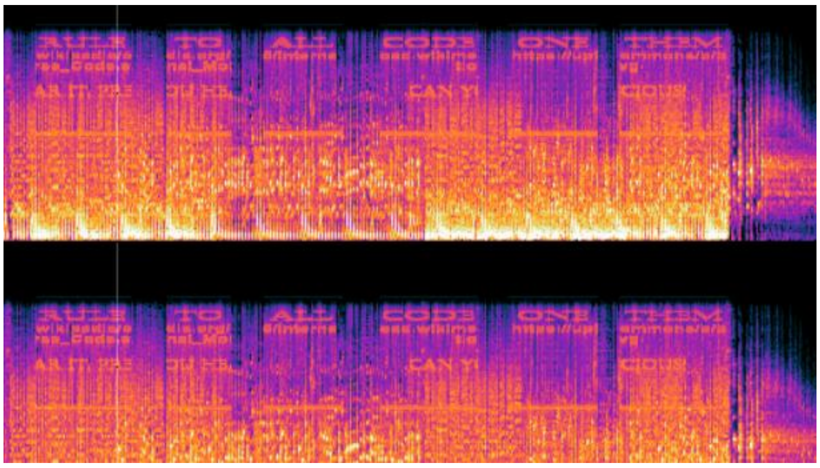
**"ciphertext":
"ER0V1ge5/0jIeBNo5FMELTuszfOVgf/cjSrFbsMlCFOEZFQMKVZfGpQLHvtPIXmcSoM7ahskjgPcBvvc/4TxmQujF48RiSm2kNqaUuQn3+8UdY5/zYoXUglkddVfUfLedSh0F8H6/093dICMojjUTzgnCbztrzdwCUW2pFUpb5y4XVFzKa5AJG6KEMTwsArKZBQ2i3cHTpsby6Jwzm45WkTF+Ns/NK3kISBNiNhSYOHkmSTxi56eRpTCPp6fTy5MDHduOdao+r8csrr5QOwWWaMulDdUmC42zBY1ysmNXaA="}**

After a quick Google search we found 'iv' stands for 'Initialization Vector' which was some sort of a salt in encryption. Both the IV and the ciphertext are in base64 and converting them to ASCII gave us gibberish so we knew we had to decrypt it.

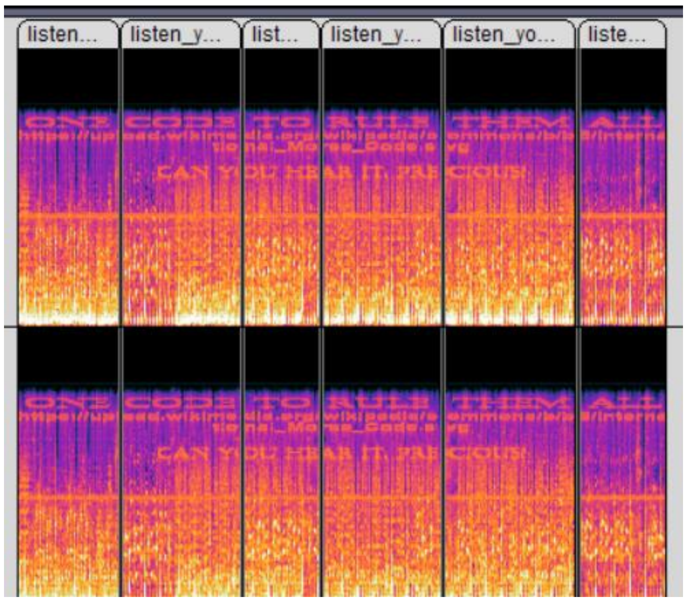
We got stuck on finding the encryption algorithm and thus were blocked.

Misc400

We were given an .wav audio file. First thing we did was listen into it, and heard some buzzing noise along with a Lord of The Rings remix inside it, so we downloaded Audacity and imported the file. The audio waves didn't show anything out of the ordinary, however when turning the file into spectrogram view we were shown this.



If you look under the word ONE you can see there is a https:// which suggested the chunks of the words were out of order. So after cutting it and rearranging it, we were given:



The link brought us to
https://upload.wikimedia.org/wikipedia/commons/b/b5/International_Morse_Code.svg

which is a morse code table. We couldn't find the morse code part or didn't notice what would have been the morse code. Was it the syllables of the lyrics, the words in the spectrogram or a beeping sound in the file.