TheZZAZZGlitch's April Fools Event 2022: Achievements and Technical Writing

By: Darkshadows

Special thanks to Runea for helping me with technical, programatical, mathematical, and emotional support.
Also thanks to ISSOtm for dropping hints when I brainlocked on CCR4. Make sure you also look at his excellent writeup.
Also also thanks to Kirby703 for teaching me how to be a girl.
Also also also thanks to Radixán for reminding me where a few things were that I forgot to write in my hasty notes.
And of course, thank you to TheZZAZZGlitch for holding the event in the first place. It's always amazing to actually have an excuse to dig through all of this stuff.

Much like last year, before anything else: If you want to try anything in Cracker Cavern Reborn, I recommend mGBA connected to GDB. I'll give a short tutorial on some pitfalls I hit both trying to setup GDB and compiling pokeemerald for proper symbols.
If you read my 2018 guide, the broadstrokes are the same for the overworld quests, but the details are different.
Make sure to talk to everyone as you go along and visit every building you come across. It put a big smile on my face and made me laugh with every dumb, little joke.

Return to Glitchland: Main Story Revisited:

Exploration I: There was an attempt:
Exploration II: Starting out:
Exploration III: Taste of adventure:
Exploration IV: Expert:

Mentioning these ones first as a mental note for anyone reading this to keep track of: Make sure to talk to everyone as you go along and visit every building you come across. I finished the event with 131 maps visited, so you can miss a few and still hit the required 120, but it shouldn't be too hard to get all of them. I've included a map that you can use to systemically travel through Glitchland and find every interior map.

To the North!:
From Central Square, go:
North, North, North, North, East, North, North, East, East.

From here, the only place to head is North. The answers to the questions three are:
ODDISH, TORKOAL, and CACTURNE.

After heading North for a final time, talk to Dad to obtain the ability to Surf.

To the East!:
From Central Square, go:
East, East, South, South, South, East, East.

This should put you in Lost Woods. Now go:
East, East, North, North.

Talk to Dad to obtain the ability to push boulders with Strength.

To the South!:
Make sure you have Strength from "To the East!" before doing this one.

From Central Square, go:
South, South, South, South, East, South, East, and South.

Talk to Dad to obtain the ability to ~~touch~~ Cut grass.

To the West!:
Albeit not required, make sure you have Strength from "To the East!" before doing this one. It makes it way easier to get through the dark cave.

From Central Square, go:
West, West, North, and North into the cave

Inside the cave, push the boulder upwards; talk to the person, and they'll light up the cave for you.
If you don't want to get Strength first, bruteforcing is rather easy, as if you don't immediately hit a dead-end, you're going the correct way. It just makes it easier to see to have them Flash you.

Once you've exited the cave, go:
West, West, and West.

Talk to Dad to obtain the ability to teleport between major settlements.

Feels Different:
Altering Cave has three forms cycling every hour, in order:
Dome fossil with braille past water
Cave with braille
Cave with braille that requires strength

Not Lost Anymore:
The secret areas of Lost Woods are reached by moving in the following ways:
North, South, East, East
North, East, South, East (Zelda solution)
East North, South, South

Delirium achieved:
Obtain the pass for Deliria from the bottom left house in Northern Camp. Deliria itself is in the West. Glory to Arstotzka, comrade.

Field of Koopers 5:
Beat the most ultimate awesomest level of all time in the Designer's Creation. It's so amazing I won't give spoilers.

Depth First Search:
Head the self-explanatory way down to the bottom-left of Binary Woods.

It's very hot in the lab:
There are four microscopic nuclear things in the waters of Too Much Water. They're hidden, and they're at the following locations relative to the scientist:
Five North
Six East
Three South, one East
Four West, four South

The best Pokémon:
In Southern Passage, talk to the lady giving a survey. Only say yes to Dunsparce being the best. She'll give you a Nugget as a reward for being right.

Reunited:
Discover their loveless marriage by cutting down the trees in front of the Glitched House and going inside.

Demisemihemidemisemiquaver:
See "The best Pokémon" for getting the Nugget.

After finding the nugget, head to the Shady Merchant and trade for the Poké Flute.

Then, enter the bottom left house of the Southern Encampment and hand the man the Poké Flute.

Overhead view is cheating:
This requires Cut from "To the South!"

At Yet Another Route, near the entrance, go to the switches. Hit them in the order:
Bottom, Top, Middle

Cut one of the trees and talk to the guy to be accused of filthy cheating. I wouldn't call it cheating, I'd call it having a superpower, personally.

Yet Another Secret:
This requires Surf from "To the South!"

From Yet Another Path, Surf East to the tides. Take the third tile from the bottom. At each stop, go down then left. Continue East to Yet Another Secret to unlock yet another achievement.

Recursive Madness:
Head to the Path of Recursion straight East from Central Square. Talk to the lady and keep answering yes to learn more about recursion.

Fossilized Trio 1: Anarchy:
"Femboy Supremacy" is required before doing this one. Make sure you're all dressed up in your programmer socks before attempting.

On the way to the Worried Explorer, you'll have to solve the Shifting Grove puzzle. The solution is:
Bottom Left x1, Top Left x2

From the Worried Explorer, we're going to each adjacent area in the order of North, South, then West. Once you do the required action, you'll know you did it right:
North, Strange Pillar: Walk in and out of the hole in the fence five times
South, Strange Bush: Walk exactly 20 steps before talking to the bush.
West, Strange Sign: Be a femboy and talk to the sign.

After that, the Helix Fossil should be yours.

Fossilized Trio II: Democracy:
Dome Fossil is found in Altering Cave. The interior of the cave changes every hour. See "Feels Different" for the order of locations.

Fossilized Trio III: OLDEN:
Requires Surf from "To the North!"

Head South from the Guard Post to Tiny Cavern. Head inside and talk to the scientist there to obtain the Amber Fossil.

Fossilized Trio IV: In the name of science:
Turn-in for the fossils is in the Western Encampment, the Explorer Syndicate building. Talk to the ~~scientist~~ doctor in the top right.

Balls I: Growing a pair:
Balls II: Growing two pairs:
Balls III: Baller crawler:
Balls IV: I've seen it all:
The nine pokeballs are found in the following locations:
1. Shimmering Pass, bottom left. Talk to the guy walking around and he'll give it to you.
2. The Harbor, there's a moving Poké Ball across the water on a platform. Use Surf to get to it.
3. Road of Ultimate, on the other side of the South pond.
4. Yet Another Cave, invisible, in the bottom left corner between the two adjacent flowers.
5. Broken Bridge, Surf South to find a Poké Ball off the beaten path.
6. Binary Woods main, top left, past a little surf pond.
7. Jumpity Jump, bottom left, talk to the Poké Mart employee.
8. See "Femboy Supremacy".
9. See "Braille Happens".

Return to Glitchland: Main Story Reimagined
Okay Folks:
Requires Strength from "To the East!"

Head to the mansion and inspect the locked door inside. Once you've done that, head to the Western Camp and enter the top right house. Once you've talked to Mr. Lockpicking Lawyer, head back to the mansion and talk to him to get him to open the door.

Once you reach the locked door, inspect the broken door to the right of the closed gate, then progress to Mr. Lockpick Lawyer and talk to him again to open the door. Proceed in the final room to recieve the heartscale for "Femboy Supremacy".

Hack and Slash:
Requires Cut from "To the South!"

Go to the Grasslands and cut down all the trees before talking to karate-man.

RYDEL RYDEL RYDEL RYDEL:
Obtain the Bike Voucher from the fisherman in the house at The Harbor by telling him you like to fish. The Bike Shop is in the West.

Interjection:
Requires Surf from "To the North!"

First, make sure to grab Pot of Greed from the guy at the top right of Glitchland Library.

Then, head South from Southern Camp and interact with the door of Team Magma HQ to draw two additional cards from your deck and add them to your hand.

Once inside, interact with the computer in the top left corner. Show your understanding of dependency chains by answering yes to the girl's question.

Femboy Supremacy:
Requires "Okay Folks" first to complete, as you need the Heart Scale reward. With that out of the way, we can finally be the cutest there ever was.

At Southern Camp, in the bottom right house is where the femboy lives. Gift them a heartscale and they'll teach you the secrets of beautiful femininity.

At Eastern Camp, in the bottom left house, talk to the guy, and he'll give you his ball for being so cute.

Braille Happens:
We can piece this solution together from translating all the braille we find throughout the world:
Western Relic: trendy phrase
Altering Cave Boulders: hope
Altering Cave Cave: less
Tiny Cavern: disa
Altering Cave Dome Fossil: ster
Trendy Phrase: hopeless disaster

Going to the trendsetter and attempting to set the phrase to hopeless disaster causes the world to shake with your edginess, making a Poké Ball appear in Western Relic.

Game Corner
Gambling I: So it begins...:
Lottery I: Suboptimal RNG:
You're practically guaranteed to get these so long as you have a decent length name and do the lottery at least once.

Gambling II: We have a winner!:
Otherwise known as CCR0. We're not actually gambling, especially because I can only assume some of these machines are rigged/lucky like in the base game.

Open up a slot machine and then open mGBA's memory searcher. Initial value of coins is 200, so search for that.

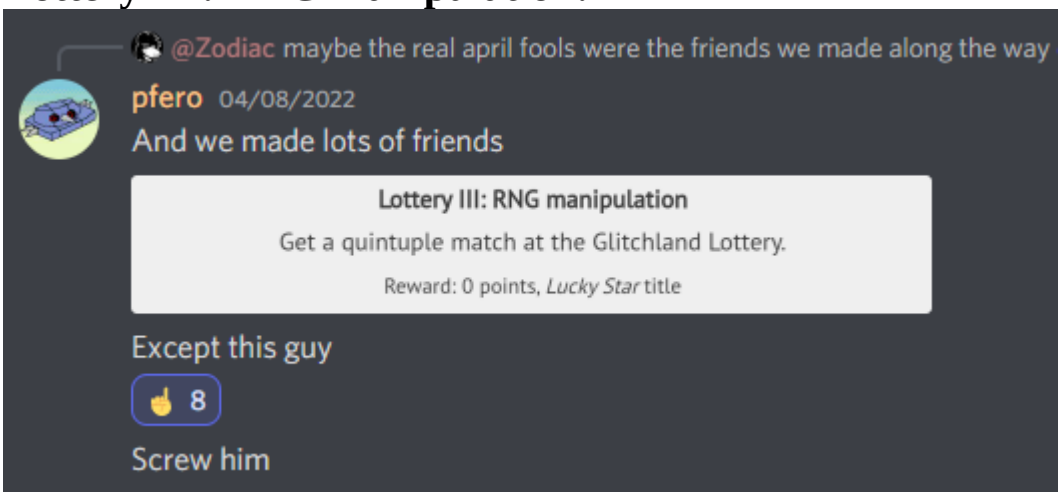
Spend any amount and finish the reels, then search for coin values at the new number. Five entries should show up at **0x02002DEC**, **0x02020A60**, **0x02020AA4**, **0x02020AE8**, and **0x02020B2C**. Scattershot change them all to 600 or something else similarly above the required amount. It should reward you the achievement after leaving the slot machine.

YEET I: Welcome to YEET:
YEET II: Enemy at First YEET:
YEET III: Living in the YEET:

There are five YEET's. Each YEET cycles every 4 hours starting at Midnight UTC, and an NPC says that if you go through a cycle of 10, you should see all 5.

Here are all 5 with a general description of how to solve them:
Yeetordle: It's Wordle. I used an online solver to get First Blood on this one. I'm bad at this and I have no shame.
Out of Sight Out of Mind: Test your knowledge of spinner mechanics. Honestly, this felt like the most straightforward one. The only hitch is making sure you're patient and not running unless it's the absolutely required time going through the narrow pass with two alcoves.
Slide Freely Everywhere: Basic slide puzzle.
First area: Go in the block on the right, then **Right**, **Down**, **Left**, and **Up**.
South area: Enter from right, then **Left**, **Up**, **Left**, **Down**, **Right**, **Down**, **Left**, **Down**. Grab the scroll and head **Up**, **Right**, **Down**, **Right**, **Up**, **Left**, **Up** to leave.
North area: Enter from the right block and go **Right**, **Up**, **Left**, **Down**, **Left**, **Up**, **Right**, **Down**, **Left**, **Up**, **Left**, **Down**, **Left**.
Be Boulder About It: This one requires Strength. It will test your knowledge of block pushing and character movement. Honestly, I have no idea how to actually do most of this and brute forced a lot of it. My main advice is for the second to last section, mashing the boulder left through the lower path where the statues move is required to block the second boulder on the top row, so you collide into both boulders and don't shift them out of the way before moving up. Don't just hold left, as the statue will move in the way and block your progress. Mash the input.
Puzzling Delerians: The pool for questions seems to be relatively small, so here's the ones I got:
"sooo.. my hometown.. is it in south part?" No
"the map of designer vERY BIG!! but.. there.. was five of signs?" Yes
"so in north.. they as for pokemans.! Any of them is grass type yes?" Yes
"altering cave has 3 forms yes?" Yes
"in old glitchland... very VERY old! It was 4F item to teleport??" No
"in glitchland.. yet another maps.. there 3 of them right??" No
"this person of boardwalk.. asking about star! they is girl?" Yes
"yay.. the north.. very big.. vERY! but is east smolest?" No
"crossroads in north very cool map! but has it three peopel stand?" No
"in current library.. are 14 bookshelf?" Yes
"is there any mon in houses of south?" No
"sooo.. that boy who plays flute.. is it in south part?" Yes
"sooo.. in all north towns.. is there 5 house?" No
"are here 3 peopel in the trendy hous ??" Yes

Lottery II: On WR pace:
Lottery III: RNG manipulation:



I got nothing for these ones. It seems like the save file sends two words to the client, which is just a "give me lottery results" message.

xfix's theory crafting was you could maybe calculate the RNG from the 2018 server's source code and mark your calendar for when to get your lotto match, but I've done no research on this and am only sharing heresay.

Cracker Cavern Reborn Prep:

Become permanent girl:

First, how to permanently be a girl so we don't have to keep getting dressed up.

Kirby703 pointed out that she only had to modfy two addresses in the save file to force the May model after logging in:

`0xE008: 00->01`
`0xEFF6: 0C->0D`

After that, you should be able to quickly recieve sweet euphoria.

Installing GDB and compiling pokeemerald for symbols:

Second, we want to get a symbol table for GDB, which the process of will also give us our GDB install. Overall, the [pokeemerald install.md] has most of the instructions we need already.

I used a WSL1 Archlinux to install DevKitARM. No version of GDB except for the **`arm-none-eabi-gdb`** that's packaged with DevKitARM hase worked with the generated Emerald elf file for me.

Make sure when you actually get to compiling pokeemerald that you're making with the flag **`DINFO=1`**, otherwise, no symbols or source mapping.

Once you have **`pokeemerald.elf`**, launch **`arm-none-eabi-gdb`** with the elf file as an argument, then connect to the mGBA server.

I couldn't get mGBA's GDB server to connect with the GDB client through localhost, so I had to point both ends to my actual local IP address. I also had to add a special exception to my firewall for my WSL1 install. Be aware of these hitches in a Windows environment.

All relevant code for the event, as far as I'm aware, is THUMB. I'd recommend using the command **`set arm force-mode thumb`** so no hitches occur trying to view the disassembly and code.

Cracker Cavern Reborn:

Hacking I: Walking through walls:

Our goal is to get through the rock wall and reach the next level. Thankfully, there's a man that's nice enough here to open the way, if we have a Luxury Ball.

I feel like the solution here is simple enough. If we're going to be using the big girl tools later, may as well start here. Use GDB to stop execution and use the **`finish`** command until we're in **`AgbMain()`**. Then, call **`AddBagItem()`** using the following command:
`call AddBagItem(0xB, 1)`

`0xB` is the hex value of the luxury ball, and the **`1`** just means we want one.

While this throws a **`SIGILL, Illegal instruction`** after we try and continue, the game doesn't crash after continuing a few more times, and we now have a luxury ball.

Runea gave me an alternate solution involving some old Gameshark codes that DOESN'T throw an illegal instruction, in case you can't get the previous one to work. The codes are as follows:

```
Master code:
9266FA6C97BD
905B5ED35F81
B76A68E5FAB1

Balls in shop:
5FC4BF056CCE
10D97E935CBE
70BBB571DD48

Open shop (L+A+Select):
AE44960E2DD6
99468975ECD8
AE44960E2DD6
FF62825EBC40
AE44960E2DD6
805C1D31C295
30BA206B961F
79C4A3076CDE
79BA7465DC00
```

After entering the previous codes, while in Cavern Depths I, hit L+A+Select to open the shop and buy a luxury ball.

Talking to the guy opens the way and we can move on to challenge 2!

Hacking II: Going places:

This one was just a hail mary guess for me that saved a bunch of time. Going off the assumption that all map ID's are the same as 2018's event, inside Cavern Depths I, search for the memory value **`0x318C`**, which is the map ID of Cavern Depths II. From this location, four different addresses show up, but what we're looking for is the first one, **`0x020181F4`**, which is the memory address for the warp data of the ladder going down. Modify it to **`0x1337`**, which, since little endian, reads as **`37 13`** in the memory viewer with one byte alignment.

Hacking III: Scripting:

Here's where the headaches and the time sinks begin. As a baseline, our goal is to find the password and tell it to the guy, and he'll open the way through the rocks to the next level.

Starting off, since password verification is done through talking to an NPC, I started doing some research. To preface, I have no experience with gen 3 glitches and the game's technical aspects before I started this part of the event. I've been guessing my whole way through up to here.

With some basic research and intuition, tutorials on romhacking tell me that most everything in the overworld related to NPC's is handled by scripts attached to the NPC's directly.

Doing some research on the scripting engine, I discovered that a function is called every time a command is executed, RunScriptCommand(). Setting a breakpoint on that function in GDB, when talking to the guy, it immediately breaks, and we can dump the script context pointer with **`p/x ctx->scriptPtr`**, finding our entrypoint of **`0x020182AD`**.

I started by hand disassembling small parts of the starting function to verify I understood the format of the scripting language, before realizing that the sheer bulk of everything was quickly spiraling out of control. So, I comandeered my old Gameboy Disassembler and quickly and sloppily repurposed it as an Emerald Script Disassembler, adding in script codes as I found new, untranslated parts of the script. With this new tool and a lot of patience to comment and sift through it, I finally ended up with the script file I placed in the CCR3 folder.

The core of the password script is a hash function, going over the first ten letters once each, passing over the whole buffer twice, ending up with two separate hash values to compare against. The last five places of the input are not even considered as a part of the password.

It appears that the main script is a loop that's simply been unrolled all the way out, and the main core of each loop is as follows in pseudoscript:

```
addvar                      destination:0x8001, value:CONST
loadbytefromptr             destIndex:0x00, source:buffer[index]
setptrbyte                  srcIndex:0x00, destination:following_command_operand
addvar                      destination:0x8001, value:CONST
loadbytefromptr             destIndex:0x00, source:var_0x8001_least_significant_byte
setptrbyte                  srcIndex:0x00, destination:loop_addvar_least_significant_byte
loadbytefromptr             destIndex:0x00, source:var_0x8001_most_significant_byte
setptrbyte                  srcIndex:0x00, destination:loop_addvar_most_significant_byte
setvar                      destination:0x8001, value:0x0000
setvar                      destination:0x8002, value:0x0049     ; Loop counter
Loop:
addvar                      destination:0x8001, value:???        ; Assigned earlier
subvar                      destination:0x8002, value:0x0001     ; Sutract loop counter
compare_var_to_value        var:0x8002, value:0x0000             ; Loop check
goto_if                     condition:0x05, destination:Loop     ; Condition 0x05 is !=
```

Translated into C pseudocode:

```
increment = CONST + buffer[index];
iterations = CONST;
do {
        var_8001 += increment;
        var_8002--;
}while(var_8002 != 0);
```

And since this is just repeated addition, we can further optimize the do-while loop into simply the following:

```
var_8001 += increment;
var_8001 *= iterations;
```

After it passes over the 10th character in the buffer with this, it'll store the hash in **0x8003** for later comparison and start over. After finishing the 20th iteration, it will then check the hash in **0x8003** for the value of **0xB0EF** and the hash in **0x8001** for the value of **0xD4B9**.

So with this information in hand I got to work on my bruteforcer, and after two days of heartbreak, bad assumptions, and not understanding why it didn't work, I fixed my bugs and got a good match in a couple minutes:

Hex: BA, B6, A4, D9, DE, A1, 00, 00, 00, 00
Characters: /♀3ej0____

Note the four trailing spaces. Leaving the bruteforcer on for a couple days revealed hundreds of passwords, probably thousands, but I didn't bother keeping track. It's a very weak hash and it's not surprising that there are so many matches.

After entering this password, we can now pass the rocks and head to the real deal big girl final challenge:

Hacking IV: Loss of integrity:

This is the part where I cried for a variety of reasons, both good and bad.

Our mission, if we choose to accept it, is to forge a Gold Certificate of Commendation. The issue is the person only wants to give us Silver Certificates of Commendation. The given tickets are encrypted with "military-grade AES encryption", which just means 256-bit. We're teased that there might be a weakness somewhere, and that's it as far as hints.

Considering I'm completely lost for this, I figured a good place to start would be the script entry point again, since I already have my script disassembler. Even if the answer isn't found there, I should at least be able to figure out how communication works and how certificate generation works.

Well, upon disassembling the script at **0x020182B5**, I find there's not much I can work with there in particular, as most of the work seems to be done in the assembly. So, I get to work learning myself some ARM THUMB assembly from zero.

So I start working on disassembling with help from GDB's **examine** instruction, piecing together all 600 lines of assembly that make up the CCR4 specific functions and wireless communication in general, and lottery and YEET's for good measure.

On further analysis of the disassembly, the first assembly function called to by the scripts is setting the request buffer length, the send buffer, the request number, a request constant, and a recieve buffer. For certificate generation, the send/recieve buffers are **0x020186D0** and **0x020186F4** respectively, and for certificate verification, the send/recieve buffers are **0x020186F0** and **0x02021DC0** respectively. The request number is appended to the beginning of the send buffer, and the rest of the buffer length is used for the actual data to be sent. Every section of the buffer is a 32-bit word.

With a little bit more investigation and intuition, it's pretty easy to figure out by looking at the memory viewer that after the request number, **0x020186D4** contains an ASCII buffer with the following contents:
holder=Aelita/type=silverÿ

It also has a padding of **C0 46** in byte order after that to fill the word, but the most important part is the **FF** terminator at the end, represented by the **ÿ**.

0x020186F4 is where the certificate is recieved. Analyzing it in the memory viewer as well shows a seemingly random string of letters, numbers, slashes, and pluses, with some equal signs at the end. After literally googling "text string padded with equal sign", I figure out this is a Base64 encoded string. Using some online tools to decode it shows that it's a set of 64 hex bytes.

On verification, the certificate is sent in full from **0x020186F4**, along with a long static buffer, and in return, an ASCII text string is recieved at **0x02021DC0**, before quickly being converted to Pokemon Emerald's text encoding and being displayed to the player... sometimes. It tends to fail displaying the serial which corrupts the rest of the text box given to the player, so it's easier to view it in memory and try and catch it before it gets converted.

Now, we always try the easy solutions first in this house, so I tried modifying the buffer sent to the server. Simply changing the type to gold doesn't work, as it will instantly reject anything requested other than a silver certificate. However, I find I can freely change the name of the holder to whatever I want.

So with the certificate generation happening server side, and the only thing I have access to being the cleartext and the cipher text, I spend days doing vague research over the internet and trying to amass knowledge about AES and general methods of attacking encryption. While this all makes me feel really dumb and I have a bad cry during this, I keep pushing through, and a few days in, I finally feel like I'm starting to learn something.

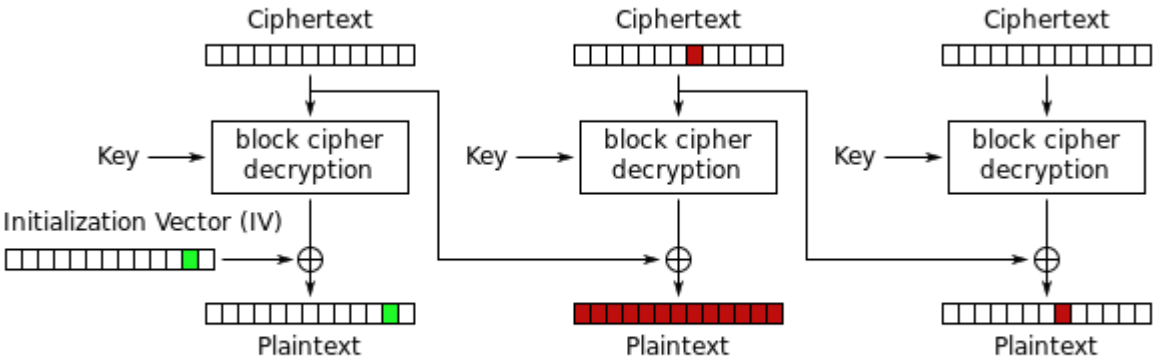
After a few days of light investigation, I come back to the buffers and take another look, and I realized the certificate buffer always has the same first block. This gets the gears turning a bit, when I stumble across this very helpful video describing an attack on ECB ciphers that puts AES into a more understandable context for me. Looking at these two plaintexts, splitting them up into 16 byte blocks as per the AES standard block size:

authority=Cracke	**authority=Cracke**
rFour/serial=828	**rFour/serial=786**
7751/holder=Aeli	**1741/holder=Aeli**
ta/type=silverÿ	**ta/type=silverÿ**

If this encryption algorithm is using ECB, since each block is encrypted separately with nothing touching another block, the first and fourth block of the ciphertext should be identical between requests, if the same key is used to encrypt every time. So testing that theory, here are two different requests decoded from Base64 to hex:

```
213227A7E85C4B07CFD82E6BBD3032CA    213227A7E85C4B07CFD82E6BBD3032CA
8C16585AF0B74CED43BCAF06CD308073    71465B541F2B8CCA471614C639725991
6AF6A1E25C119BB85FDF6F6B7FA35F0D    14C49B808435B685C3A7BB029DDA43EF
823DA46938D6BD91C70D6379B86A95D0    4E27AC4F9B254B39757F75665C315654
```

As we can see, the first block matches, but none of the others do. While this eliminates the possibility of an ECB attack, the gears are spinning really fast now, and now I'm pointed in the direction of checking other modes of operation for AES block ciphers. A little bit of extra research on the next mode of operation on Wikipedia and doing some web searches leads me to [this Medium article describing a bit-flipping attack], which I will now lovingly steal the diagram from:



Cipher Block Chaining (CBC) mode decryption

Essentially, because CBC uses the previous ciphertext as part of the decryption at the last step, any change to the previous ciphertext propogates to the current plaintext and no further. This causes the previous block to become garbage data, but the plaintext we're targetting can be arbitrarily changed, if we know the plaintext, which we seem to.

I figure I have nothing to lose and start doing my best to see if I can get any modifications of the third block to come through to the fourth, and sure enough, I start recieving invalid certificate type errors in response with one letter of the type changed, in the byte I would expect.

So after a large amount of experimentation, I figure the best way to utilize this will be to try to extend this from a 4 block certificate to a 5 block by expanding the holder name. We can do this by simply editting the plaintext buffer held at **`0x020186D4`** to expand it to the proper size, and by making sure we edit the send buffer size at **`0x02018236`** to **`0B`** for the setup asm to encompass the extra words that should now be in the request. This will push our ASCII into the recieved cert buffer, but that's fine if the end of our request gets overwritten, since we won't need it after it's sent once. So, our request should end up looking something like this:

```
holder=Aelitaaaaaaaaaaaaaa/type=silverÿ
```

We then recieve the following plaintext and decoded ciphertext in response:

authority=Cracke	**213227A7E85C4B07CFD82E6BBD3032CA**
rFour/serial=679	**AAA51E870989559DBF1F508159D8CD85**
4486/holder=Aeli	**A98E745AE997D0B27A6B23E6333AB1D3**
taaaaaaaaaaaaaaa	**9B7B0BBAA395123335ED6732749BA1BF**
/type=silverÿ	**70D6591428C75C5BC936B43FE24DE864**

NOW we're getting somewhere. By XORing the known ASCII value in the plaintext with the desired character, then XORing that with the corresponding byte in the previous ciphertext and replacing it, we can arbitrarily modify the final block. So, the fourth block should change like so:

Original: 9B7B0BBAA395 123335ED6732 749BA1BF
Modified: 9B7B0BBAA395 063535FFFDBF 749BA1BF

This should change the final line of the certificate to read:

/type=goldÿÿÿ

And on reencoding this string to Base64 and sending it in for verification:

We did it, Reddit!

And when I descended into the Depths and the sappy music started playing after talking to your dad, I burst into tears of joy for finally finishing this after all the struggle. I feel like I learned a lot and I'm super grateful for being able to do all of this, despite any pain it may have caused my sanity along the way.

Final thoughts:

Honestly, I don't have many other than joy after being done with all of this. I always really appreciate being given the opportunity to participate in the event and being able to learn things and flex some knowledge. I rarely find the motivation to do anything programming or technology related in my spare time, so it's always nice to be given something like this on a silver platter, especially this year, coming at a perfect time when I wasn't sure what I should do with my time.

It's also nice to be reminded that there are great people out there willing to teach and help, and that I have people that I do consider friends, even if I feel like a very shallow and distant person sometimes.

If you read this far, thank you for giving so much attention to something I put a lot of sweat, tears, and love into, and I hope you have a wonderful day. c: