



## Swagshop writeup

Hack the Box

<https://www.hackthebox.eu/home/machines/profile/188>

*Report for Fontys Hogeschool ICT*

Jort Geurts — i360843 — 3007235

30/09/2019

# Contents

<b>1</b>	<b>Basic information</b>	<b>3</b>
<b>2</b>	<b>Basic recon</b>	<b>4</b>
<b>3</b>	<b>Exploit 1</b>	<b>6</b>
3.1	Uploading the script . . . . .	6
<b>4</b>	<b>Exploit 2</b>	<b>8</b>
<b>5</b>	<b>The script</b>	<b>10</b>
<b>6</b>	<b>The flags</b>	<b>12</b>
6.1	Root . . . . .	12

# Glossary

**CMS** Content Management System. 4, 6

**CVE** The Common Vulnerabilities and Exposures system is a way to responsibly document and provide reference for vulnerabilities found in public systems.. 4

**RCE** Remote Code Execution. 6

**SQLi** SQL injection. 4

# Chapter 1

## Basic information

The specs of this box are:

Name:	SwagShop
OS:	Linux
Difficulty:	Easy
Points:	20
Release:	11 May 2019
Retirement:	28 September 2019
IP:	10.10.10.140
Retired:	Yes
URL:	<a href="https://www.hackthebox.eu/home/machines/profile/188">https://www.hackthebox.eu/home/machines/profile/188</a>

Time until user pwn:  $\pm 10$  hours

Time until root pwn:  $\pm 40$  hours (including scripting the user shell)

User flag: `a448877277e82f05e5ddf9f90aefbac8`

Root flag: `c2b087d66e14a652a3b86a130ac56721`

## Chapter 2

# Basic recon

As per usual when you get a box, the first step is to find as much information about it as you can. A quick `nmap` reveals that port 80 and 22 are open. Http and ssh. Going to the ip in a browser shows a Magento webshop, with three items pre-configured. After some basic SQL injection (SQLi) attempts, I decide that it's probably gonna be a bit more complex than that.

Then the hunt starts. The specs of the box say that CVEs are a big part of this box, so I start looking around for relevant CVEs. Not knowing what version of Magento is used, it was kind of a spray method. I noticed that the copyright was on 2014, so it's very likely that it's an older version.

There are two Magento versions, v1 and v2. I was still unsure what version was used, so I was just trying some exploits I could find. The exploits on <https://www.exploit-db.com/> turned out to be really useful, and it had a script (#37977) to get a login on the admin dashboard. This script worked, and the admin page was on the default location (`10.10.10.140/index.php/admin`).

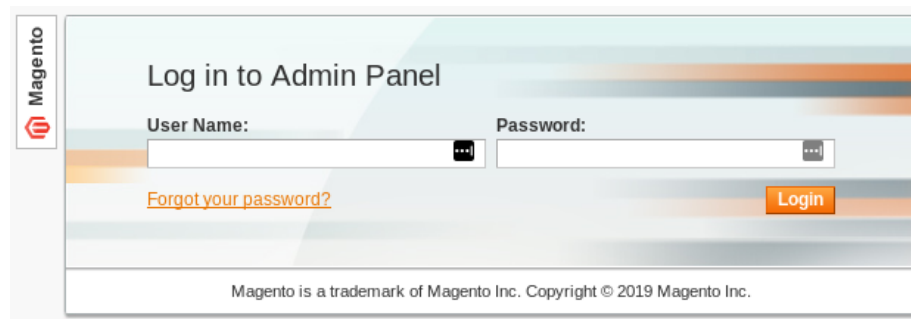


Figure 2.1: *The login page of Magento*

The admin page turned out to contain the version number of Magento, v. This meant I could pin my exploit search to a specific version. After searching through the Content Management System (CMS) for quite some time for interesting ways in, I was unsure where the exploit would be.

There was a lot written about exploiting a downloader included in Magento, but, after some reading on the HTB forums, I found out that this downloader was disabled because it was an unintentional way in.

Then I looked at exploit #37811, hoping this would be it. I found

```
root@kali:~/Downloads/swagshop# python 37811.py http://10.10.10.140/index.php/ad
min "uname -a"
Traceback (most recent call last):
  File "37811.py", line 70, in <module>
    tunnel = tunnel.group(1)
AttributeError: 'NoneType' object has no attribute 'group'
```

Figure 2.2: *The python script crashing*

<http://10.10.10.140/app/etc/local.xml>, giving me the install date, which could be used in the script. However, after running, with some slight modifications I got it to a point where it would crash with the error seen in fig. 3.1. Searching for this error actually brings up the HTB forums for this challenge, stating that I should stop looking. It was apparently not the way forward.

## Chapter 3

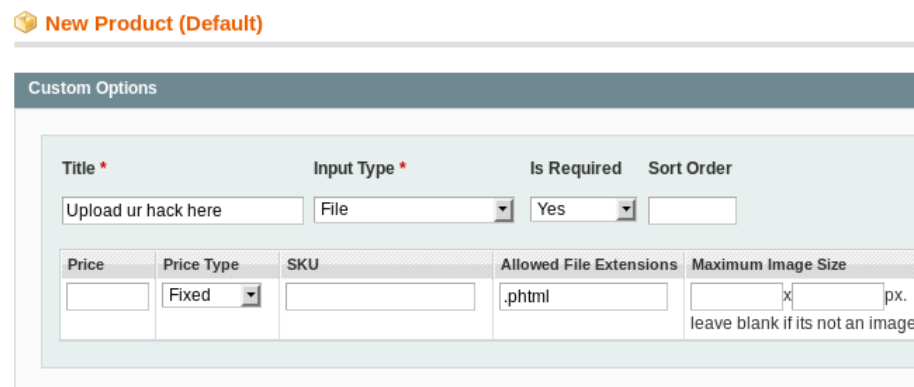
# Exploit 1

After a lot of DuckDuckGo-ing, I found the blogpost which ended up giving a solution. The blog can be found here: <https://blog.scr.t.ch/2019/01/24/magento-rce-local-file-read-with-low-privilege-admin-rights/>. Recently, a Remote Code Execution (RCE) was found which abused the way custom XML can be used in Magento. In this case, the path to uploading the malicious PHP script and executing it is divided into two steps.

### 3.1 Uploading the script

The python script mentioned in chapter 2 was used to gain admin access to the CMS. I then followed the following steps:

Firstly, I created a new product.<sup>1</sup> A 'Simple Product' is enough. I entered some dummy information to fill in all the fields, because the product itself doesn't matter. But, there is a way to add an upload form to a product. This is our way in.

The image shows the 'New Product (Default)' form in Magento. It has a 'Custom Options' section with a table for adding options. The table has columns: Title, Input Type, Is Required, Sort Order, Price, Price Type, SKU, Allowed File Extensions, and Maximum Image Size. One option is already added with the title 'Upload ur hack here', input type 'File', and 'Is Required' set to 'Yes'. Below the table, there are fields for 'Price', 'Price Type' (set to 'Fixed'), 'SKU', 'Allowed File Extensions' (set to '.phtml'), and 'Maximum Image Size' (with a note to leave blank if not an image).

Title *	Input Type *	Is Required	Sort Order
Upload ur hack here	File	Yes	

Price	Price Type	SKU	Allowed File Extensions	Maximum Image Size
	Fixed		.phtml	px. leave blank if its not an image

Figure 3.1: *The way to add an upload form to a product page.*

So, now that we have a way to upload our reverse shell PHP file to the server, we need to trick the server into executing it. The aforementioned blogpost does this via an XML trick.

<sup>1</sup>Catalog > Manage Products > Add Product

Here I deviate from the blog post a little. In the used version of Magento I couldn't find the mentioned 'design tab' on the product page, so I started to look for other ways that I could give custom **Layout Update XML**. I found that creating new page<sup>2</sup> also provided the option to give custom XML<sup>3</sup>.

Then I had to figure out where the uploaded **.phtml** file actually was placed on the filesystem. I made a guess that it'd be the default installation, so **/var/www/html**. This was correct, and I obtained my reverse shell using `nc -v -n -l -p 10365`.

However, then disaster struck. I had obtained the reverse shell, but after I submitted the user flag, I had left it alone. I hadn't documented the exact XML I used, and the server was reset by someone else. I had lost my reverse shell. I tried to do every step again, but I couldn't get it to work again. To this day I have no clue what was different in that specific instance of the server. I suspect another user had already changed some settings, which allowed this hack to work.

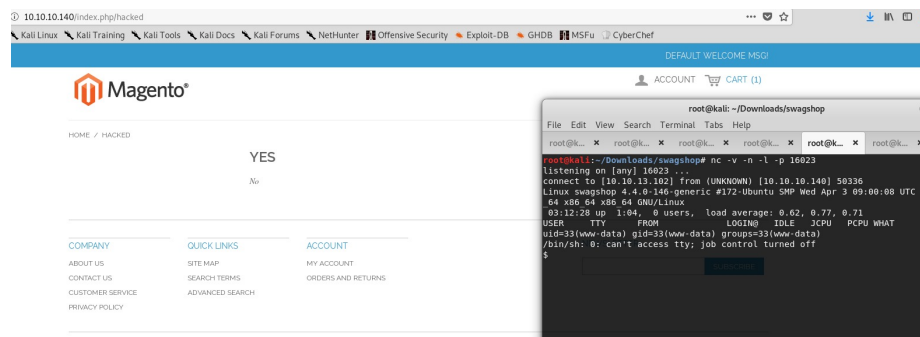


Figure 3.2: Screenshot of the initial shell I obtained.

<sup>2</sup>CMS > Pages

<sup>3</sup>CMS > Pages > New Page > Design > Page Layout



## Chapter 4

# Exploit 2

So with the first exploit no longer working, I was determined to find another way. I started to message around on the HTB forums, to see if someone had found the same way; and could tell me what I was doing wrong. I got a response outlining another way to get a reverse shell into the server, so I started to work on that.

The blogpost from the first exploit also mentioned another file-read exploit with the email templates. However, there is another issue with the newsletter templates. Apparently, they are also capable of executing `.phtml` files.

The initial exploit to gain admin access was the same, as I already expected. However, he (ab)uses the image-upload functionality given when you create a category. The instructions I got were quite bare, so I had to do some figuring out on how to exactly exploit this.

I started with a lot of DuckDuckGo-ing, figuring out how to include a malicious PHP script into an image. I found that there are two main<sup>1</sup> ways:

- Firstly, you can add a small PHP script to the EXIF data of the image, basically giving it a description with a PHP script. The issue with this method is that this doesn't give a full reverse shell.
- Secondly, you can rename the PHP script, and try to trick the website into thinking you uploaded an image; whilst in reality it's a script. This can give a proper full shell, since you can upload any PHP script this way.

So I decided to start with the second option, since it seemed like the easier one. I copied my reverse shell script a few times, using some of the suggestions from the blogpost found in the footnote. I uploaded them, but I wasn't sure if I'd get feedback from the site if it succeeded. Then I started to doubt whether the traversal from the second step would even work, so I used the cute cat picture seen in fig. 4.1 for a test.

Doing this, I found out that this trick indeed actually works. The raw image is displayed, indicating



Figure 4.1: *The cute cat picture I used for testing.*

---

<sup>1</sup>More information and two other ways:  
<https://hackers2devnull.blogspot.com/2013/05/how-to-shell-server-via-image-upload.html>

to me that this might be susceptible to executing code. After this I started to try more strange extensions, finding out that `.php.jpg` worked. I knew this, because it showed up after I uploaded the file.

On to the second stage, the execution of the script. I got hinted towards another issue with the newsletter templates. Including the following code would allow me to traverse to the saved PHP script, and it should be executed via the 'Preview Template':

```
{{block type='core/template' template='.././.././.././../med_
↪ ia/catalog/category/reverse_shell.php.jpg}}}
```

Another setting that needed to be changed was to allow symlinks. This setting can be found under **System > Developer > Template Settings**. I suspect this allows the path traversing that is used in the path to the 'image'.

The template could then be executed via the 'Preview Template' option. This way, the server would show an iframe with the email content; including the 'image'. The page would show an infinite loading screen, which is normal behavior for the reverse shell script.

## Chapter 5

# The script

So, this box has a tendency to be resetted a lot. On average, this machine gets reset over a hundred times a day. Apart from this, **script kiddies** keep launching automated attacks, crippling the server. This makes it really difficult to keep your shell on the machine. So, as part of the requirement **Automating offensive security tasks.**, I decided to write a custom Python script, to automatically re-gain my shell.

This was easier said than done. I knew the steps that I had to take, but in this total undertaking took me about a week to complete. This was partly due to my inexperience in Python, partly due to weird documentation of the scraping plugin, **BeautifulSoup** and partly due to some bad API behavior of Magento.

I am aware that the code quality isn't the highest. Some things could be more fail-proof, tho I've tried to write the code flexible enough that it should work on other machines with the same vulnerable Magento version. Also, the script can be ran multiple times on the same machine without any negative effect.

This is because all the custom things I upload are using UUID's as filenames, meaning they're practically unique. This way there are no collisions in namings on the server, with the newly created category, uploaded exploit and created newsletter template.

In broad lines, the script does the following:

- Uses the **37977.py** script to gain admin credentials to the server.
- Logs in to the server, and using the **Session** functionality from the **requests** package; I'm able to keep the session going between requests.
- Disables the url-keys. This means the requests don't have to be appended with a random key. This has the advantage that I don't need to scrape myself through the site, but I can just directly make the requests I need.
- Enabling symlinks, this is a requirement for the traversal in a later stage of the exploit.
- Next, I create the category with an unique UUID, and upload the PHP exploit.

```

1      multipart_form_data = {
2          'form_key': (None, system_config_form_key),
3          'groups[security][fields][use_form_key][value]':
4              ↪ (None, 0)
5      }
6
7      response = session.post(url +
8          ↪ '/admin/system_config/save/section/admin/key/' +
9          ↪ urlKey + '/', files=multipart_form_data)

```

Listing 1: *Python code used to make a valid request to change a setting (disabling the url-keys).*

- Then, I have to create the newsletter template, with the right code inside referring to the uploaded PHP exploit from the previous step.
- Finally, the newsletter template needs to be 'previewed' to run the exploit.

As with every piece of code, there were a thousand different challenges involved in this. Traversing through the scraped html code was a huge challenge. Since this is so error-prone, I wanted to do this with as much built-in flexibility as possible.

Another thing that was new to me, is that all the settings are sent to the server in a **multipart form data** form. I had never heard of this, and had to research how this method of transferring data works, and how to do it in Python.

The server refused to apply the settings if the request contained a **filename** attribute, so I had to strip that out. Figuring this out, and figuring out how to do it was already a few hours down the drain. In the end, the solution ended up being a structure like this:

The **'thing': (None, value)** ended up being the solution. The **None** apparently means that it isn't a file, meaning that it won't send the **filename** attribute.

## Chapter 6

# The flags

So, now that I have a reverse shell; how do I get the user and root flag? Luckily this first is rather easy. The reverse shell already gives me access to an user, `www-data`. All that was left for this flag was to traverse to the `/user/harim` folder, which would hold the `user.txt` flag.

One flag down, one flag to go.

### 6.1 Root

The search for root started with the `sudo -l` command. This listed that the current user (`www-data`) had rights to use sudo without a password for `/usr/bin/vi` and `/var/www/html/*`. Vi being the famous text editor, and the html folder in which the Magento installation resides.

```
sudo -l
Matching Defaults entries for www-data on swagshop:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/snap/bin

User www-data may run the following commands on swagshop:
    (root) NOPASSWD: /usr/bin/vi /var/www/html/*
```

Figure 6.1: *The output of `sudo -l`.*

I figured this was no coincidence, and this would be the pivoting point to root. After some duckduckgo-ing around, I found an article on Medium<sup>1</sup> about using Vim for privilege escalation. The user in the screenshots was `haris`, which I suspect is actually this box.

The final example is about vim, and the output of `sudo -l` actually looked really similar to mine. But this is where I went wrong, because the command `sudo vi` didn't work. So I thought this was wrong, and started looking for other ways in.

In the end, I only had a few hours left before the box would be retired to find the root flag. I started to ask around on the Hack the Box forums, hoping to find some help to find the root flag. I got again nudged towards the sudoers file and Vim, which confused me. So, after a lot of playing around I found what

---

<sup>1</sup><https://link.medium.com/0dwd0VgL7Z>

I did wrong.

`sudo vi` doesn't work. `sudo /usr/bin/vi` does.

So, simple solution right? Just open the `/root/root.txt` file with `sudo /usr/bin/vi` and you have your flag. No. This actually was not allowed for some reason. So, what I ended up doing was copying the `/root/root.txt` file to the `/var/www/html` folder, and opening it there. This ended up giving me the root flag.

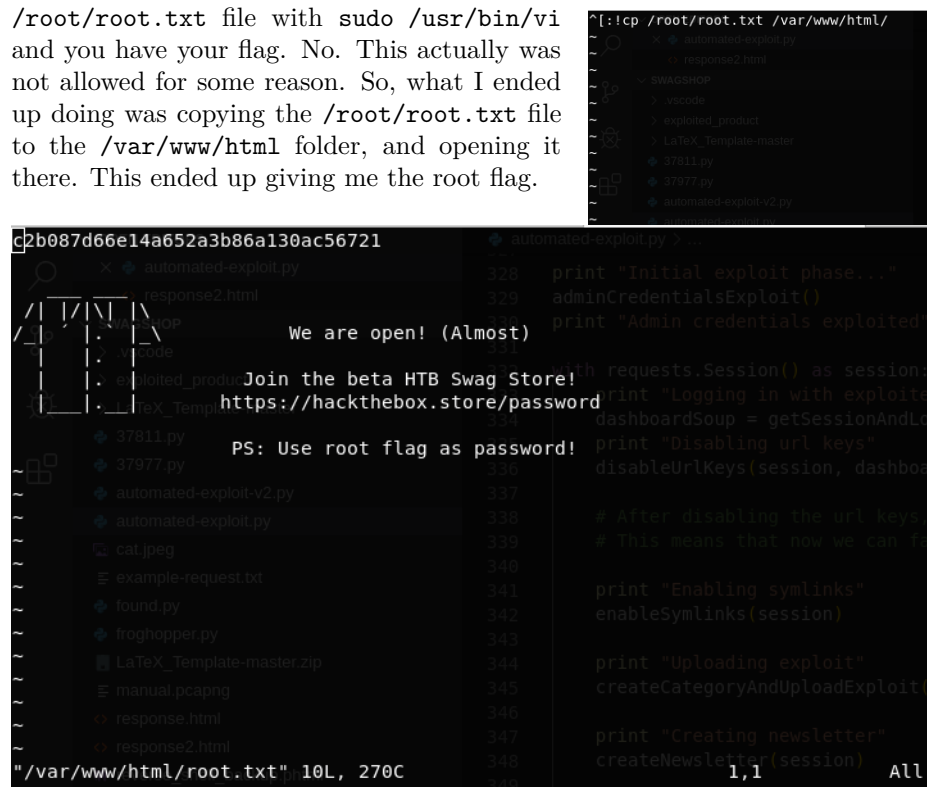


Figure 6.3: *The content of the root.txt file.*