2017 bctf boj writeup

原创

Anciety 于 2017-05-22 16:28:32 发布 825 收藏

分类专栏： ctf 文章标签： ctf

BCTF 2017 BOJ

This is a very interesting challenge for me.

We were given a working oj on http://oj.bctf.xctf.org.cn, there was only a single classical problem a + b
on it. I tried several times to play with it, only to found that nothing can
solve this a+b problem. :)

Solve

I lost a lot of time during the game on this challenge. Actually I could do better since I've been really close.

The first thought is to try to find out which header is blocked by this online judge system. So I tried use some
dangerous functions like “system”, “socket” etc. And all it just reported “Wrong Answer”. This means we can try
to use socket to communicate, and get what is actually going on on that server.

On my server, I opened a nc.Like this:

```
nc -lvv 32222 -k
```

So when I open a socket, and connect it back to my own server, I can get info.

Use this technique, I got the file directories on the server, only to find that under /usr/bin/ we have only 6
bins. That means we were under chroot environment. So, my thought here, was to get out.

But the next thing coming was that when I tried something like chroot function, the socket communication was down.
I tried several other functions, chroot and system not working, but many other functions like read or write works.
So, blocked. I looked up on Internet a little, found out there were some similar situations. They blocked this
functions using some systemcall filter technique. To get through this, we used X32 ABI. This is a series of system calls
that has been brought out on linux 3.4. It's kinda like a series of system calls doing the same thing as the original
ones but with different system call numbers. Since the filter uses the system call numbers to block system calls, with
this X32 ABI, we are able to pass. Thus, we used X32 ABI to chroot. Using the very basic chroot breakage technique, we
were able to get out.

The breakage in short, is like this:

```
    mkdir("fakeroot");
    x32_chroot("fakeroot"); // I don't know how libc support x32 abi, so I wrote my own x32_chroot
    chdir("../../../"); // now, we are out!
```

And just to be clear, x32_chroot is like this:

```
    int x32_chroot(const char *new_root) {
        return syscall(0x400000a1, new_root);
    }
```

After we got out, we were able to see things about this oj. After digging a little, we found the binary of the online judge. There is a “sandbox” binary and a “cr” binary.

And under root, there was a file named “flag”. But, unfortunately, we didn’t have the permisson to read it.

Use read and socket, we were able to download these two binary. With IDA, we found that in cr, it uses system to pass sandbox an argument. This argument is the name of out file.

Since we were able to use open system call, we can actually create a file with any name.

Now, if we **inject shell commands in file name**, the system function call in cr will run our command.

Since cr is basically running all the time, and, cr is runned by the user who owns the file, this command can read the flag.

So, by inject a command to read the file, and nc it back to our own server, we were able to get flag.

I haven’t solve this challenge until last 5 minutes, and there was no time left for me to make a right command. And the website now is down, I can’t say if my exp is right. Thus there is no exp here. :P

Some interesting thoughts

During the game, I havn’t solved this challenge. This was a big mistake since I already found the strange file name unser state/ directory. So, if you seek under state directory you will get a huge hint.

And, aside of that, I managed to use apache process to try to read the flag. This is kinda of misleading, the apache process had no permisson to read the flag but cost me a lot of time.

Well, I think I just should try to understand what normall people should think next time. :)

总结(What I’ve learned)

1. x32 abi是一套与普通系统调用号不一样，但是功能极其类似的一套完整的系统调用，可以用来解决过滤系统调用的问题
2. system里的字符串不应该是由用户可以控制的，否则将会导致由文件名带来的命令注入。