

《漏洞利用及渗透测试基础》实验报告

姓名：张洋 学号：2111460 班级：信安二班

实验名称：

格式化字符串漏洞实验

实验要求：

根据第四章示例 4-7 代码，完成任意地址的数据获取，观察 Release 模式和 Debug 模式的差异，并进行总结。

实验过程：

一、Debug 模式

The screenshot shows a debugger window with the following components:

- Assembly Code:** The main function `First.main(void)` is shown. The code starts with `push ebp`, `mov ebp, esp`, and `sub esp, 108`. It then pushes `ebx`, `esi`, and `edi`. A `lea edi, [ebp-108]` instruction is followed by `mov ecx, 42` and `mov eax, CCCCCCCC`. The `rep stos dword ptr [edi]` instruction is highlighted. Subsequent instructions include `push offset _iob`, `push 0C8`, `lea eax, [ebp-0C8]`, `push eax`, `call fgets`, `add esp, 0C`, `lea ecx, [ebp-0C8]`, `push ecx`, `call printf`, `add esp, 4`, `xor eax, eax`, `pop edi`, `pop esi`, `pop ebx`, and `add esp, 108`.
- Registers (FPU):** The register window shows the state of the CPU registers. `EAX` is `0000001F`, `ECX` is `00422A50`, `EDX` is `00422A50`, `EBX` is `7FFD4000`, `ESP` is `0012FE68`, `EBP` is `0012FF80`, `ESI` is `00000015`, and `EDI` is `0012FF80`. The `EIP` register points to `00401053`.
- Memory:** The memory window shows the stack contents. The address `00422000` to `0042200F` contains the value `CCCCCCCC`. The address `00422010` to `0042201F` contains the value `00000000`. The address `00422020` to `0042202F` contains the value `00000000`. The address `00422030` to `0042203F` contains the value `00000000`. The address `00422040` to `0042204F` contains the value `00000000`. The address `00422050` to `0042205F` contains the value `00000000`. The address `00422060` to `0042206F` contains the value `00000000`. The address `00422070` to `0042207F` contains the value `00000000`. The address `00422080` to `0042208F` contains the value `00000000`. The address `00422090` to `0042209F` contains the value `00000000`. The address `004220A0` to `004220AF` contains the value `00000000`.

逐过程分析：

1. 进入主函数，栈帧初始化，设置大小为 0x108

The screenshot shows the assembly code for the first step of the analysis:

- `00401010`: `> 55 push ebp`
- `00401011`: `. 8BEC mov ebp, esp`
- `00401013`: `. 81EC 00010000 sub esp, 108`

2. 存放原栈帧信息，利用 ecx 和 rep 指令把大小为 0x108 的栈空间初始化为 CCCCCCCC。

The screenshot shows the assembly code for the second step of the analysis:

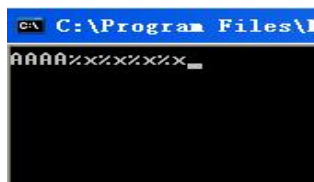
- `00401019`: `. 53 push ebx`
- `0040101A`: `. 56 push esi`
- `0040101B`: `. 57 push edi`
- `0040101C`: `. 8DBD F8FEFF lea edi, [ebp-108]`
- `00401022`: `. B9 42000000 mov ecx, 42`
- `00401027`: `. B8 CCCCCCCC mov eax, CCCCCCCC`
- `0040102C`: `. F3:AB rep stos dword ptr [edi]`

3. 初始化结束后，因为 str 字符数组的大小为 200，所以将栈帧存入 200 (0xC8) 位空间后的地址赋值给 eax，eax 为 0012FED8。

The screenshot shows the assembly code for the third step of the analysis:

- `0040102E`: `. 68 302A4200 push offset _iob`
- `00401033`: `. 68 C8000000 push 0C8`
- `00401038`: `. 8D85 38FFFF lea eax, [ebp-0C8]`
- `0040103E`: `. 50 push eax`

- 调用 `fgets` 函数，输入一个字符串。



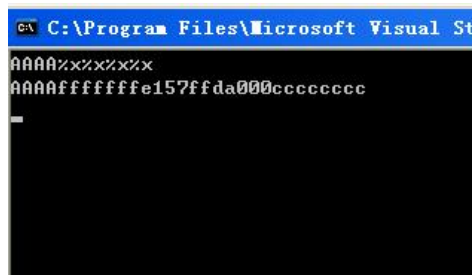
输入 “AAAA%x%x%x%x” 可以发现地址 0012FED8 上存放着输入的字符串。

0012FE60	0012FEB8	ASCII "AAAA%x%x%x%x"
0012FE64	000000B8	
0012FE68	00422A30	offset First._iob

- 调用完 `fgets` 函数后恢复 `esp` 的值，并将字符数组的起始地址作为 `printf` 函数的参数入栈。

0040103F	E8 CC000000	call fgets	fgets
00401044	83C4 0C	add esp,0C	
00401047	8D8D 38FFFFFF	lea ecx,[ebp-0C8]	
0040104D	51	push ecx	

- 调用 `printf` 函数，打印出的内容如下图所示。打印出了输入的字符串 “AAAA”，遇到 `%x` 时会自动依次打印出栈中接下来的地址中存放的内容（依次打印 `edi`, `esi`, `ebx` 的存储内容和 CCCCCCCC）



0012FE68	0012FEB8	ASCII "AAAA%x%x%x%x"
0012FE6C	FFFFFFFF	
0012FE70	00000015	
0012FE74	7FFDA000	
0012FE78	CCCCCCCC	

二、Release 模式

00401000	81EC C8000000	sub esp,0C8	First.00401000(guessed Arg1)	寄存器 (FPU)
00401005	8D4424 00	lea eax,[esp]		EAX 00410A60 ASCII "aa"
0040100A	68 30604000	push offset 00406030		ECX 004060A0 First.004060A0
0040100F	68 C8000000	push 0C8		EDX 00000003
00401014	50	push eax		EBX 7FFD5000
00401015	E8 47000000	call 00401061		ESP 0012FF84
0040101A	8D4C24 0C	lea ecx,[esp+0C]		EBP 0012FFC0
0040101E	51	push ecx		ESI 00000015
0040101F	E8 0C000000	call 00401030		EDI FFFFFFFF
00401024	33C0	xor eax,eax		EIP 00401000 First.00401000
00401026	81C4 D8000000	add esp,008		C 0 ES 0023 32Bit 0(FFFFFFFF)
0040102E	C3	ret		P 1 CS 001B 32Bit 0(FFFFFFFF)
0040102F	90	nop		A 0 SS 0023 32Bit 0(FFFFFFFF)
00401030	90	nop		Z 0 DS 0023 32Bit 0(FFFFFFFF)
00401031	53	push ebx		S 0 FS 003B 32Bit 7FFDF000(FFF)
00401032	56	push esi		T 0 GS 0000 NULL
00401037	57	push edi		O 0
00401038	56	push esi		O 0 LastErr 00000000 ERROR_SUCCESS
00401039	E8 5B020000	call 00401299		EFL 00000206 (NO,NB,NE,a,NS,PE,G)
0040103E	8BF8	mov edi,eax		ST0 empty +UNORM 009A 00129A94 0012A5C8
00401040	8D4424 18	lea eax,[esp+18]		ST1 empty +UNORM 0039 00129A7C 00129A94
00401044	50	push eax		ST2 empty +UNORM 0001 0012A020 000000A0
00401045	FF7424 18	push dword ptr [esp+18]		ST3 empty +UNORM 9A94 00AEC71D 00129FF8
00401046	5A	pop esi		ST4 empty +UNORM 0022 00000000 0057A3A4
00401047	5A	pop esi		ST5 empty +UNORM 0072 0067006F 00720050
00401048	5A	pop esi		ST6 empty +UNORM 0073 00650066 006900A6
00401049	5A	pop esi		
0040104A	5A	pop esi		
0040104B	5A	pop esi		
0040104C	5A	pop esi		
0040104D	5A	pop esi		
0040104E	5A	pop esi		
0040104F	5A	pop esi		
00401050	5A	pop esi		
00401051	5A	pop esi		
00401052	5A	pop esi		
00401053	5A	pop esi		
00401054	5A	pop esi		
00401055	5A	pop esi		
00401056	5A	pop esi		
00401057	5A	pop esi		
00401058	5A	pop esi		
00401059	5A	pop esi		
0040105A	5A	pop esi		
0040105B	5A	pop esi		
0040105C	5A	pop esi		
0040105D	5A	pop esi		
0040105E	5A	pop esi		
0040105F	5A	pop esi		
00401060	5A	pop esi		
00401061	5A	pop esi		
00401062	5A	pop esi		
00401063	5A	pop esi		
00401064	5A	pop esi		
00401065	5A	pop esi		
00401066	5A	pop esi		
00401067	5A	pop esi		
00401068	5A	pop esi		
00401069	5A	pop esi		
0040106A	5A	pop esi		
0040106B	5A	pop esi		
0040106C	5A	pop esi		
0040106D	5A	pop esi		
0040106E	5A	pop esi		
0040106F	5A	pop esi		
00401070	5A	pop esi		
00401071	5A	pop esi		
00401072	5A	pop esi		
00401073	5A	pop esi		
00401074	5A	pop esi		
00401075	5A	pop esi		
00401076	5A	pop esi		
00401077	5A	pop esi		
00401078	5A	pop esi		
00401079	5A	pop esi		
0040107A	5A	pop esi		
0040107B	5A	pop esi		
0040107C	5A	pop esi		
0040107D	5A	pop esi		
0040107E	5A	pop esi		
0040107F	5A	pop esi		
00401080	5A	pop esi		
00401081	5A	pop esi		
00401082	5A	pop esi		
00401083	5A	pop esi		
00401084	5A	pop esi		
00401085	5A	pop esi		
00401086	5A	pop esi		
00401087	5A	pop esi		
00401088	5A	pop esi		
00401089	5A	pop esi		
0040108A	5A	pop esi		
0040108B	5A	pop esi		
0040108C	5A	pop esi		
0040108D	5A	pop esi		
0040108E	5A	pop esi		
0040108F	5A	pop esi		
00401090	5A	pop esi		
00401091	5A	pop esi		
00401092	5A	pop esi		
00401093	5A	pop esi		
00401094	5A	pop esi		
00401095	5A	pop esi		
00401096	5A	pop esi		
00401097	5A	pop esi		
00401098	5A	pop esi		
00401099	5A	pop esi		
0040109A	5A	pop esi		
0040109B	5A	pop esi		
0040109C	5A	pop esi		
0040109D	5A	pop esi		
0040109E	5A	pop esi		
0040109F	5A	pop esi		
004010A0	5A	pop esi		
004010A1	5A	pop esi		
004010A2	5A	pop esi		
004010A3	5A	pop esi		
004010A4	5A	pop esi		
004010A5	5A	pop esi		
004010A6	5A	pop esi		
004010A7	5A	pop esi		
004010A8	5A	pop esi		
004010A9	5A	pop esi		
004010AA	5A	pop esi		
004010AB	5A	pop esi		
004010AC	5A	pop esi		
004010AD	5A	pop esi		
004010AE	5A	pop esi		
004010AF	5A	pop esi		
004010B0	5A	pop esi		
004010B1	5A	pop esi		
004010B2	5A	pop esi		
004010B3	5A	pop esi		
004010B4	5A	pop esi		
004010B5	5A	pop esi		
004010B6	5A	pop esi		
004010B7	5A	pop esi		
004010B8	5A	pop esi		
004010B9	5A	pop esi		
004010BA	5A	pop esi		
004010BB	5A	pop esi		
004010BC	5A	pop esi		
004010BD	5A	pop esi		
004010BE	5A	pop esi		
004010BF	5A	pop esi		
004010C0	5A	pop esi		
004010C1	5A	pop esi		
004010C2	5A	pop esi		
004010C3	5A	pop esi		
004010C4	5A	pop esi		
004010C5	5A	pop esi		
004010C6	5A	pop esi		
004010C7	5A	pop esi		
004010C8	5A	pop esi		
004010C9	5A	pop esi		
004010CA	5A	pop esi		
004010CB	5A	pop esi		
004010CC	5A	pop esi		
004010CD	5A	pop esi		
004010CE	5A	pop esi		
004010CF	5A	pop esi		
004010D0	5A	pop esi		
004010D1	5A	pop esi		
004010D2	5A	pop esi		
004010D3	5A	pop esi		
004010D4	5A	pop esi		
004010D5	5A	pop esi		
004010D6	5A	pop esi		
004010D7	5A	pop esi		
004010D8	5A	pop esi		
004010D9	5A	pop esi		
004010DA	5A	pop esi		
004010DB	5A	pop esi		
004010DC	5A	pop esi		
004010DD	5A	pop esi		
004010DE	5A	pop esi		
004010DF	5A	pop esi		
004010E0	5A	pop esi		
004010E1	5A	pop esi		
004010E2	5A	pop esi		
004010E3	5A	pop esi		
004010E4	5A	pop esi		
004010E5	5A	pop esi		
004010E6	5A	pop esi		
004010E7	5A	pop esi		
004010E8	5A	pop esi		
004010E9	5A	pop esi		
004010EA	5A	pop esi		
004010EB	5A	pop esi		
004010EC	5A	pop esi		
004010ED	5A	pop esi		
004010EE	5A	pop esi		
004010EF	5A	pop esi		
004010F0	5A	pop esi		
004010F1	5A	pop esi		
004010F2	5A	pop esi		
004010F3	5A	pop esi		
004010F4	5A	pop esi		
004010F5	5A	pop esi		
004010F6	5A	pop esi		
004010F7	5A	pop esi		
004010F8	5A	pop esi		
004010F9	5A	pop esi		
004010FA	5A	pop esi		
004010FB	5A	pop esi		
004010FC	5A	pop esi		
004010FD	5A	pop esi		
004010FE	5A	pop esi		
004010FF	5A	pop esi		

1. 进入主函数，release 模式下没有栈帧的切换，不将 ebp 入栈，只为字符数组声明 200 字节的空间，从而将 esp 抬高 200 字节，再将此时 esp 的值赋值给 eax，eax=0012FEB0。

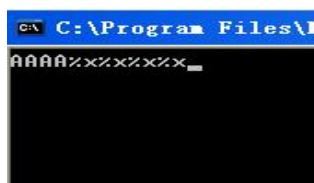
00401000	81EC C8000000	sub esp,0C8	First.00401000(guessed Arg1)
00401006	8D4424 00	lea eax,[esp]	

2. 把地址，立即数，eax 寄存器入栈。

0040100A	68 30604000	push offset 00406030	
0040100F	68 C8000000	push 0C8	
00401014	50	push eax	

3. 调用 fgets 函数，输入 “AAAA%x%x%x%x”。

00401015	E8 47000000	call 00401061	
----------	-------------	---------------	--



此时栈顶（0012FEB0）的存储内容为已经输入的字符串的地址（0012FEB0）。

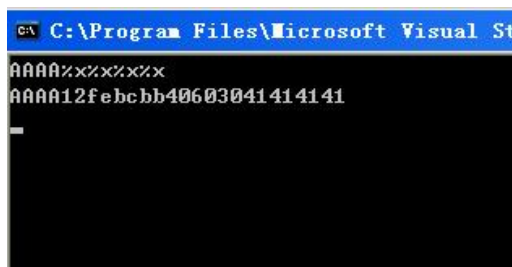
0012FEB0	0012FEB0	ASCII "AAAA%x%x%x%x"
0012FEB4	000000BB	
0012FEB8	00406030	ASCII "\n"
0012FEBC	41414141	

4. 将字符串的地址赋值给 ecx，此时 ecx=0012FEB0，将 ecx 入栈，此时栈中存放了两个字符串的地址。

0040101A	8D4C24 0C	lea ecx,[esp+0C]	
0040101E	51	push ecx	

0012FEAC	0012FEB0	ASCII "AAAA%x%x%x%x"
0012FEB0	0012FEB0	ASCII "AAAA%x%x%x%x"
0012FEB4	000000BB	
0012FEB8	00406030	ASCII "\n"
0012FEBC	41414141	

5. 调用 printf 函数，可以看到打印出了输入的字符串 “AAAA”，遇到 %x 时会自动依次打印出栈中接下来的地址中存放的内容（字符串地址和压入的变量）。

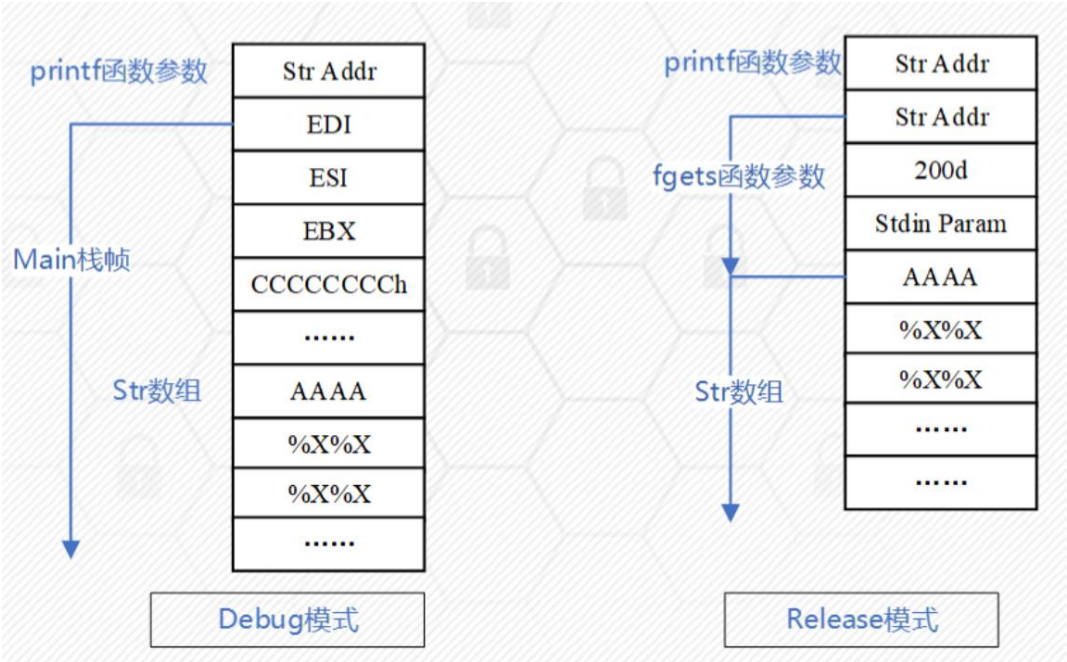


6. eax 寄存器中的内容清空，esp 恢复成原来的地址，最后 return。

00401024	33C0	xor eax,eax	
00401026	81C4 D8000000	add esp,0D8	
0040102C	C3	ret	

心得体会：

执行 printf(str) 语句的时候，对比 Debug 模式和 Release 模式的栈帧结构：



结合此题而言，我记录了 Debug 模式和 Release 模式下栈的地址和内容，以便更加直观地查看：

1. Debug 模式

0012FE68	0012FEB8（字符串地址）
0012FE6C	FFFFFFFF（最开始 edi 的值）
0012FE70	00000015（最开始 esi 的值）
0012FE74	7FFDA000（最开始 ebx 的值）
0012FE78	CCCCCCCC
.....	CCCCCCCC
0012FEB4	CCCCCCCC
0012FEB8	41414141（“AAAA”）
0012FEB8	78257825（‘%’ ->25，‘x’ ->78）
0012FEC0	78257825（‘%’ ->25，‘x’ ->78）
0012FEC4	CCCC000A（0A 代表输入结束）
0012FEC8	CCCCCCCC
.....	CCCCCCCC
0012FE7C	CCCCCCCC

2. Release 模式

0012FEAC	0012FEB8（字符串地址）
0012FEB0	0012FEB8（字符串地址）
0012FEB4	000000BB
0012FEB8	00406030
0012FEB8	41414141（“AAAA”）
0012FEC0	78257825（‘%’ ->25，‘x’ ->78）
0012FEC4	78257825（‘%’ ->25，‘x’ ->78）

通过本次实验，我深入地了解 Debug 模式和 Release 模式下栈是如何变化的，以及他们之间存在怎样的区别。Debug 模式下栈帧 ebp 会随着函数不断调整，并且会初始化一部分空间为 CCCCCC；在 Release 模式下 ebp 不会变化，并且不用初始化空间，直接抬高 esp 来分配需要的字符数组的空间。