

网络攻击与防御

刘智
西南石油大学 计算机科学学院
zhi.liu@swpu.edu.cn

第四章 漏洞与缓冲区溢出

第一节 漏洞概述

本节内容与目标

- 了解漏洞定义、漏洞分类、漏洞发展趋势
- 了解主要漏洞数据库
- 理解漏洞管理方法

近年重大漏洞



永恒之蓝漏洞

软件: Windows内核(SMB协议)

原因: 缓冲区溢出

编号: CVE-2017-0144, MS17-010)

心血漏洞

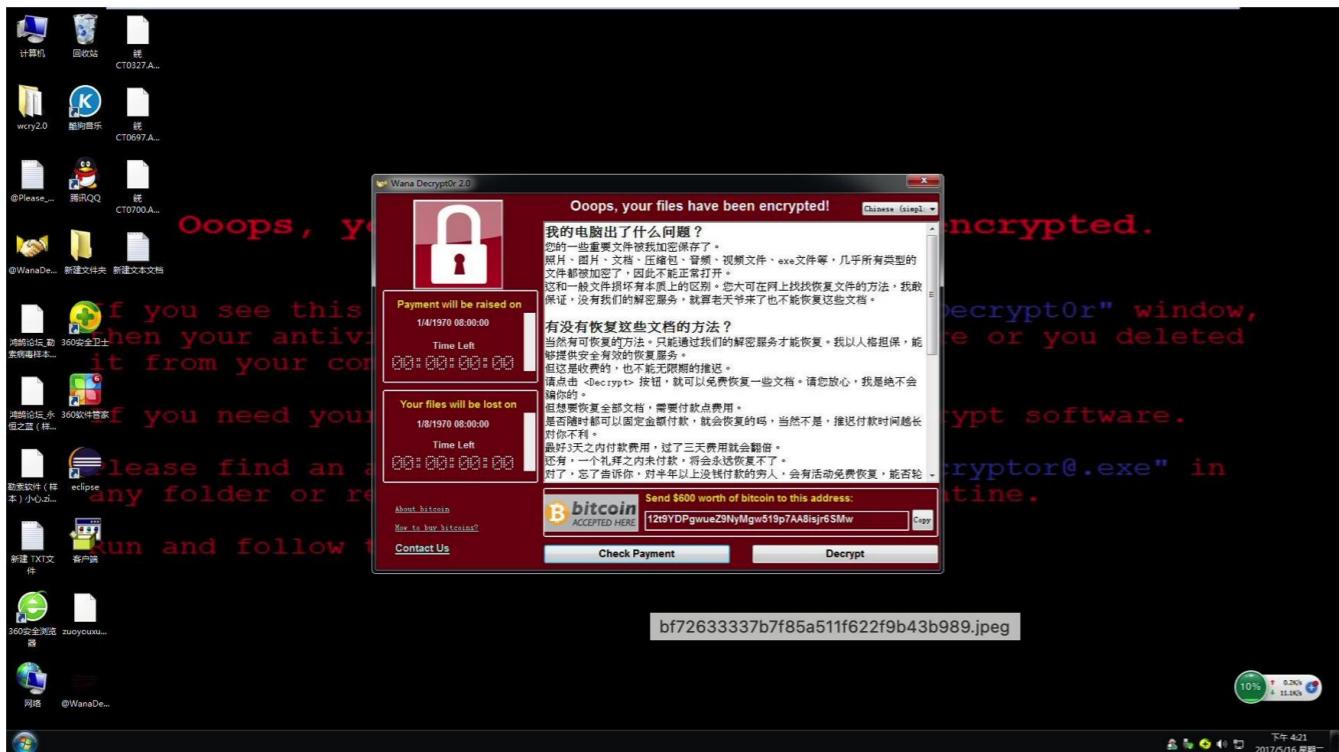
软件: OpenSSL 1.0.1

原因: 缓冲区溢出

编号: CVE-2014-0160

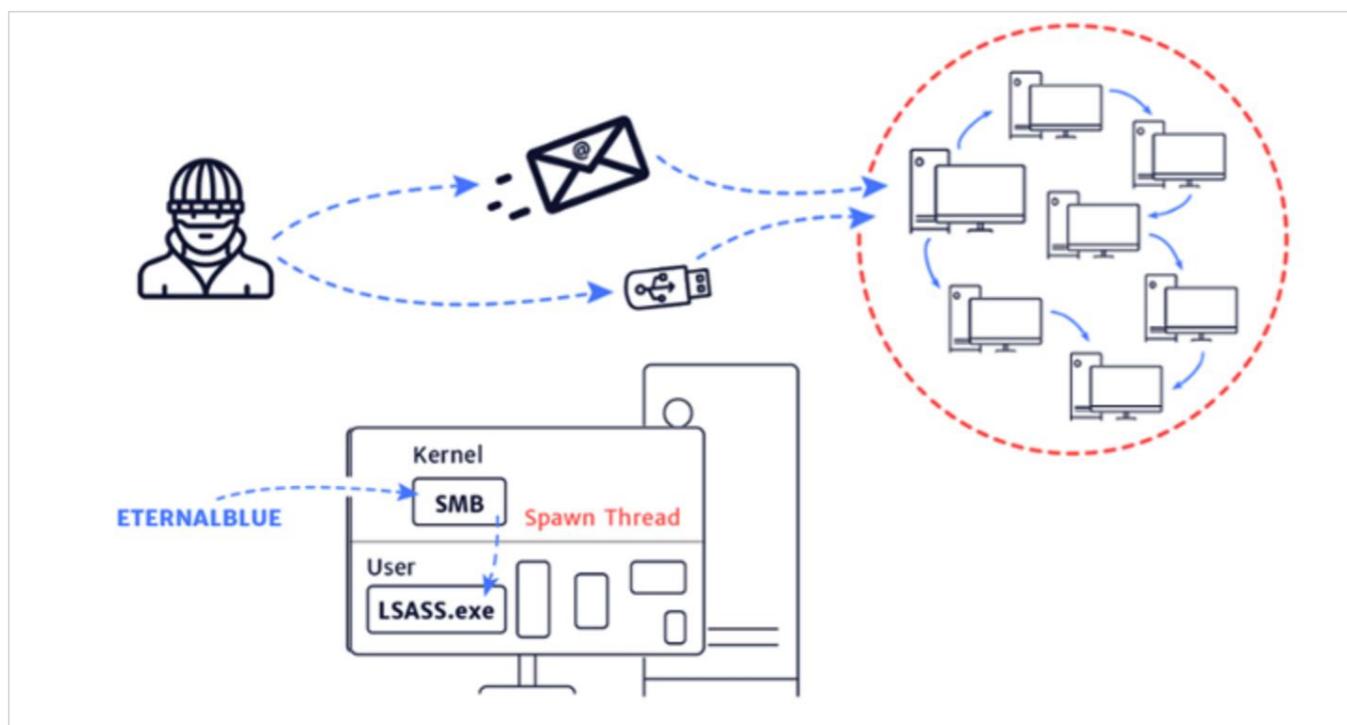
永恒之蓝漏洞

永恒之蓝漏洞(CVE-2017-0144, MS17-010)



被攻击软件: Windows SMB协议

受影响软件: Windows操作系统



漏洞定义

- **漏洞或脆弱性** (Vulnerability) 是指计算机系统安全方面的缺陷，使得系统或其应用数据的保密性、完整性、可用性、访问控制等面临威胁。
- 简单讲，漏洞就是可以被攻击利用的系统弱点。

漏洞影响范围

- 应用程序：数据库、Web程序、浏览器、...
- 操作系统
- 网络及安全设备
 - 路由器、防火墙等
- 新型计算与网络环境
 - 网络：SDN、5G、车联网等
 - 硬件：IoT固件等

漏洞带来的损失

- **永恒之蓝** 80亿美元¹
- **心血漏洞** 5亿美元²
- **熔断漏洞** 百亿美元规模³

¹ <https://baike.baidu.com/item/WannaCry/20797421?fr=aladdin&fromtitle=%E6%B0%B8%E6%81%92%E4%B9%8B%E8%93%9D&fromid=4951714>

² <https://zh.wikipedia.org/wiki/%E5%BF%83%E8%84%8F%E5%87%BA%E8%A1%80%E6%BC%8F%E6%B4%9E>

³ <https://www.secrss.com/articles/195>

0-day漏洞

- 定义

“零日漏洞（0-day vulnerability）通常是指还没有补丁的安全漏洞，而零日攻击（zero-day exploit、zero-day attack）则是指利用这种漏洞进行的攻击。”

- 著名0-day漏洞

- 永恒之蓝(MS17-010)
- 心血漏洞(CVE-2014-0160)

漏洞难以消除

- 不安全的代码和软硬件设计
 - 编程语言(特别是C/C++)
 - 软件设计因素
 - 硬件漏洞

漏洞发展趋势

- 漏洞数量日益增多，且危害日益严重
 - 2018年漏洞14201个，高危4898个，0-day 5381个
- 漏洞影响范围显著扩大
 - 应用程序57.8%，web18.7%，操作系统10.6%
 - 网络设备9.5%，安全产品2.4%，数据库1.0%



图 8 2013 年至 2018 年 CNVD 收录安全漏洞数量对比

漏洞涉及厂商	漏洞数量 (单位：个)	占全年收录数量百分比
Google	693	4.9%
Microsoft	667	4.7%
IBM	564	4.0%
Oracle	481	3.4%
Cisco	422	3.0%
Foxit	369	2.6%
Apple	367	2.6%
Adobe	352	2.5%
WordPress	261	1.8%
Linux	193	1.4%
其他	9,832	69.2%

漏洞攻击趋势

- 攻击利用趋于**简单化、平台化**
- 漏洞利用时间周期缩短

攻击名称	漏洞发布时间	大规模爆发时间	时间间隔
尼姆达	2000-10-17	2001-09-18	336天
Worm_Slammer	2002-07-24	2003-01-25	185天
冲击波	2003-07-16	2003-08-11	26天
震荡波	2004-04-13	2004-05-01	18天
MS08-067	2008-10-23	2008-10-23前	<0天

CNNVD

- **中国国家信息安全漏洞库**(China National Vulnerability Database of Information Security)
- 国内权威的漏洞库
- 负责漏洞的发布和披露

CNNVD介绍

CNNVD
国家信息安全漏洞库
China National Vulnerability Database of Information Security

首页 漏洞信息 补丁信息 网安时情 数据立方 漏洞报告 漏洞预警 兼容性

热点漏洞

- 1 Microsoft Internet Explorer 安全漏洞
漏洞编号 CNNVD-202001-876 厂商 microsoft 发布日期 2020-1-17
补丁
- 2 Juniper Networks Junos OS 安全漏洞
漏洞编号 CNNVD-202001-302 厂商 juniper 发布日期 2020-1-9
补丁
- 3 Schneider Electric EcoStruxure Geo ...
漏洞编号 CNNVD-202001-138 厂商 schneider-electric 发布日期 2020-1-6
补丁
- 4 Cisco Data Center Network Manager 信...
漏洞编号 CNNVD-202001-040 厂商 Cisco 发布日期 2020-1-2
补丁
- 5 Tencent WeChat 输入验证错误漏洞
漏洞编号 CNNVD-201912-1546 厂商 Tencent 发布日期 2019-12-31
补丁

CNNVD 动态

- 更多 »
2018年度 CNNVD技术支撑单位...
发布日期 2019-08-13
- 关于国家信息安全漏洞库（CNNV...
发布日期 2019-05-13
- 关于2018年“CNNVD兼容性服...
发布日期 2018-10-23
- 关于“CNNVD 2017年度最佳技...
发布日期 2018-09-25
- 关于CNNVD 2018年新增“技术...
发布日期 2018-09-25

漏洞信息详情

Microsoft Windows SMB 输入验证漏洞

CNNVD编号: CNNVD-201703-725

危害等级: 超危

CVE编号: CVE-2017-0144

漏洞类型: 输入验证错误

发布时间: 2017-03-28

威胁类型: 远程

更新时间: 2019-10-08

厂商: Microsoft

漏洞来源: Metasploit,Luke Je...

漏洞简介

Microsoft Windows是美国微软（Microsoft）公司发布的一系列操作系统。SMBv1 server是其中的一个服务器协议组件。

Microsoft Windows中的SMBv1服务器存在远程代码执行漏洞。远程攻击者可借助特制的数据包利用该漏洞执行任意代码。以下版本受到影响: Microsoft Windows Vista SP2, Windows Server 2008 SP2和R2 SP1, Windows 7 SP1, Windows 8.1, Windows Server 2012 Gold和R2, Windows RT 8.1, Windows 10 Gold, 1511和1607, Windows Server 2016。

漏洞公告

目前厂商已经发布了升级补丁以修复此安全问题, 补丁获取链接:

<https://technet.microsoft.com/zh-cn/library/security/ms17-010>

参考网址

来源:packetstormsecurity.com

链接:<https://packetstormsecurity.com/files/154690/DOUBLEPULSAR-Payload-Execution-Neutralization.html>

来源:www.exploit-db.com

链接:<https://www.exploit-db.com/exploits/47456>

CVE

- **通用漏洞披露**（英语：CVE, Common Vulnerabilities and Exposures）是一个与信息安全有关的数据库，收集各种信息安全弱点及漏洞并给予编号以便于公众查阅。

命令规则 CVE-年份-数字编号

永恒之蓝 CVE-2017-0144

漏洞扫描

● 网络扫描

- 网络扫描方式发现目标主机漏洞
- 使用场景：不能登录主机

● 主机扫描

- 使用**凭证**登陆到主机分析漏洞：密码/SSH/SMB等方式。
- 使用场景：允许登录主机
- 结果更准确

漏洞扫描原理是什么？

常见漏洞扫描工具

- Nmap
- Nessus
- VAS

请合法合规使用！

漏洞评估与管理

● 漏洞评估

A vulnerability assessment is the process of **identifying, quantifying, and prioritizing** (or ranking) the vulnerabilities in a system.

(漏洞评估是识别，量化和确定系统中漏洞的优先级的过程。)

● 漏洞管理

Vulnerability management is the cyclical practice of identifying, classifying, prioritizing, **remediating, and mitigating** software vulnerabilities.

(漏洞管理是识别，分类，划分优先级，补救和缓解软件漏洞的周期性实践。)

漏洞管理流程



本节小结

- 漏洞定义、漏洞分类、漏洞发展趋势
- 主要漏洞数据库
- 漏洞管理流程

第四章 漏洞与缓冲区溢出

第二节 缓冲区溢出原理

本节内容与目标

- 掌握栈溢出原理
- 熟悉格式化字符串溢出、整数溢出和函数指针覆盖
- 了解shellcode和exploit

典型漏洞类型

- 栈溢出
- 堆溢出
- 格式化字符串漏洞
- 整数溢出
- 函数指针覆盖

缓冲区溢出

- 覆盖相邻存储单元，造成程序异常或崩溃
- 巧妙设计的溢出可获取用户乃至root权限
- 缓冲区溢出持续存在
 - 大量核心软件使用C/C++编写
 - 操作系统、协议栈、数据库、浏览器....
 - 未及时安装补丁和遗留系统(legacy)

```
gcc release.c -o release && ./release
executing code...
loading wave data
artist : telectar
bpm : 160.00
peak : -0.00 db
size : 29.63 mb
audio data length: 82:56 min
key : F minor => initialising audio playback...
segmentation fault.
(Core dumped)
```

缓冲区溢出影响

1988	Morris Worm	Unix漏洞， 1000万-1亿美元损失
2001	红色代码蠕虫	微软IIS服务器漏洞， 35万台机器感染， 损失26-30亿美元
2003	冲击波蠕虫	微软DCOM漏洞， 40万台机器感染， 损失超5亿美元
2018	★永恒之蓝	微软SMB漏洞， 经济损失超80亿美元

缓冲区溢出定义

- 缓冲区溢出 (buffer overflow) , 是针对程序设计缺陷, 向程序输入缓冲区写入使之溢出的内容, **从而破坏程序运行、趁著中断之际并取得程序乃至系统的控制权。**

```
char v[128];
foo(v);

void foo(char *p) {
    char var[64];
    memcpy(var,p,128);
}
```

预备知识

- 程序空间结构
- 栈和堆存放什么数据?
- 栈和堆最大值是多少?
- **函数调用工作原理**

缓冲区

- 缓冲区

- 缓冲区(buffer)就是应用程序用来保存数据的内存空间。

- 主要存储位置

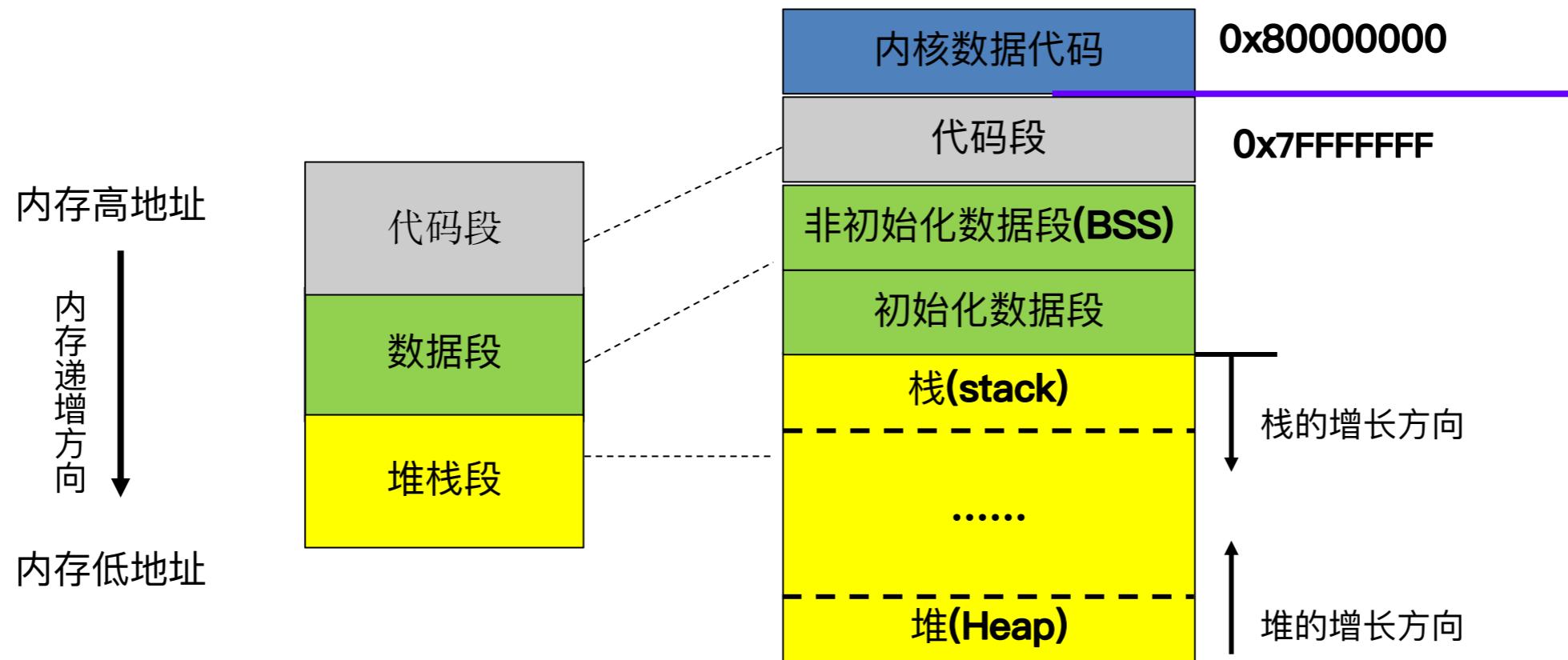
- 栈(stack)
- 堆(heap)
- 静态区

```
char buf[128];  
int (*func_ptr)(int);  
  
void foo(char *p) {  
    char var[64];  
    char *q=(char *)malloc(128);  
}
```

栈(stack)

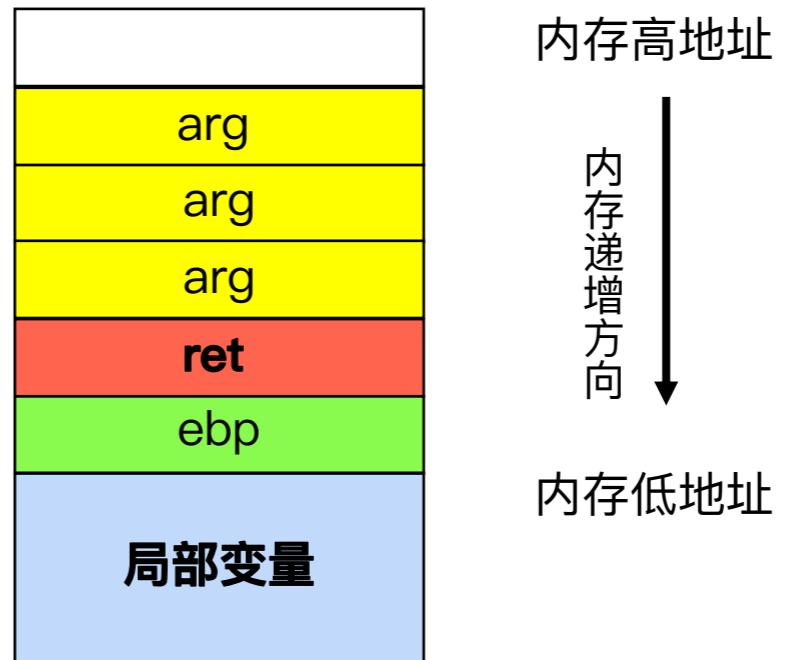
- 栈是一块连续内存空间
 - 先入后出
 - 从高地址向低地址增长
- 每一个线程有自己的栈
- 提供一个“**暂时存放数据**”区域
- 使用**POP/PUSH**指令来对栈进行操作
 - 使用ESP寄存器指向栈顶，EBP指向栈帧底

程序地址空间



栈布局

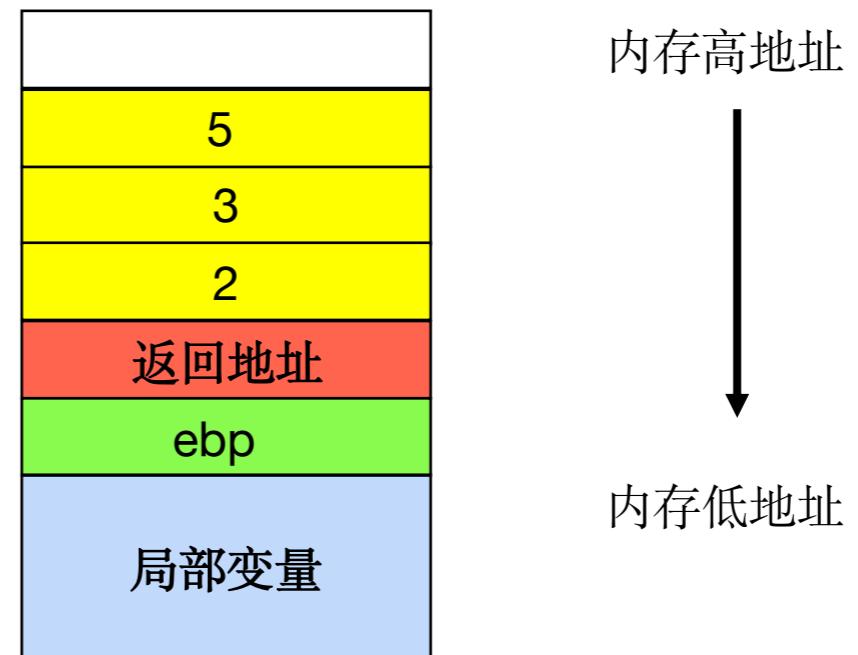
- 函数参数(arg)
- 函数返回地址(ret)
 - 即函数返回后执行指令的地址
- EBP
- 函数局部变量



函数调用

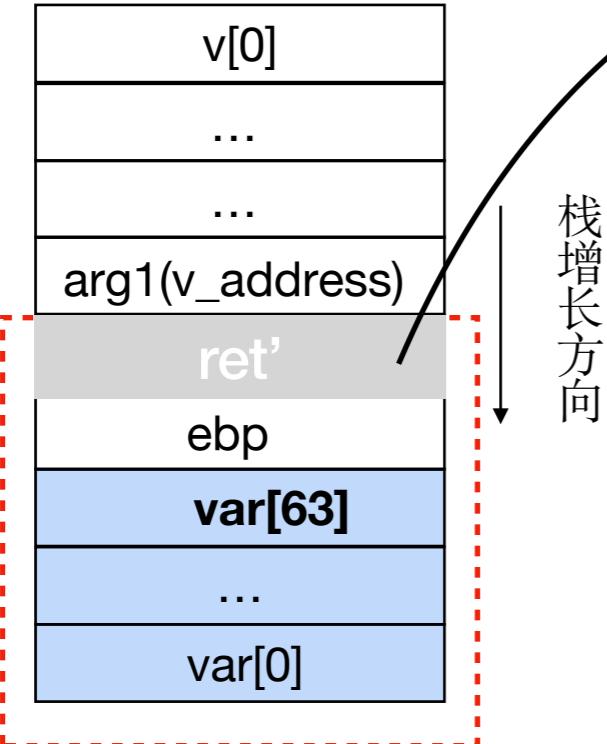
- 参数都保存在栈
- 通常从右到左压栈(cdcel调用)
- 参数与局部变量访问
 - 参数：ebp+offset
 - 变量：ebp-offset

```
void foo(int a, int b, int c){  
    ...  
}  
  
foo(2,3,5);
```



栈溢出分析

内存增长方向 ↑



```
char v[128];  
foo(v);  
... //正常返回  
  
void foo(char *p) {  
    char var[64];  
memcpy(var, p, 128);  
}
```

```
mov %v, %eax  
push %eax  
push %ret_foo  
push %ebp  
mov %esp, %ebp  
sub %esp, 64  
push var1,%esx  
call memcpy  
  
add %esp, 64  
pop %ebp  
mov %ebp %esp  
ret
```

常见溢出函数

- 字符串函数
 - **strcpy()**, **strcat()**
- 内存操作函数
 - **memcpy()**
- print函数
 - printf(), sprintf(), vsprintf()
- get/put函数
 - getc(), puts(), gets(), puts()

整数溢出

- 当算术运算试图创建一个**超出给定数**

字位数（大于最大或小于最小可表示值）的范围的数值时，就会发生整数溢出。溢出情况可能会导致意外行为。特别是如果未预料到这种可能性，则溢出可能会损害程序的可靠性和安全性”。

```
#define LEN 512

void vuln(char *src, int s){
    char dst[LEN];
    int size = s; → s = 0xFFFFFFFF
    if (s < len){
        memcpy(dst, src, size)
    }
}
```

unsigned int

整数溢出案例

- **Apache Web Server分块(chunked)编码溢出漏洞
(2002年)**

- 出于安全考虑，在将分块数据拷贝到缓冲区之前，Apache会对分块长度进行检查。
- 如果分块长度小于缓冲区，根据分块长度进行数据拷贝；
如果分块长度大于缓冲区，Apache将最多只拷贝缓冲区长度的数据。
- 然而在进行上述检查时，**没有将分块长度转换为无符号型进行比较**。
因此，如果攻击者将分块长度**设置成一个负值**，就会 绕过上述安全检查，Apache会将一个超长的分块数据拷贝到缓冲区中，这会造成缓冲区溢出。

- **影响软件**

- 1.3到1.3.24(含1.3.24)版本的Apache

漏洞利用流程

- 注入攻击数据
- 溢出缓冲区
- **修改控制流**
- 执行攻击代码

shellcode定义

- shellcode是一段用于利用软件漏洞而执行的代码，shellcode为十六进制数据，以其经常让攻击者获得shell而得名”。

```
shellcode = (
    '\x31\xc0\x89\xc3\xb0\x17\xcd\x80\x31\xd2\x1e +illed.
    '\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89' +
    '\xe3\x52\x53\x89\xe1\x8d\x42\x0b\xcd\x80'
)oot@kali:~# cd kali2
paddingl='A'*i(112p-b64b4 32)
eip=k\x70\xf3\xff\xbf'b4
```

Shellcode介绍

- Shellcode如何构造?
- Shellcode可以实现哪些功能?

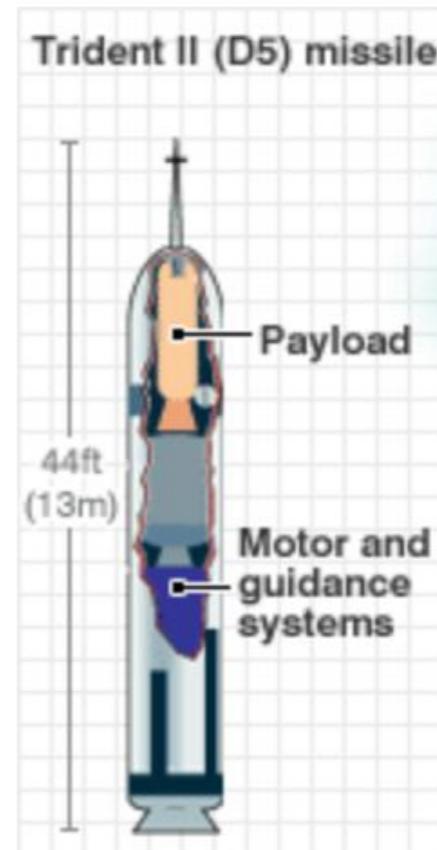
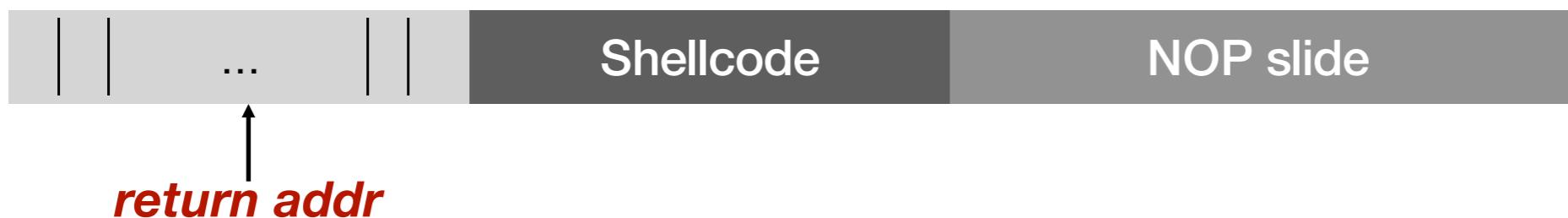
```
[msf5] exploit(windows/smb/ms17_010_eternalblue) > set payload windows/x64/  
set payload windows/x64/exec  
set payload windows/x64/loadlibrary  
set payload windows/x64/messagebox  
set payload windows/x64/meterpreter/bind_ipv6_tcp  
set payload windows/x64/meterpreter/bind_ipv6_tcp_uuid  
set payload windows/x64/meterpreter/bind_named_pipe  
set payload windows/x64/meterpreter/bind_tcp  
set payload windows/x64/meterpreter/bind_tcp_rc4  
set payload windows/x64/meterpreter/bind_tcp_uuid  
set payload windows/x64/meterpreter/reverse_http  
set payload windows/x64/meterpreter/reverse_https  
set payload windows/x64/meterpreter/reverse_named_pipe  
set payload windows/x64/meterpreter/reverse_tcp  
set payload windows/x64/meterpreter/reverse_tcp_rc4  
set payload windows/x64/meterpreter/reverse_tcp_uuid  
set payload windows/x64/meterpreter/reverse_winhttp  
set payload windows/x64/meterpreter/reverse_winhttps  
set payload windows/x64/pingback_reverse_tcp  
set payload windows/x64/powershell_bind_tcp  
set payload windows/x64/powershell_reverse_tcp  
set payload windows/x64/shell/bind_ipv6_tcp  
set payload windows/x64/shell/bind_ipv6_tcp_uuid  
set payload windows/x64/shell/bind_named_pipe  
set payload windows/x64/shell/bind_tcp  
set payload windows/x64/shell/bind_tcp_rc4  
set payload windows/x64/shell/bind_tcp_uuid  
set payload windows/x64/shell/reverse_tcp  
set payload windows/x64/shell/reverse_tcp_rc4  
set payload windows/x64/shell/reverse_tcp_uuid  
set payload windows/x64/shell_bind_tcp  
set payload windows/x64/shell_reverse_tcp  
set payload windows/x64/vncinject/bind_ipv6_tcp  
set payload windows/x64/vncinject/bind_ipv6_tcp_uuid  
set payload windows/x64/vncinject/bind_named_pipe  
set payload windows/x64/vncinject/bind_tcp  
set payload windows/x64/vncinject/bind_tcp_rc4  
set payload windows/x64/vncinject/bind_tcp_uuid  
set payload windows/x64/vncinject/reverse_http  
set payload windows/x64/vncinject/reverse_https  
set payload windows/x64/vncinject/reverse_tcp  
set payload windows/x64/vncinject/reverse_tcp_rc4  
set payload windows/x64/vncinject/reverse_tcp_uuid  
set payload windows/x64/vncinject/reverse_winhttp  
set payload windows/x64/vncinject/reverse_winhttps
```

Metasploit提供的部分payload

Exploit

- 一个利用程序(exploit)就是一段通过触发一个漏洞进而控制目标系统的代码。攻击代码通常会释放攻击载荷 (shellcode) , 里面包含了攻击者想要执行的代码。

Exploit经典布局



本节小结

- 栈溢出原理
- shellcode和exploit

第四章 漏洞与缓冲区溢出

第三节 缓冲区溢出防御

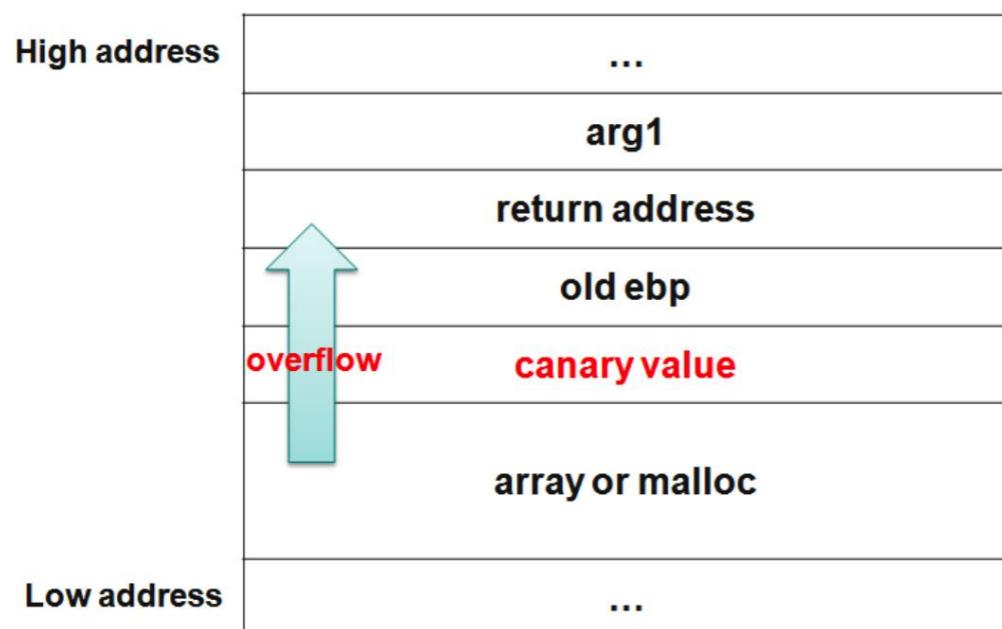
主要内容

- 缓冲区溢出经典防御技术
 - Stack Canary
 - DEP(数据不可执行)
 - ASLR(地址随机化)
- Return-to-libc

Stack Canary

- “金丝雀”(canary)是一种**编译器级栈溢出防御技术**
- 栈中EBP寄存器下面插入一个随机值
- 若返回地址被覆盖，则canary也会被修改

Memory map



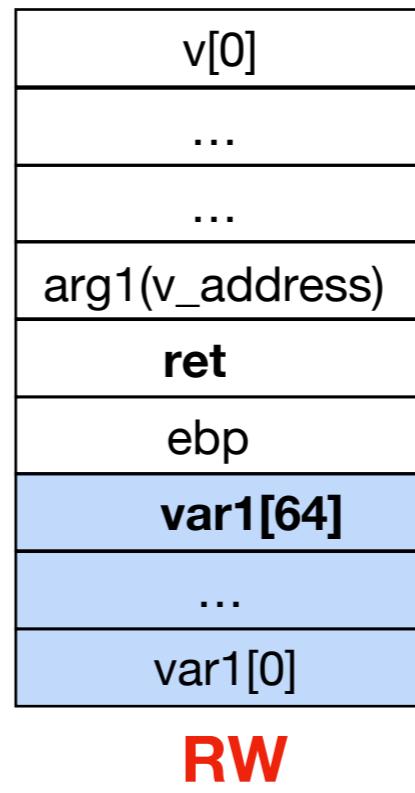
DEP(数据不可执行)

- 技术原理

- 操作系统将数据页面设为不可执行
- 即使栈发生溢出，也无法执行代码

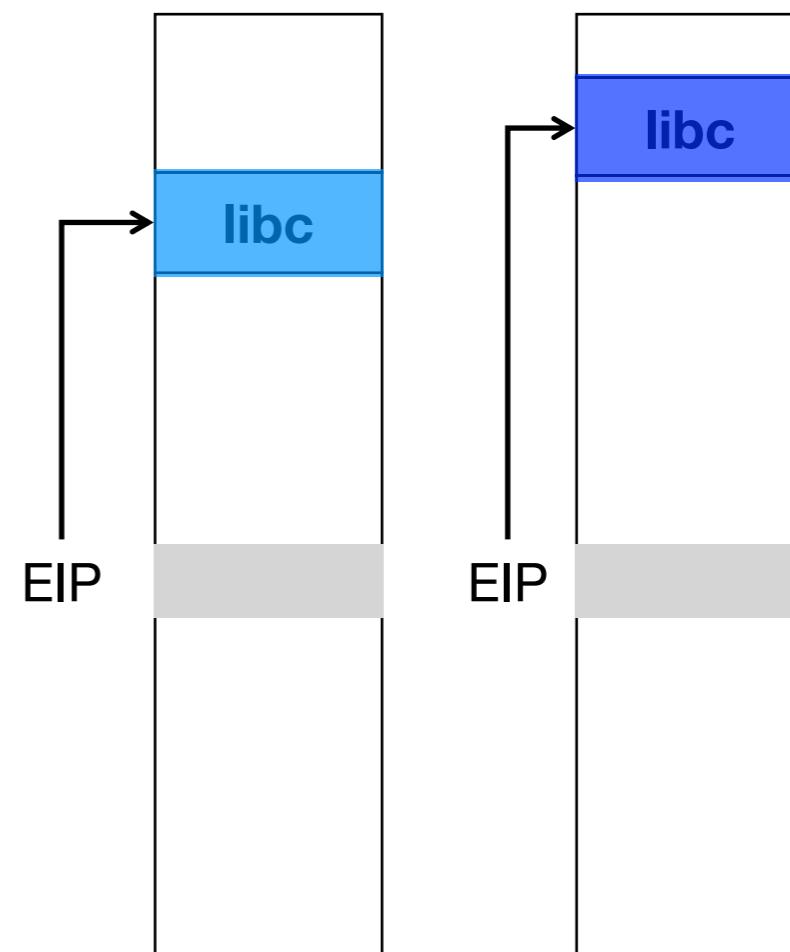
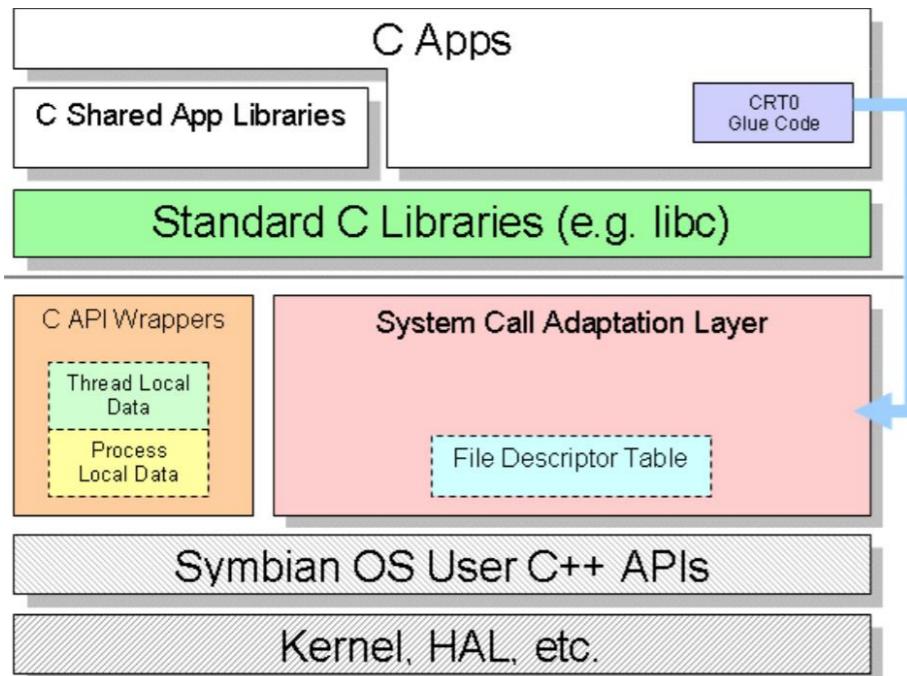
- 技术特性

- 阻止shellcode执行
- 需要操作系统支持



ASLR(地址空间随机化)

- 操作系统启动时将libc库加载到随机地址
- 攻击者难以猜测libc库地址



地址空间随机化分析

- 本质上是一种概率性保护方案，显著增加攻击成本
- 优点
 - 防御成本低、效果好
 - 技术实现难度不高
- 缺点
 - 理论上无法实现100%防御
 - 需要操作系统支持

操作系统的ALSR支持

Linux

v2.6.12, 2005

Android

4.0 IceCream

Windows

Vista, 2007

Solaris

11.1, 2012.10

Mac OSX

10.5 Leopard, 2007

FreeBSD

13.0

Return-to-libc

- return-to-libc可绕过DEP防御
 - 执行代码位于位于libc库
- libc库有大量有用函数
 - 如execv(创建进程), open(打开文件)等



本节小结

- 理解两种溢出经典防御机制
- 了解Return-to-libc攻击原理

第四章 漏洞与缓冲区溢出

第四节 模糊测试(fuzzing)

主要内容和目标

- 掌握模糊测试原理
- 了解模糊测试应用场景

漏洞挖掘目标

- 如何在漏洞被攻击者利用之前找到漏洞?
- 主要方法
 - 软件测试(包括fuzzing)
 - 程序验证

软件测试

- 测试定义
 - 动态的运行一组测试用例，将实际结果与预期结果进行分析
- 测试无法保证程序的正确性，但能**有效发现bug**

功能代码

```
int foo(int a, int b) {  
    return a+b;  
}
```

测试代码

```
assert (x+y == foo(x,y))
```

模糊测试

- 模糊测试也是一种安全测试

“微软公司有一半员工是测试工程师，他们用一半时间用于测试。” – 比尔 盖茨, 1995

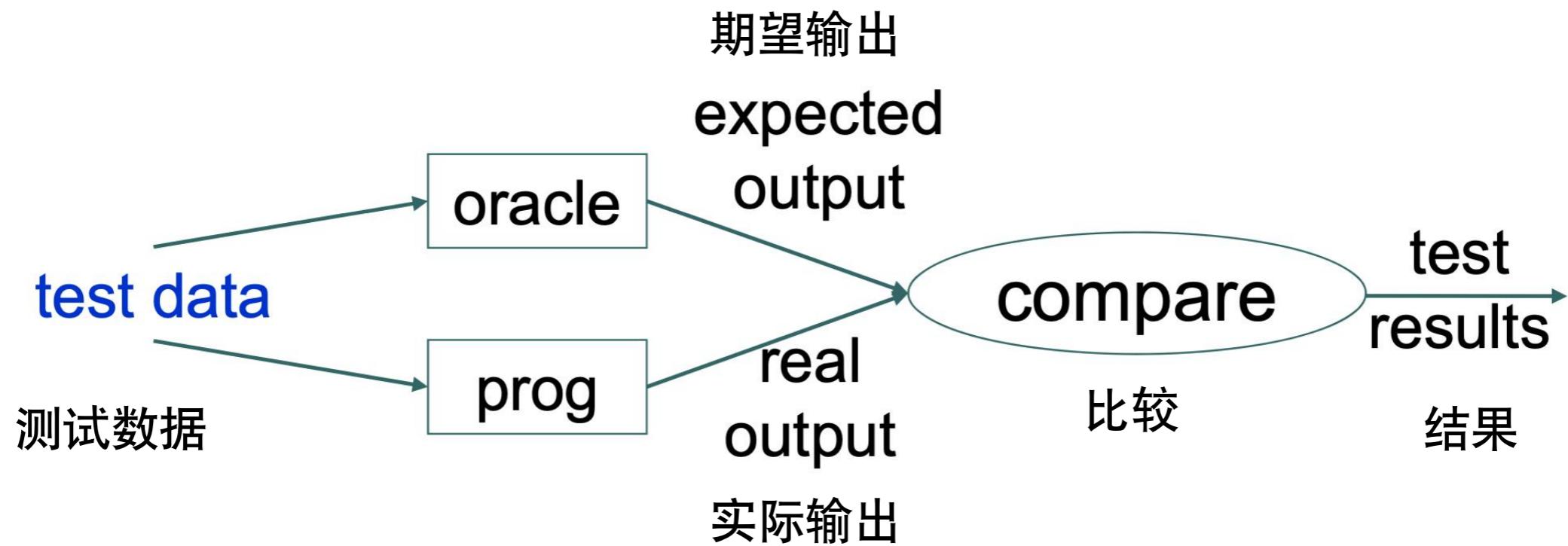
程序验证

- 程序接受输入，进行处理并输出结果。
- 程序验证：旨在自动化地证明程序的正确性。
- 程序验证通常一种静态分析方法，分析代价较高

对于如下代码，函数f对于所有n是否都适用？

```
int f (int n) {  
    y := 1;  
    z := 0;  
    while (z != n) do {  
        z := z + 1;  
        y := y * z  
    }  
    return y;  
}
```

软件测试流程



软件测试用例选择

- 测试用例选择是软件测试关键，影响：
 - 代码覆盖率
 - 路径执行深度
 - 最终影响能否有效发现bug
- 穷举所有测试用例几乎不太可能
 - 一个函数的参数为3个整数，每个整数范围为(1, 10000)
 - 每个测试用例用于测试花费1秒钟，**实际将花费多少时间(以年为单位)？**

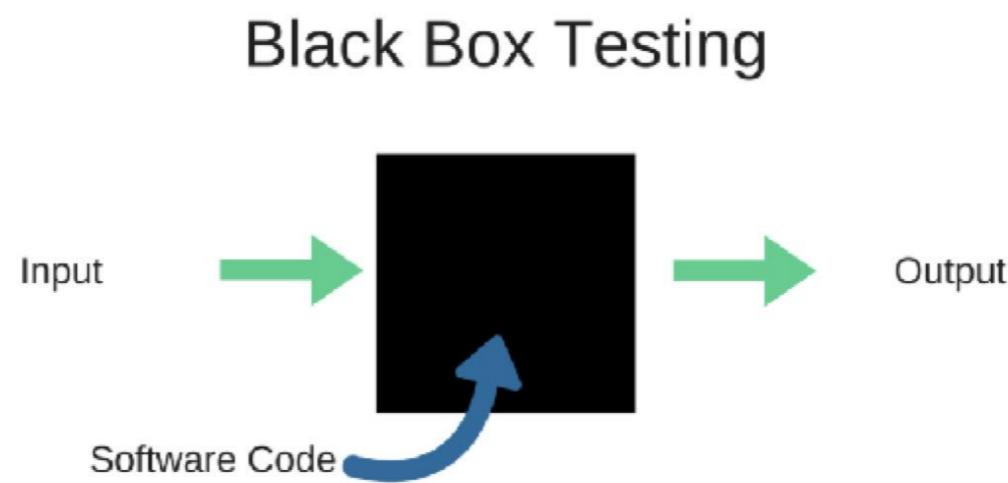
软件测试用例生成

- 如何生成有效的测试用例集?

- **黑盒测试**
- 白盒测试
- 灰盒测试

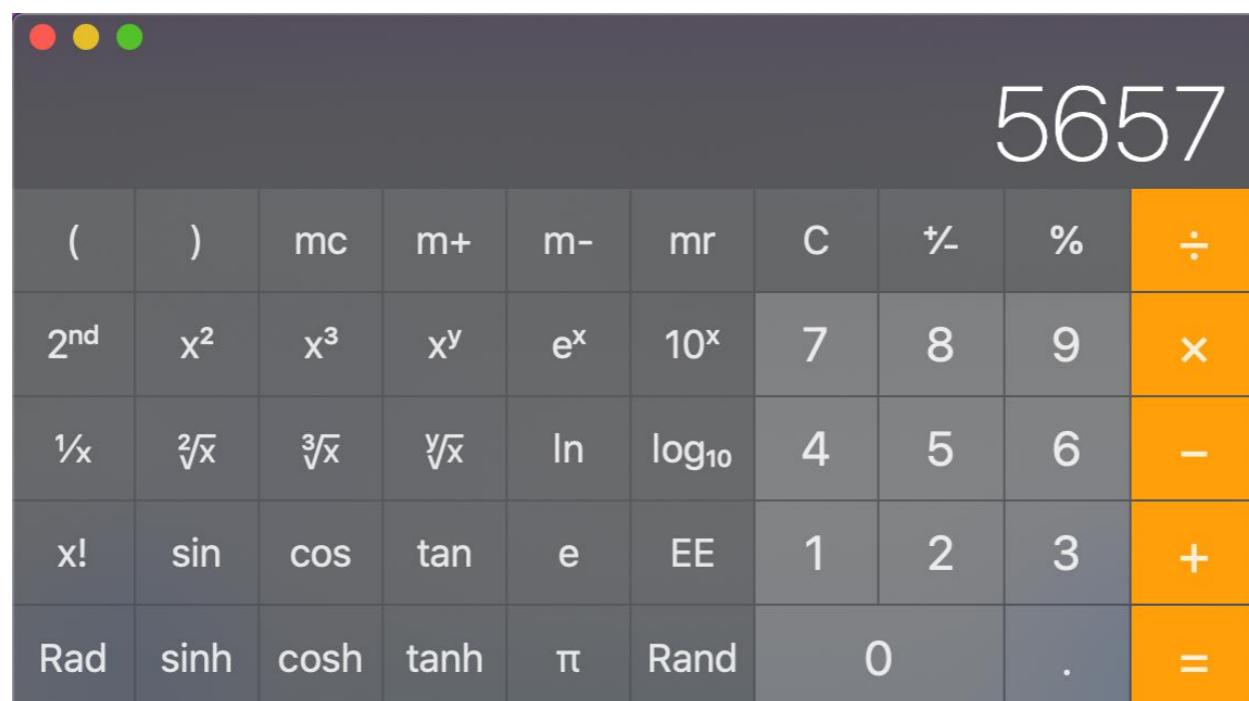
黑盒测试

- 软件测试的主要方法之一，**测试者不了解程序的内部情况，不需具备应用程序的代码**、内部结构和编程语言的专门知识。只知道程序的输入、输出和系统的功能，从用户角度针对软件界面、功能及外部结构进行测试，而不考虑程序内部逻辑结构。



黑盒测试练习

- 设计**计算器**测试用例，验证**加减乘除功能的正确性**。



黑盒测试优缺点

- 主要优点
 - 实现不复杂
 - 无需了解程序内部的代码及实现，与软件内部实现无关
- 主要缺点
 - 无法穷举全部测试用例
 - 不了解软件内部实现，导致测试用例**盲目性**
 - 当测试用例较多时，系统开销较大

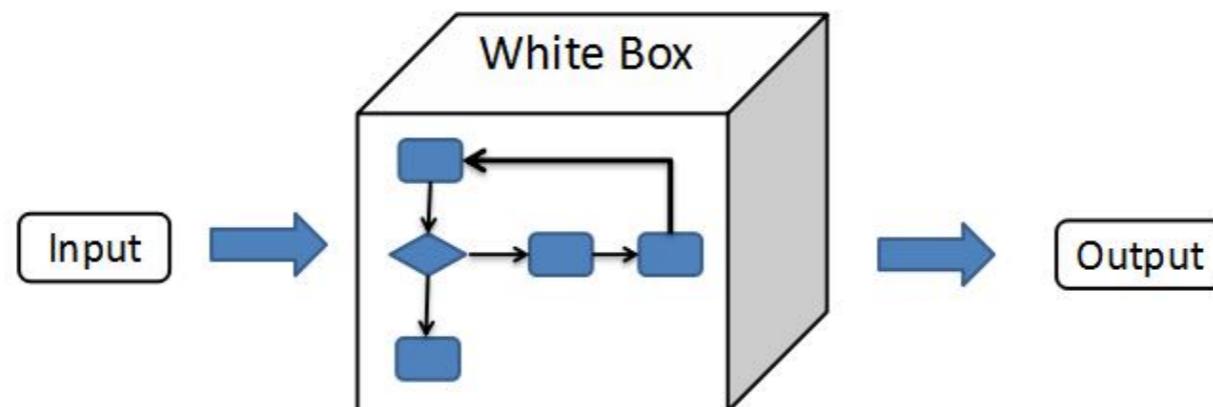
黑盒测试优缺点

- 主要优点
 - 实现不复杂
 - 无需了解程序内部的代码及实现，与软件内部实现无关
- 主要缺点
 - 无法穷举全部测试用例
 - 不了解软件内部实现，导致测试用例**盲目性**
 - 当测试用例较多时，系统开销较大

白盒测试

- 白盒测试是软件测试的主要方法之一，在**了解软件内部运作的情况下**对软件的功能和安全性等方面进行测试。

白盒测试：



白盒测试优缺点

- 主要优点
 - 了解程序内部实现，测试用例有效性较高
- 主要缺点
 - 在部分场景下程序内部原理不公开

软件测试小结

- 软件测试是保证软件质量的必要和重要组成部分
- 编写高质量的测试用例是关键
- **能否测试软件安全性，并进一步发现漏洞？**

fuzzing(模糊测试)

- 模糊测试定义

- 模糊测试(fuzzing)是一种**软件测试技术**。其核心思想是將自动或半自动生成的随机数据输入到一个程序中，并监视程序异常，如崩溃，断言(assertion)失败，以发现可能的程序错误，比如内存泄漏。模糊测试常常用于检测软件或计算机系统的安全漏洞。
- 1988年威斯康星大学Barton Miller教授发明

- 使用场景

- 软件安全性测试
- 漏洞分析与挖掘

思考：fuzzing与传统软件测试的区别是什么？

fuzzing重要性

- fuzzing是安全性测试和漏洞挖掘**最重要的方法之一**
- 谷歌2017年使用fuzzing发现超过千余个漏洞
- 微软将fuzzing作为软件开发周期一部分

谷歌Fuzz Bot在开源项目中嗅探出超过千余Bug

2017年05月10日 11:25 2916 次阅读 稿源: cnBeta.COM 0 条评论

自 Google 宣布了 OSS-Fuzz 之后的五个月里，这款工具在开源项目中嗅探出了超过 1000 多个 bug，其中包括 264 个潜在的安全漏洞。谷歌团队非常努力，每天都要处理 10 万亿的测试输入。有 47 个项目早已整合了 Fuzz，找到的千余个 bug 中它们功不可没。Google 表示：“OSS-Fuzz 已经在几个关键开源项目中找到了多个安全漏洞”。



谷歌Project Zero团队揭苹果众多新漏洞，聊天图像暗藏危机

2020-04-30 14:02:38

Apple Mail风波才刚刚过去，谷歌又炸出了苹果一波新漏洞，这一操作是否又让苹果用户的信心碎了一地？

昨日，谷歌安全团队Project Zero披露一系列苹果的安全漏洞，涉及iPhone、iPad、Mac用户。该团队利用“Fuzzing”测试发现苹果基础功能Image I/O框架中的6个漏洞，OpenEXR中的8个漏洞。

fuzzing整体流程



fuzzing目标和平台

- 应用程序
 - 文字处理软件，浏览器，音乐播放器，web程序，**操作系统**...
- 操作系统
 - Linux, Windows, OSX, Android...
- 指令集
 - x86, ARM, MIPS, ...

fuzzing分类

- **黑盒fuzzing**
 - 将目标系统或软件当作黑盒进行fuzzing
- **白盒fuzzing**
 - 根据程序内部流程进行fuzzing
- **灰盒fuzzing**
 - 结合黑盒与白盒的fuzzing

黑盒fuzzing

- **主要方法**

- 将目标系统或软件当作黑盒进行fuzzing

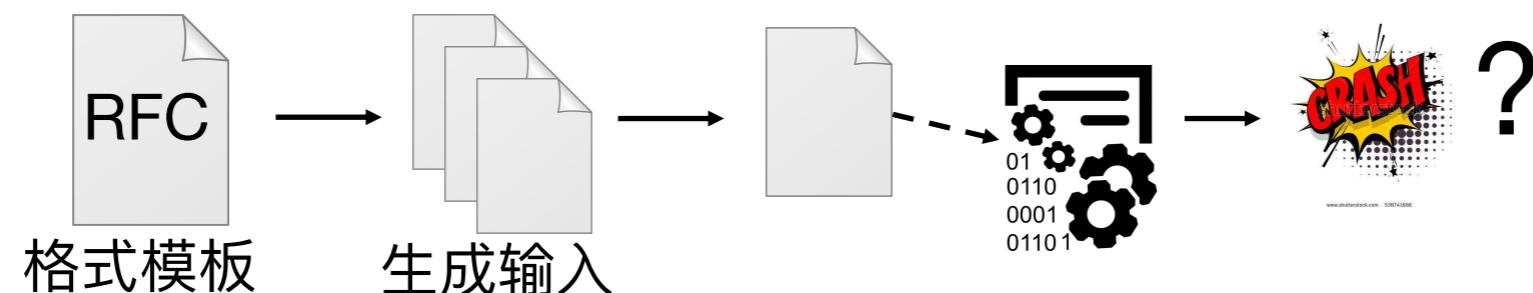
- **主要缺点**

- 盲目性较高，代码覆盖率低
- 无法执行较深路径
- 由于软件逻辑处理等原因可能出错

```
function( char *name, char *passwd, char *buf )
{
    if ( authenticate_user( name, passwd ) ) {
        if ( check_format( buf ) ) {
            update( buf ); // crash here
        }
    }
}
```

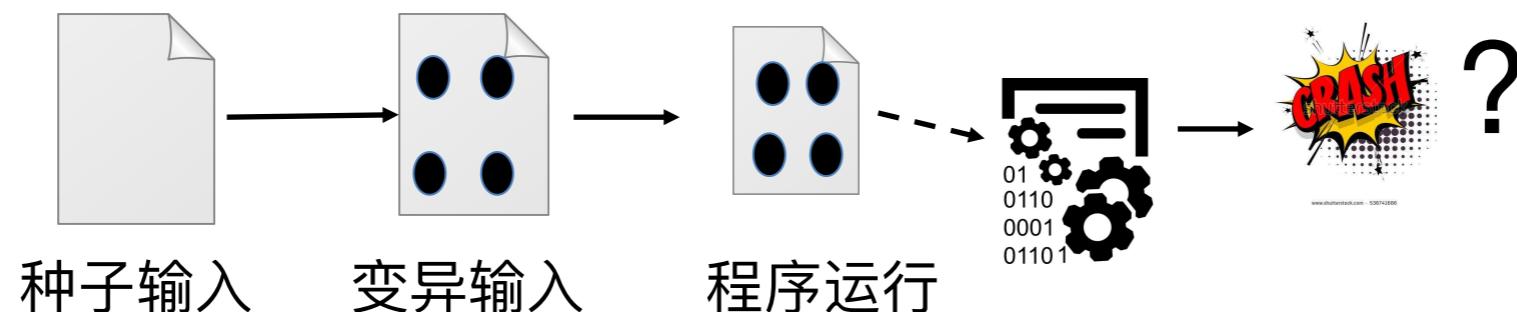
基于生成的用例生成

- 对输入格式有一定了解，如网络协议、文件格式
- 改变关键字段生成测试用例
- 常见工具：SPIKE, Sully等

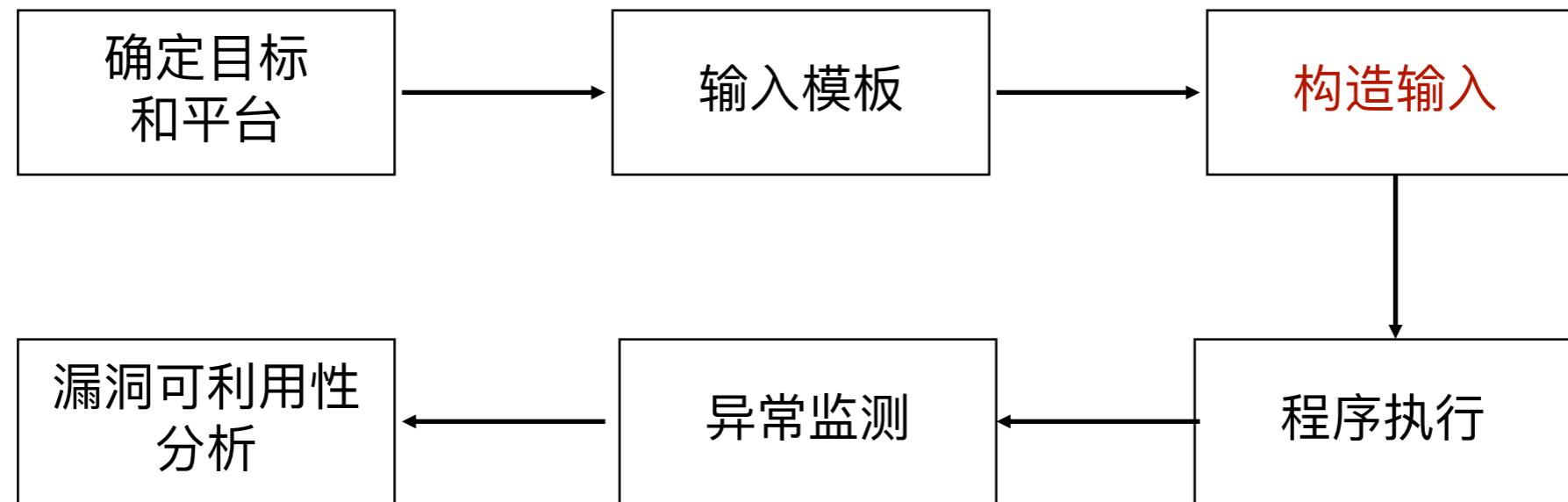


基于变异的用例生成

- 对输入格式**不了解**的情况下进行fuzzing
- 选择一个**种子文件**为输入，并改变输入内容生成测试用例
- 常见工具：**AFL**, ZZuf, FileFuzz等



基于生成的fuzzing流程



两种方法的比较

- 基于生成的fuzzing
 - 对输入格式有要求
 - 编写测试用例生成器有一定难度
- 基于变异的fuzzing
 - 上手快，对输入格式无要求
 - 对种子输入有依赖性

程序异常监测

- 程序异常
 - 使用调试器，如GDB、OllyDbg、WinDbg
- 内存错误
 - Valgrind等内存分析软件

如何检测操作系统崩溃？

Fuzzing评价指标

深度Fuzzing技术研究水位表												
相关指标	一级指标	二级指标	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8	Level 9	Level 10
平台/架构/对象支持	架构支持		x86	+x64	+arm32	+arm64	+mips32	+mips64	+ppc32	+ppc64	+stm32	+risc-v
	操作系统支持		windows	+linux	+android	+macosx	+ios	+vxworks	+rtos	+raw	同左	同左
	分析对象支持		app	net	+驱动	+os内核	+hypervisor	同左	同左	同左	同左	同左
输入/变异策略支持	输入类型支持		二进制	同左	同左	+结构文本	同左	同左	+脚本	同左	+复杂混合	同左
	变异策略支持		二进制随机	同左	同左	+结构文本	同左	同左	+脚本	同左	+复杂混合	同左
测试效率	反馈效率		N/A	N/A	+软插桩	同左	同左	+硬件特性	同左	同左	同左	同左
	测试效率损失		MAX	-1/9	-1/9	-1/9	-1/9	-1/9	-1/9	-1/9	-1/9	MIN
	异构测试效率损失		MAX	-1/9	-1/9	-1/9	-1/9	-1/9	-1/9	-1/9	-1/9	MIN
	集群支持		+单CPU	+多核	同左	同左	+小规模集群	同左	同左	+大规模集群	同左	同左
覆盖率	是否支持覆盖率反馈		+基本块	+Edge	+分支	+状态	+哈希校验	+间接变量	+环境变量	+漏洞	同左	
	基本块覆盖率		MIN	+1/8	+1/8	+1/8	+1/8	+1/8	+1/8	+1/8	+1/8	MAX
	Edge覆盖率		MIN	+1/8	+1/8	+1/8	+1/8	+1/8	+1/8	+1/8	+1/8	MAX
	漏洞覆盖率		MIN	+1/8	+1/8	+1/8	+1/8	+1/8	+1/8	+1/8	+1/8	MAX
	分支覆盖率				MIN	+1/6	+1/6	+1/6	+1/6	+1/6	+1/6	MAX
	状态覆盖率				MIN	+1/6	+1/6	+1/6	+1/6	+1/6	+1/6	MAX
	哈希校验等复杂函数保护代码可达率				MIN	+1/6	+1/6	+1/6	+1/6	+1/6	+1/6	MAX
	间接循环变量保护代码可达率				MIN	+1/6	+1/6	+1/6	+1/6	+1/6	+1/6	MAX
	环境变量保护代码可达率								MIN	+1/3	+1/3	MAX
功能支持	漏洞检测能力	crash	MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
		SOF	MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
		HOF	MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
		OOB	MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
		UAF	MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
		Uninit	MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
		TC	MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
		命令注入	N/A	N/A	N/A	MIN	+1/6	+1/6	+1/6	+1/6	+1/6	MAX
		其它逻辑类	N/A	N/A	N/A	MIN	+1/6	+1/6	+1/6	+1/6	+1/6	MAX
		其它	N/A	N/A	N/A	MIN	+1/6	+1/6	+1/6	+1/6	+1/6	MAX
	Crash复现率		MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
	自动样本精简成功率		MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
	漏洞样本消重成功率		MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
	directed Fuzz		MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX
	自动化攻击面分析（成功率）	单程序						MIN	+1/4	+1/4	+1/4	MAX
		全系统							MIN	+1/3	+1/3	MAX
漏洞检出数量			MIN	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	+1/9	MAX

https://www.sohu.com/a/438559568_104421

Zhi Liu, copy rights reserved

常见fuzzing工具

- AFL
 - Peach
 - Spike
- ...

本节小结

- fuzzing测试原理
- 两种测试用例生成方法