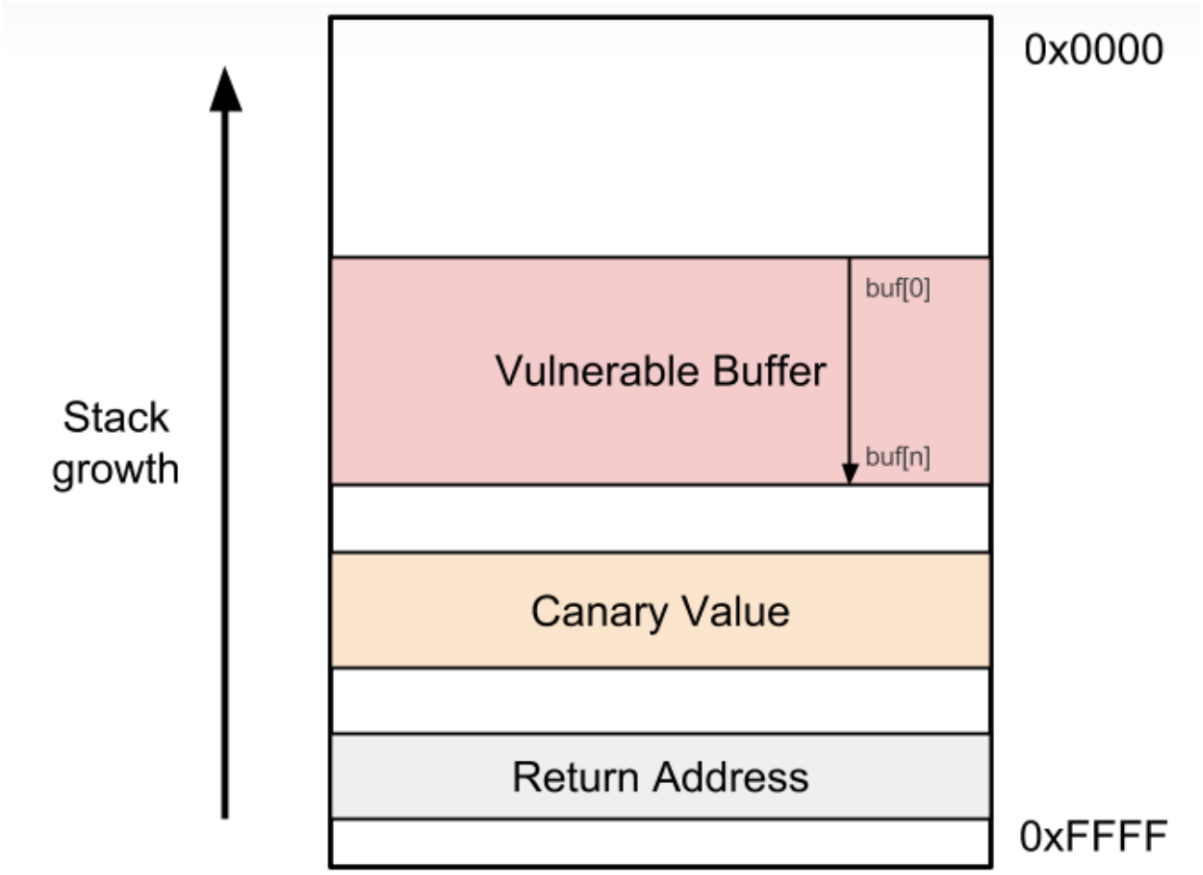
Stack Canaries 栈金丝雀

Canary的值是栈上的一个**随机数**，在程序启动时随机生成，并保存在比函数返回地址**更低**的位置。

由于栈溢出是从低地址向高地址进行覆盖，因此攻击者要想控制函数的**返回指针**，就一定要先覆盖到Canary。

程序只需要在函数返回前检查Canary是否被篡改，就可以达到保护栈的目的。



canaries 通常可分为3类：

- terminator
- random
- random XOR

具体的实现有StackGuard，StackShield，proPoliced等。

· Terminator canaries：

由于许多栈溢出都是由于字符串操作（如 strcpy）不当所产生的，而这些字符串由 **NULL“\x00”** 结尾，换个角度看就是会被 “\x00” 所截断。基于这一点，terminator canaries将低位设置为“\x00”，既可以防止被泄露，也可以防止被伪造。

截断字符还包括 **CR(0x0d)**，**LF(0x0a)**，**EOF(0xff)**。

· Random canaries：

为了防止canaries被攻击者猜到，random canaries通常在程序初始化时就随机生成，并保存在一个相对安全的地方。当然如果攻击者知道他的位置，还是有可能被读出来的。

随机数通常由**/dev/urandom**生成，有时也使用**当前时间的哈希。**

· Random XOR canaries：

与random canaries类似，但是多了一个**XOR操作**，这样无论是canaries被篡改还是与之XOR的控制数据被篡改，都会发生错误，这样就增加了攻击难度。

一，Canaries 的效果

在 Linux 系统中，可以通过 GCC 编译器的选项来开启 Stack Canary。

GCC包含多个与Canaries有关的参数，常见的有：

参数	作用
-fstack-protector	对alloca系列函数和内部缓冲区大于八个字节的函数启用保护
-fstack-protector-strong	增加对包含局部数组定义和地址引用的函数保护
-fstack-protector-all	对所有函数启用保护
-fstack-protecto-explicit	对包含stack protect属性的函数启用保护
-fno-stack-protector	禁用保护

1. 默认情况：GCC 编译器在默认情况下不会开启 Stack Canary

```
gcc -o test test.c
```

2. 开启 Stack Canary：
    - -fstack-protector：只对局部变量中含有字符数组的函数插入保护代码

      ```
      gcc -fstack-protector -o test test.c
      ```

    - -fstack-protector-all：为所有函数插入保护代码

      ```
      gcc -fstack-protector-all -o test test.c
      ```

    - -fstack-protector-strong（GCC 4.9 及以上版本）：提供更广泛的保护

      ```
      gcc -fstack-protector-strong -o test test.c
      ```

3. 关闭 Stack Canary：

   ```
   gcc -fno-stack-protector -o test test.c
   ```

二，Canaries的实现

32位

- 存储位置：通常在栈帧中 **EBP - 0x4** 的位置
- 获取方式：从 **gs:0x14**（TLS）中获取一个4字节的随机值作为 Canary
- 大小：4字节

64位

- 存储位置：通常在栈帧中 **RBP - 0x8** 的位置
- 获取方式：从 **fs:0x28** 中获取一个8字节的随机值作为 Canary
- 大小：8字节
- Canary 最低位有时被设置为 0x00，可以防止其在存储时被意外截断。
- 如果 dl_random 指针为 NULL，意味着随机值尚未初始化，此时 Canary 可能会使用一个默认值（如全零或其他固定值）。

canaries的代码实现

攻击Canaries的主要目的是避免程序崩溃，那么就有两种思路：

- 第一种将Canaries的值泄露出去，然后在栈溢出时覆盖上去，使其保持不变
- 第二种则是同时篡改TLS和栈上的Canaries，这样在检查的时候就能通过。

三，窃取Canaries

泄露栈金丝雀代码

是否能基于MA实现?

绕过思路：

- 通过某些手段（如利用格式化字符串漏洞）将canary泄露
- 利用栈溢出，用垃圾数据覆盖栈上数据时，保证canary数据不变。
- 程序检查canary没有改变，继续正常执行。

代码如下：

```
#include <stdio.h>

void shell(){
    system("/bin/sh");  // 执行/bin/sh命令，启动一个新的shell
}
```

```
// 缓冲区溢出漏洞
void vulnerable(){
    char buf[12];  //buf分配12字节空间
    puts("input 1:");
    read(0, buf, 100);  // 输入特定数量的字符串，将buf与canary连接起来
    puts(buf);  // 通过缓冲区溢出将canary写入buf，输出
    puts("input 2:");
    fgets(buf, 0x100, stdin);  // 从标准输入读取最多256字节到buf中，不会导致溢出
}

void main(){
    vulnerable();
}
```

编译链接

```
// gcc编译器默认开启canary保护
// 关闭栈保护 -fno-stack-protector
// 打开栈保护 -fstack-protector-all
gcc main.c -m32 -fstack-protector-all -no-pie -o canary
```

在终端使用 checksec 检查 canary 是否启动

```
sudo apt-get install checksec   //安装checksec
```

```
checksec canary
```

如果出现 Stack：Canary found 则说明 Canary 已启动。

查看vulnerable函数的反汇编代码。check canary部分显示了程序将在全局段某处（gs:0x14）取出的数据作为canary放置于栈帧中的位置。

四，栈溢出攻击复现

栈溢出攻击复现代码详解

CVE-2013-2028：nginx 栈溢出漏洞

CVE-2020-8423：TP-Link WR841N 栈溢出漏洞