计算机科学
COMPUTER SCIENCE

基于数据流特征的比较类函数识别方法

胡安祥，尹小康，朱肖雅，刘胜利

引用本文

胡安祥，尹小康，朱肖雅，刘胜利. 基于数据流特征的比较类函数识别方法[J]. 计算机科学, 2022, 49(9): 326-332.

HU An-xiang, YIN Xiao-kang, ZHU Xiao-ya, LIU Sheng-li. Strcmp-like Function Identification Method Based on Data Flow Feature Matching[J]. Computer Science, 2022, 49(9): 326-332.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220200163

基于数据流特征的比较类函数识别方法

胡安祥　尹小康　朱肖雅　刘胜利
数学工程与先进计算国家重点实验室(信息工程大学)　郑州 450001
(hasafox@163.com)

摘　要　嵌入式设备已经随处可见,它们常常出现在安全领域的关键位置和靠近终端的隐私场所。然而,最近的研究表明,很多嵌入式设备存在后门,发现最多的为硬编码后门(即口令后门)。在口令后门的触发过程中,字符串比较函数(比较类函数)是不可或缺的,其重要性不言而喻。目前,针对比较类函数的识别主要借助于函数签名和控制流特征的匹配,前者不适用于对未知的比较类函数进行识别,并且受编译环境的影响较大,后者具有较高的误报率和漏报率。针对上述问题,提出了一种新颖的比较类函数识别方法 CMPSeek。该方法在函数控制流的基础上,对比较类函数的数据流特征进行分析并构建了识别模型,用于对二进制程序中比较类函数的识别,并且适用于剥离的二进制程序(Stripped Binary)。此外,将二进制代码转换为中间语言 VEX IR 指令,以支持 ARM,MIPS,PowerPC(PPC)和 x86/64 指令集。实验结果表明,当缺少源码、函数名等信息时,相比 FLIRT 和 SaTC,CMPSeek 在精准率和召回率上都有着更好的结果。

关键词:函数识别;数据流分析;特征匹配;字符串比较函数;二进制程序
中图法分类号　TP393

Strcmp-like Function Identification Method Based on Data Flow Feature Matching

HU An-xiang,YIN Xiao-kang,ZHU Xiao-ya and LIU Sheng-li
State Key Laboratory of Mathematical Engineering and Advanced Computing,Zhengzhou 450001,China

Abstract　Embedded devices have become visible everywhere,and they are used in a range of security-critical and privacy-sensitive applications. However,recent studies show that many embedded devices have backdoor,of which hard-coded backdoor(password backdoor) is the most common. In the triggering process of password backdoor,strcmp-like functions are necessary and important absolutely. However,the current identification of strcmp-like functions mainly relies on function signature and control flow feature matching. The former can't recognize user-defined strcmp-like functions,and the identify effect is greatly affected by the compile environment. The latter has high false positive rate and false negative rate. To solve the above problems,this paper proposes a novel strcmp-like recognition technology CMPSeek. This method builds a model for strcmp-like function identification based on the analysis of control flow and data flow characteristics,which is used to identify strcmp-like functions in binary programs,and is suitable for stripped binary programs. Furthermore,ARM,MIPS,PPC and x86/64 instruction sets are supported by converting binary codes to the intermediate language representation VEX IR codes. Experimental results show that CMPSeek has better results in accuracy rate and recall rate than FLIRT and SaTC in the absence of source code,function name and other information.

Keywords　Function identification,Data flow analysis,Feature matching,Strcmp-like function,Binary program

1　引言

随着物联网的快速发展,越来越多的嵌入式设备走进了人们的生活,它们(如路由器、交换机和打印机等设备)遍布于日常生活中的方方面面。然而,最近的研究表明,市场上有很多嵌入式设备存在后门,其中披露最多的为硬编码后门(即口令后门)[1]。如 2013 年在 D-Link 路由器中发现的口令后门,将浏览器的 User-Agent 字段设置为字符串"xmlset_roodkcableoj28840ybtide"后就可以远程控制路由器[2];2016 年在 FortiGate 防火墙中发现的 SSH 后门,硬编码的后门口令为"FGTAbc11*xy+Qqz27"[3];2021 年在合勤科技公司 Zyxel 的防火墙、VPN 等设备中发现的具有管理员权限的账户{username:"zyfwp",password:"PrOw! aN_fXp"},使用该账户就可以通过 SSH 或者 Web 管理面板对设备进行访问[4]。在口令后门的触发过程中,字符串比较函数(简称比较类函数)用于对口令后门的校验,起着至关重要的作用。下面以 CVE-2015-7755 后门为例,说明比较类函数识别对嵌入式设备中口令后门的检测与发现的重要性。

到稿日期:2022-02-25　返修日期:2022-06-03
基金项目:科技委基础加强重点项目(2019-JCJQ-ZD-113)
This work was supported by the Foundation Strengthening Key Project of Science & Technology Commission(2019-JCJQ-ZD-113).
通信作者:刘胜利(mr_shengliliu@163.com)

2015 年在 Juniper 的 ScreenOS 防火墙中发现的口令后门(CVE-2015-7755)在远程连接时使用指定字符串“<<< %s(un='%s')=%u”就能绕过登录流程，获取管理员权限[5]。使用 IDA Pro 工具逆向分析 ssg5ssg20.6.3.0r19.0.bin 固件，在固件中 loc_13DC5C 处调用 sub_ED7D94 函数以比较输入参数和字符串“<<< %s(un='%s')=%u”，实现绕过登录流程功能，如图 1 所示。图 1 中的 sub_ED7D94 函数为自定义实现的比较类函数，在 sub_ED7D94 中通过循环(loc_ED7DDC，ED7DE4，loc_ED7DDC)实现比较字符串的功能，如图 2 所示。

```
loc_13DC5C
ADD     R0, R5, #0x44
LDR     R1, =aSUnSU ; "<<< %s(un='%s')=%u"
BL      sub_ED7D94
CMP     R0, #0
BNE     loc_13DC78
MOV     R0, #0xFFFFFFFD
LDMDB   R11, {R4-R8,R11,SP,PC}
```

图 1 CVE-2015-7755 后门验证代码

Fig. 1 Check codes of CVE-2015-7755 backdoor

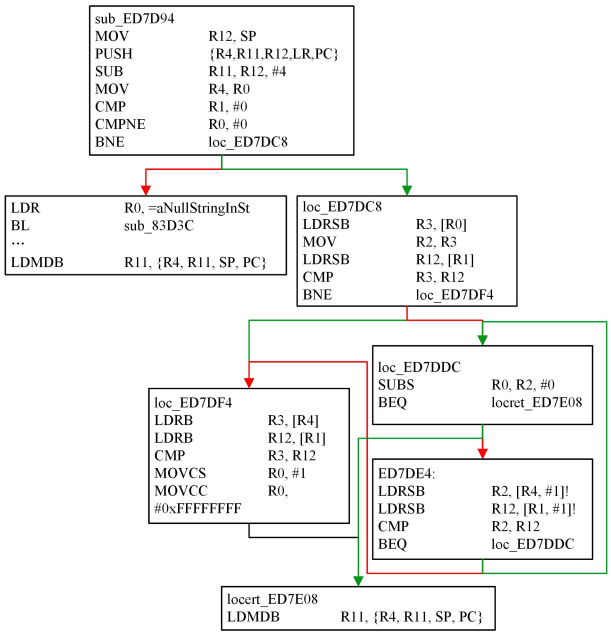


图 2 sub_ED7D94 函数

Fig. 2 Function sub_ED7D94

本文通过分析比较类函数在控制流和数据流上的特征，构建了二进制程序中的比较类函数识别模型，并提出了一种新颖的比较类函数识别技术 CMPSeek。该方法在控制流分析的基础上，结合比较类函数的执行流程在数据流上的表现，来识别二进制程序中的比较类函数。由于不依赖于源码、函数名等信息，该方法适用于剥离的二进制程序，并且受编译器的影响较小，有效提高了口令后门的检测效果。本文的主要贡献如下：

(1)分析比较类函数源码，将其编译成不同指令集架构，提取控制流和数据流的特征，构建基于数据流特征匹配的比较类函数识别模型。

(2)提出并实现了比较类函数识别方法 CMPSeek，该方法不依赖源码、函数名等信息，识别效果几乎不受编译环境影响，能够识别任何方式实现的比较类函数，支持 ARM，MIPS，PowerPC(PPC)和 x86/64 指令集，具有良好的适用性。

(3)从 GitHub，Stack Overflow 等平台上收集比较类函数，构建开源库函数数据集 A 和自定义实现函数数据集 B 进行测试。实验结果表明，在有源码的情况下，CMPSeek 的函数识别效果虽然不如 FLIRT，但也有着不错的表现；当缺少源码、符号表等信息时，相比 FLIRT 和 SaTC，CMPSeek 在精准率和召回率上都有着更好的结果。

2 相关工作

本节首先介绍函数识别在嵌入式设备中的应用，然后介绍已有的 3 种函数识别方法，即 FLIRT[6]，Karonte[7] 和 SaTC[8]，最后讨论这 3 种方法的缺陷和不足。

在嵌入式设备中基于比较类函数的漏洞挖掘技术的研究颇多。2015 年 Shoshitaishvili 等[9] 提出的二进制分析框架 Firmalice 借助 strcmp 和 strncmp 等函数来定位关键路径，对关键路径进行切片，通过静态符号执行来判断是否存在确定性约束，若存在，则有后门。2017 年 Thomas 等[10] 提出的基于静态数据权重比较的后门识别方法 Stringer 的原理为，利用 strcmp，memcmp 和 strstr 等函数来查找参与比较的字符串，然后利用提出的算法计算字符串的权重，并根据权重大小识别二进制程序的口令后门。2020 年 Redini 等[7] 提出的静态污点分析框架 Karonte 通过建模和跟踪多二进制文件交互分析固件，在二进制文件之间传播污点信息以检测不安全的交互并识别漏洞，如图 3 所示，该方法仍需要借助 strcmp 和 strncmp 等函数的信息静态跟踪和传播污点数据。2021 年 Chen 等[8] 提出的污点分析工具 SaTC 匹配前端的关键字和后端的输入项，对敏感输入进行污点分析来识别漏洞，如图 4 所示，SaTC 同样需要 strcmp 和 strncmp 等函数来定位敏感输入。

```
def get_memcp_like(p):
    addrs = get_dyn_sym_addrs(p, ['strcmp', 'strcasecmp'])
    summarized_f = {}
    for f in addrs:
        summarized_f[f] = summary_functions.memcmp_unsized
    return summarized_f

def get_memncp_like(p):
    addrs = get_dyn_sym_addrs(p, ['strncasecmp', 'strncmp'])
    summarized_f = {}
    for f in addrs:
        summarized_f[f] = summary_functions.memcmp_sized
    return summarized_f
```

图 3 Karonte 中的识别比较类函数

Fig. 3 Identify strcmp-like function in Karonte

```
def get_memcmp_like_unsized(p):
    addrs = get_dyn_sym_addrs(p, ['strcmp', 'strcasecmp'])
    summarized_f = {}
    for f in addrs:
        summarized_f[f] = summary_functions.memcmp_unsized
    return summarized_f

def get_memcmp_like_sized(p):
    addrs = get_dyn_sym_addrs(p, ['strncasecmp', 'strncmp'])
    summarized_f = {}
    for f in addrs:
        summarized_f[f] = summary_functions.memcmp_sized
    return summarized_f
```

图 4 SaTC 中的识别比较类函数

Fig. 4 Identify strcmp-like function in SaTC

当前剥离的二进制文件函数识别技术以静态分析方法为主。如 IDA Pro 中的库文件快速识别与鉴定技术(Fast Li-

brary Identification and Recognition Technology, FLIRT), 首先分析二进制程序,找到程序使用的库函数名称和版本信息,然后在 GitHub 或者其他开源平台下载库函数的源码,编译源码为目标程序架构,选取函数二进制代码的前 32 位字节制作函数签名,最后根据签名匹配的方式识别程序中的函数。然而,FLIRT 方法需要源码,函数识别效果会受到编译器类型、编译器版本、优化等级、优化选项等的影响[11]。在 Karonte 和 SaTC 中都提出了基于特征匹配的静态函数识别方法,但所采取的特征比较简单:有循环结构以及循环中有自增指令和比较指令。然而,满足上述条件的函数不一定是比较类函数,因此 Karonte 和 SaTC 方法具有较高的误报率和漏报率。

当前的函数识别方法存在以下问题:FLIRT 方法一般通过源码制作函数签名,恢复剥离程序中的函数名称等信息,然而函数签名可能会发生冲突,识别效果依赖于编译环境,并且无法识别用户自定义的函数;Karonte 和 SaTC 方法采用的函数特征较为简单,具有较高的误报率和漏报率。为了解决二进制程序中比较类函数的识别问题,本文提出了一种基于数据流特征匹配的函数识别方法 CMPSeek,分析函数的控制流,从中提取关键路径,构建数据流并检测数据流特征是否匹配;将路径中的二进制代码转换为中间语言 VEX IR 指令,支持 ARM, MIPS, PPC 和 x86/64 指令集;不依赖源码等信息,并且受编译器选项、编译器版本以及优化等级的影响较小,对 glibc 库和自定义实现的比较类函数都有很好的识别效果。实验评估表明,在有源码的情况下,CMPSeek 方法的函数识别效果虽然不如 FLIRT,但也有着不错的表现;当缺少源码、函数名等信息时,相比 FLIRT 和 SaTC, CMPSeek 有着更高的精准率和召回率。

3 比较类函数识别方法 CMPSeek

本节首先对比较类函数进行分析,然后介绍 CMPSeek 对比较类函数的识别流程,提出了算法并说明了算法的具体实现过程,最后以 strcmp 函数为例论述 CMPSeek 方法的有效性。

3.1 比较类函数分析

C 语言 glibc 库中的比较类函数较多,常见的有 strcmp, strncmp 和 memcmp。下面重点分析 strcmp 函数,其源码如图 5 所示。

```
int STRCMP (const char *p1, const char *p2)
{
  const unsigned char *s1 = (const unsigned char *) p1;
  const unsigned char *s2 = (const unsigned char *) p2;
  unsigned char c1, c2;

  do
    {
      c1 = (unsigned char) *s1++;
      c2 = (unsigned char) *s2++;
      if (c1 == '\0')
        return c1 - c2;
    }
  while (c1 == c2);

  return c1 - c2;
}
```

图 5 glibc 库中 strcmp 函数的源码

Fig. 5 Function strcmp source codes in glibc Library

strcmp 函数借助 do…while 循环结构依次比较字符,当字符不相等或者遇到'\0'终止符时,跳出循环,返回比较结果;如果字符串相等,则返回 0。实际上 glibc 库中的比较类函数往往借助 for 或者 while 循环来实现字符串的比较功能。

使用 GCC 编译器将 strcmp 函数编译为 ARM 指令集架构,经 IDA Pro 逆向工具反汇编后的 strcmp 函数如图 6 所示。源码中的循环结构经过编译器编译后在控制流图(Control Flow Graph, CFG)中的表现为存在循环路径。

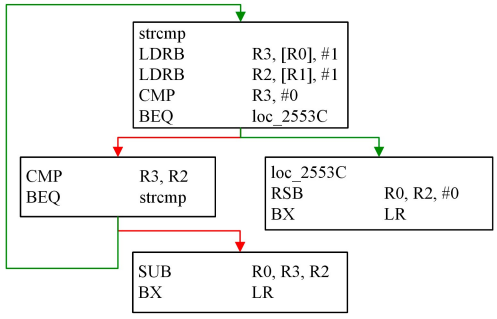


图 6 ARM 架构下的 strcmp 函数

Fig. 6 Function strcmp in ARM architecture

为了实现字符串比较功能,比较类函数有以下特征。

(1)存在循环结构。无论是源码中的 for 或者 while 指令,还是二进制程序中的 CFG,比较类函数都一定有循环。

(2)存在比较操作。在源码中比较字符是否相等,在 ARM, PPC 和 x86/64 架构中,实现方式为比较指令,如图 6 中的 CMP 指令,在 MIPS 中虽然没有比较指令,但通过算术指令 subu 等和跳转指令 bnez/beqz 来实现比较操作。

(3)循环中存在自增操作。无论是源码中还是二进制程序中,都存在字符串下标自增 1 个单元的操作。例如,图 6 中的 LDRB R3,[R0],#1 和 LDRB R2,[R1],#1。

(4)函数的返回值源自比较运算的结果。如图 6 所示,strcmp 函数返回值来源于 RSB R0, R2, #0 和 SUB R0, R3, R2(ARM 中函数返回值存放于寄存器 R0 中)。

对具有字符串比较功能的函数进行研究发现,比较类函数可能不单单实现比较两个字符串是否相等的功能,它可能具备更多其他的功能,例如将字符串"abcdbcade"按字符串"bc"切片为数组["a""d""ade"],因此可以将比较类函数分为狭义的比较类函数和广义的比较类函数。狭义的比较类函数只完成字符串比较的功能;广义的比较类函数除了完成字符串比较的功能外,还存在更多的功能。后者往往比前者更加复杂,外在表现为源码和控制流更多,编译成二进制程序时则是基本块更多。通常情况下狭义的比较类函数的基本块数量小于 80(这里阈值的设置基于对当前的比较类函数分析所得,实际应用中可根据具体的需求设置阈值)。

3.2 CMPSeek 流程和算法

比较类函数识别技术 CMPSeek 的流程如图 7 所示。

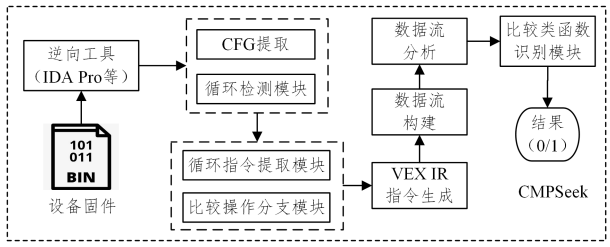


图 7 CMPSeek 流程图

Fig. 7 CMPSeek flowchart

CMPSeek 总共分为 7 个模块：CFG 提取和循环检测模块、循环指令提取模块、比较操作分支模块、VEX IR 指令生成模块、数据流构建模块、数据流分析模块和比较类函数识别模块。其中，CFG 提取和循环检测模块主要用于分析函数中是否包含循环路径；循环指令提取模块和比较操作分支模块分别用于提取循环块内和循环内比较操作分支到函数返回的指令；VEX IR 指令生成模块借助开源库 pyvex[12] 将二进制代码转换为中间语言 VEX IR 指令，方便系统支持多种指令集架构；数据流构建和数据流分析模块从 VEX IR 指令中提取数据流中的变量关系，生成数据流图(Data Flow Graph，DFG)并进行分析；比较类函数识别模块根据数据流特征检测循环路径和分支路径是否匹配，以此判断函数是否为比较类函数。具体的比较类函数识别算法如算法 1 所示。

算法 1 比较类函数识别算法

```
输入：(ea) / * 函数起始地址 * /
输出：(0/1) / * 是否为比较类函数 * /
1. func ←get_func(ea)
2. V ← [],E ← []
3. for block in FlowChart(func) do
4. V ← V∪block / * 获取顶点 * /
5. for succor in block. succs() do
6. E ← E∪(block,succor) / * 获取有向边 * /
7. end for
8. end for
9. cfg ←generate_cfg(V,E) / * 生成控制流图 * /
10. loops ←simple_cycles(cfg) / * 提取循环路径 * /
11. if loops ≠ Ø
12. branchs ←get_branch(cfg) / * 提取分支路径 * /
13. dfg ←generate_dfg(loops,branchs) / * 生成数据流图 * /
14. if is_cmp_func(dfg) / * 数据流特征是否匹配 * /
15. return 1
16end if
17end if
18return 0
```

通过 3.1 节中对比较类函数的分析可知，比较类函数需要借助循环来实现字符串比较功能，因此循环是比较类函数识别的基础。CMPSeek 根据函数的 CFG 判断是否存在循环。将函数 CFG 转换为有向拓扑图 G，然后根据图论中的相关知识判断 G 中是否存在循环。将基本块视为顶点 v，基本块之间的跳转关系视为有向边 e，有向图 G 表示如下：

$$G=\{V,E\}$$

$$V=\{v_1,v_2,v_3,\cdots,v_n\}$$

$$E=\{e_1,e_2,e_3,\cdots,e_m\}$$

$$e_k=\{v_i,v_j\}(1\leqslant k\leqslant m,1\leqslant i,j\leqslant n)$$

IDA Pro 内置的 IDAPython 插件[13] 提供了 *FlowChart*() 类和 *self.succs*() 函数来获取函数基本块和基本块间的关系，因此直接调用 *FlowChart*() 和 *self.succs*() 来获取 V 和 E(算法 1 中第 3－8 行)。当前对于识别有向图中是否有循环已经有非常成熟的解决方案，这里使用开源库 NetworkX[14] 的 DiGraph() 函数来快速构建有向图 G(算法 1 中的第 9 行)，并调用 *simple_circles*() 识别图 G 中是否存在循环(算法 1 中的第 10 行)。当 G 中包含循环路径时，提取分支路径(算法 1 中的第 12 行)并进行后续的比较类函数识别，否则认定其为非比较类函数。

通过 CFG 提取循环路径和分支路径后，分析路径中的数据流向，如图 6 中汇编命令 *LDRB R3*，[*R0*]，#1，数据流向为 *R0*→*R3*，*R0*+1→*R0*。将寄存器或者内存地址视为顶点 v，数据的传递方向转为有向边 e，按相同的方式构建 DFG(算法 1 中的第 13 行)。为了更好地说明数据流中的特征，给出如下定义。

(1)*Cz* 点：数据流中与零进行比较的值。在字符串比较过程中，需要确定字符串的终止符'\0'。

(2)*Add* 点：数据流中自增运算的值。字符串按顺序进行比较，下标会进行自增操作，有时循环比较单个字符，有时在一次循环中比较 4 个字符。

(3)*Cmp* 对：数据流中参与比较的值，即参与比较的一对字符。

(4)*Rt* 点：数据流中函数的返回结果。当 *Cmp* 对不相等时，会跳出循环，函数返回 *Cmp* 对的差值。需要注意的是，在不同架构中函数返回值存放的寄存器不同。表 1 列出了摘要以及 IR 指令举例。

表 1 数据流中的特殊值

Table 1 Special values in dataflow

Name	Interpretation	IR Example
Cz 点	数据流中与零进行比较的值(t6)	t9＝CmpEQ(t6,0x00)
Add 点	数据流中加 1 运算的值(t7)	t7＝Add(t8,0x01)
Cmp 对	数据流中参与比较的值([t2,t3])	*t*4＝CmpEQ(t2,t3)
Rt 点 *	数据流中函数返回结果(r0)	PUT(r0)＝t52

注：在不同架构中函数返回值存放的寄存器不同，如 ARM 中为 r0，MIPS 中为 *v*0－*v*1，表 1 所列为 ARM 架构

通过分析发现比较类函数 DFG 具有以下 4 个特征：

(1)数据流上存在环结构，并且环上存在比较运算和算术运算；

(2)数据流上存在 *Cz* 点和 *Cmp* 对；

(3)存在 *Cz* 点到 *Cmp* 对的路径，并且 *Cz* 点先于 *Cmp* 对出现；

(4)环上 *Cmp* 对跳出循环后，都能到达函数 *Rt* 点。

如果函数满足上述 4 个特征，则该函数被识别为比较类函数，否则不是比较类函数。考虑到函数 CFG 中可能有多条循环路径，因此只要其中一条路径满足条件，该函数就被识别为比较类函数。为了支持不同指令集架构，将循环路径和分支路径中的二进制代码转换为中间语言 VEX IR 指令，构建 DFG，后续分析流程不变。

3.3 strcmp 函数识别

strcmp 函数源码和汇编代码如图 5 和图 6 所示，提取图 6 中的循环路径和分支路径，转换为 VEX IR 指令后构建 DFG，循环路径 DFG 如图 8 所示(分支路径较为简单，分别为(*r*2→*r*1，*r*3→*r*1)和(*r*2→*r*1)。图 8 中，*Cz*＝{*t*11}，*Add*＝{*t*10，*t*14}，*Cmp*＝{(*r*3，*r*2)}。循环中存在比较运算 *Cmp* 对

{(r3,r2)}和算术运算 *Add* 点{t10,t14}(特征 *a*),并且还有 *Cz* 点 t11 以及 t11 到 r3 的路径(特征 *b* 和 *c*),最后分支路径中都有从 *Cmp* 对{(r3,r2)}到 *Rt* 点 r0 的路径(特征 *d*),因此 CMPSeek 识别 strcmp 为比较类函数。

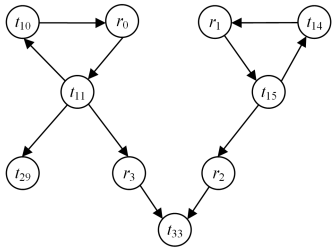


图 8 strcmp 函数 DFG 中的循环路径

Fig. 8 Loop path in DFG of strcmp

4 实验评估

本节从不同的角度评估 CMPSeek 的比较类函数识别效果,并与现有方法 FLIRT,Karonte,SaTC 进行对比,分别对 glibc 库函数和自定义比较类函数进行实验,对比 3 种函数识别方法受编译器选项、编译器版本、优化等级和不同指令集架构的影响,最后比较真实后门中自定义比较类函数的识别效果。

本文中的数据集分为数据集 *A* 和数据集 *B*。数据集 *A* 为 C 语言 glibc 库中的比较类函数,如表 2 所列,共计有 25 个,集中分布在 string.h 和 wchar.h 头文件中,测试时将这两个头文件引入主文件进行静态编译。数据集 *B* 为自定义实现的比较类函数,从 Github 和 Stack Overflow[15] 等平台上收集比较类函数,如表 3 所列,将它们整合到一个 C 程序中进行编译测试。

表 2 glibc 库中的比较类函数

Table 2 Strcmp-like functions in glibc library

Header files	Functions
string.h	*memmem*, *memchr*, *memcmp*, *strcmp*, *strncmp*, *strcasecmp*, *strncasecmp*, *strcoll*, *strstr*, *strpbrk*, *strchr*, *strrchr*, *rawmemchr*, *strspn*, *strcspn*
wchar.h	*wmemcmp*, *wmemchr*, *wcscmp*, *wcsncmp*, *wcschr*, *wcscoll*, *wcscspn*, *wcspbrk*, *wcsrchr*, *wcsstr*

表 3 自定义的比较类函数

Table 3 Self-defined strcmp-like functions

Source Project	Functions
OpenWrt	*BigIntegerCmp*, *BigIntegerCmpInt*, *dpa_bp_cmp*, *BN_cmp*, *BN_ucmp*, *br_vlan_tunid_cmp*
Mirai	*util_strcmp*, *util_memsearch*, *mem_exists*
Others	*ret_strcmp*, *_strcmp*, *test_wrapper_strcmp*, *ft_cmp*, *ret_my_cmp*, *xml_strcmp*, *self_strcmp*, *da_where*, *cmp_mem*, *cmp_string*, *string_diff*, *self_strstr*, *diff*, *str_sub*, *ft_memcmp*, *cmd_setcheck*, *check_str*

本文中的对比方法有 FLIRT,Karonte 和 SaTC(如图 3 和图 4 所示,Karonte 和 SaTC 中针对比较类函数的识别方法完全相同,因此两者选择 SaTC 即可)。考虑到 FLIRT 在制作函数签名时需要源码,而且识别效果受到编译环境的影响,因此 FLIRT 中使用的签名来自 glibc 库函数,编译器类型及版本为 GCC 5.5.0,优化等级为-O0,选择 ARM 架构为目标程序。

本文中的实现工具如下:CMPSeek 是基于 IDAPython 和 pyvex 实现的,并借助于 NetworkX 对 CFG 和 DFG 进行处理。表 4 列出了各种方法实现主要依赖的工具和支持的指令集架构。

表 4 3 种方法的对比

Table 4 Comparison of three methods

Methods	Dependent tools	Supported architectures
FLIRT	IDA Pro	All*
SaTC	angr, pyvex	ARM, MIPS
CMPSeek	IDA Pro, pyvex	ARM, MIPS, PPC, x86/64

注:实验中 FLIRT 使用的函数签名对应 ARM 架构,因此后续实验中将数据集编译为 ARM 架构

4.1 评价指标

使用精准率 *P* 和召回率 *R* 来评估所提方法的效果,精准率和召回率越高,效果就越好。

表 5 列出了实验评估中使用的编译器类型、编译器版本和优化等级,使用 GCC 和 Clang 两个编译器进行测试,并使用 buildroot 构建了不同指令集的交叉工具链,用于生成不同指令架构的程序。

表 5 实验评估中使用的编译环境选项

Table 5 Compile environment options used in experiments

Compiler types	Compiler versions	Optimization levels
GCC	4.5.4, 4.7.4, 4.9.4, 5.1.0, 5.3.0, 5.5.0, 6.3.0, 6.5.0	O0−O3
Clang	3.8.0, 4.0.1, 5.0.1, 6.0.1	O0−O4

4.2 实验结果和分析

4.2.1 编译器选项对函数识别效果的影响

为了比较编译器选项对函数识别效果的影响,使用 GCC 和 Clang 编译器,优化等级为-O0,将数据集 *A* 和 *B* 编译成 ARM 指令集架构的二进制程序,函数识别效果如表 6 和表 7 所列。

表 6 不同编译器下对数据集 *A* 的识别效果对比

Table 6 Comparison of recognition effects of dataset *A* with different compilers

Compiler types	FLIRT		SaTC		CMPSeek	
	P	*R*	*P*	*R*	*P*	*R*
GCC-4.5.4	1.00	0.80	0.67	0.16	**0.62**	**0.80**
GCC-4.7.4	1.00	0.84	0.67	0.16	0.58	0.72
GCC-4.9.4	1.00	0.88	0.50	0.12	0.55	0.76
GCC-5.1.0	**1.00**	**0.88**	0.50	0.12	0.52	0.72
Clang-3.8.1	0.71	0.64	0.67	0.16	0.57	0.76
Clang-4.0.1	0.69	0.56	0.67	0.16	0.57	0.76
Clang-5.0.1	0.71	0.64	0.67	0.16	0.54	0.72
Clang-6.0.1	0.71	0.64	0.67	0.16	0.54	0.72

从表中可知,对于数据集 *A*(glibc 库中的比较类函数),FLIRT 函数的识别效果最好,在 GCC 和 Clang 编译器下 *P* 分别为 1.00 和 0.71,但受编译器选项的影响较大,CMPSeek 和 SaTC 的 *P* 相差不大,但 CMPSeek 的 *R* 能达到 0.80,而 SaTC 的 *R* 只有 0.16;对于数据集 *B*(自定义实现的比较类函数),FLIRT 的 *P* 和 *R* 都为 0.00,完全识别不出比较类函数,CMPSeek 的 *P* 在 0.56 以上,*R* 在 0.67 以上,*P* 和 *R* 都优于

SaTC，整体表现较为稳定。当有函数源码时，FLIRT 函数的识别效果最佳，但受编译器选择的影响较大，在 GCC-5.1.0 中有着最好的识别效果，P 和 R 分别为 1.00 和 0.88，CMPSeek 也有着较为不错的识别效果，在 GCC-4.5.4 中表现最好，P 和 R 分别为 0.62 和 0.80；当没有源码时，CMPSeek 函数的识别效果优于 SaTC 和 FLIRT，且在 GCC 和 Clang 编译器下识别效果基本一致。

表 7 不同编译器下对数据集 B 的识别效果对比

Table 7 Comparison of recognition effects of dataset B with different compilers

Compiler types	FLIRT		SaTC		CMPSeek	
	P	R	P	R	P	R
GCC-4.5.4	0.00	0.00	0.38	0.18	**0.58**	**0.68**
GCC-4.7.4	0.00	0.00	0.38	0.18	0.58	0.68
GCC-4.9.4	0.00	0.00	0.38	0.18	0.58	0.68
GCC-5.1.0	0.00	0.00	0.38	0.18	0.58	0.68
Clang-3.8.1	0.00	0.00	0.35	0.26	0.56	0.67
Clang-4.0.1	0.00	0.00	0.35	0.26	0.56	0.67
Clang-5.0.1	0.00	0.00	0.35	0.26	0.56	0.67
Clang-6.0.1	0.00	0.00	0.35	0.26	0.56	0.67

4.2.2 编译器版本对函数识别效果的影响

为了进一步比较编译器版本对函数识别效果的影响，选择 GCC 编译器，其版本如表 5 所列，优化等级为-O0，将数据集 A 和 B 编译为 ARM 架构的二进制文件，函数识别效果如表 8 和表 9 所列。

表 8 不同版本下对数据集 A 的识别效果对比

Table 8 Comparison of recognition effects of dataset A with different compiler versions

Compiler versions	FLIRT		SaTC		CMPSeek	
	P	R	P	R	P	R
GCC-4.5.4	1.00	0.80	0.67	0.16	**0.62**	**0.80**
GCC-4.7.4	1.00	0.84	0.67	0.16	0.58	0.72
GCC-4.9.4	1.00	0.88	0.50	0.12	0.55	0.76
GCC-5.1.0	1.00	0.88	0.50	0.12	0.52	0.72
GCC-5.3.0	1.00	0.88	0.50	0.12	0.52	0.72
GCC-5.5.0	**1.00**	**1.00**	0.50	0.12	0.52	0.72
GCC-6.3.0	1.00	0.92	0.50	0.12	0.52	0.72
GCC-6.5.0	1.00	0.96	0.50	0.12	0.52	0.72

表 9 不同版本下对数据集 B 的识别效果对比

Table 9 Comparison of recognition effects of dataset B with different compiler versions

Compiler versions	FLIRT		SaTC		CMPSeek	
	P	R	P	R	P	R
GCC-4.5.4	0.00	0.00	0.38	0.18	0.58	0.68
GCC-4.7.4	0.00	0.00	0.38	0.18	0.58	0.68
GCC-4.9.4	0.00	0.00	0.38	0.18	0.58	0.68
GCC-5.1.0	0.00	0.00	0.38	0.18	0.58	0.68
GCC-5.3.0	0.00	0.00	0.38	0.18	**0.70**	**0.71**
GCC-5.5.0	0.00	0.00	0.38	0.18	0.70	0.71
GCC-6.3.0	0.00	0.00	0.38	0.18	0.70	0.71
GCC-6.5.0	0.00	0.00	0.38	0.18	0.70	0.71

从表中可知，对于数据集 A，FLIRT 的识别效果最好，P 为 1.00，R 在 0.80 以上，CMPSeek 次之，P 和 R 分别在 0.52 和 0.72 以上，SaTC 最差，召回率 R 最高只有 0.20。对于数据集 B，CMPSeek 的识别效果最好，P 和 R 分别在 0.58 和 0.68 以上，FLIRT 的 P 和 R 依然为 0.00。3 种方法受编译器版本的影响不大，对于数据集 A，FLIRT 在 GCC-5.5.0 中表现最佳，P 和 R 达到 1.00，因为 FLIRT 中函数签名的编译环境也是 GCC-5.5.0-O0；SaTC 和 CMPSeek 在 GCC-4.5.4 下表现最好；对于数据集 B，3 种方法在 GCC-5.3.0 及以上版本表现最好。

4.2.3 优化等级对函数识别效果的影响

为了比较不同优化等级对函数识别效果的影响，选择 GCC-5.5.0 编译器，优化等级为-O0－O3，将数据集 A 和 B 编译为 ARM 架构的二进制文件，函数识别效果如表 10 和表 11 所列。

表 10 不同优化等级下对数据集 A 的识别效果对比

Table 10 Comparison of recognition effects of dataset A with different optimization levels

Compiler levels	FLIRT		SaTC		CMPSeek	
	P	R	P	R	P	R
-O0	**1.00**	**1.00**	0.50	0.12	0.52	0.72
-O1	0.77	0.84	0.50	0.12	0.52	0.72
-O2	0.73	0.76	0.50	0.12	0.52	0.72
-O3	0.73	0.76	0.50	0.12	**0.57**	**0.80**

表 11 不同优化等级下对数据集 B 的识别效果对比

Table 11 Comparison of recognition effects of dataset B with different optimization levels

Compiler levels	FLIRT		SaTC		CMPSeek	
	P	R	P	R	P	R
-O0	0.00	0.00	0.38	0.18	**0.70**	**0.71**
-O1	0.00	0.00	0.38	0.18	0.70	0.71
-O2	0.00	0.00	0.38	0.18	0.70	0.71
-O3	0.00	0.00	0.38	0.18	0.70	0.71

从表中可以看出，FLIRT 受优化等级影响较大，P 和 R 随优化程度的上升而不断下降，相对而言，CMPSeek 和 SaTC 基本不受优化等级的影响。对数据集 A 来说，即使在-O3 优化等级，综合 P 和 R 来看，依旧是 FLIRT 的效果最好，P 和 R 分别为 0.73 和 0.76；对数据集 B 而言，则是 CMPSeek 的识别效果最优，P 和 R 都在 0.70 以上。无论是数据集 A 还是数据集 B，CMPSeek 的 P 和 R 都高于 SaTC。

4.2.4 指令集对函数识别效果的影响

由于只制作了 FLIRT 的 ARM 架构函数签名，本实验只测试 CMPSeek 对 4 种指令集的识别效果，因此不再分别对数据集 A 和 B 进行测试，将 A 和 B 的源码整合成一个文件，使用 GCC 5.5.0 编译器，优化等级选择-O0－O3，分别编译成 4 种不同架构的二进制文件，实验结果如表 12 所列。

表 12 不同指令集对函数识别效果的影响

Table 12 Effect of different instruction sets on function recognition

Optimization levels	ARM		MIPS		PPC		x86/64	
	P	R	P	R	P	R	P	R
-O0	0.64	0.72	**0.67**	**0.80**	0.65	0.76	0.62	0.72
-O1	0.64	0.72	0.67	0.80	0.65	0.76	0.62	0.72
-O2	0.64	0.72	0.64	0.76	0.65	0.76	0.56	0.68
-O3	0.68	0.77	0.64	0.76	0.65	0.76	0.56	0.68

从表 12 中可以看出，CMPSeek 支持对多指令集架构的函数识别，对不同架构中比较类函数的识别效果相差不大，在

MIPS 架构优化等级为-O0 中表现最好，P 和 R 分别为 0.67 和 0.80。

4.2.5 对后门中比较类函数的识别效果

为了比较 3 种方法对实际后门中的函数识别效果，根据披露的后门信息，收集相关的固件，其中 CVE-2015-7755 和 CVE-2016-1909 对应的 Juniper 的路由器固件和 Fortinet 的防火墙固件都是剥离的二进制程序。分别用 3 种方法对固件的比较类函数进行识别，其中 FLIRT 使用的函数签名来源于 glibc 库，结果如表 13 所列。Juniper 和 Fortinet 的网络设备中都使用自主研发的操作系统 ScreenOS 和 FortiOS，而非 Linux 内核。SaTC 中查找的比较指令为 cmpne/cmpeq，但是固件的架构为 x86/64 指令集，属于复杂指令集，使用独立的比较指令 cmp 和控制跳转指令 jnz/jle，因此 FLIRT 和 SaTC 都无法识别后门中的比较类函数。实验表明，CMPSeek 能够识别真实后门中的比较类函数，对嵌入式设备口令后门的检测与发现具有重大意义和价值。

表 13 对后门中比较类函数的识别效果

Table 13 Identify effect of strcmp-like function in backdoor

Firmware	FLIRT	SaTC	CMPSeek
ssg500.6.2.0r15.0.bin	√	√	√
ssg500.6.3.0r12b.0.bin	√	√	√
FGT_300C-v400.out	√	√	√
FGT_300D-v433.out	√	√	√

结束语 比较类函数识别方法 CMPSeek 基于数据流特征匹配的方式，识别二进制程序中的比较类函数。CMPSeek 分析函数的控制流，从中提取循环路径和分支路径，构建数据流并检测数据流特征与比较类函数是否匹配。将路径中的指令转换为中间语言 VEX IR 指令，支持多种指令集。不依赖源码等信息，并且受编译器选项、编译器版本以及优化等级的影响较小。无论是 glibc 库中的比较类函数，还是自定义实现的比较类函数，都有很好的识别效果，对嵌入式设备中口令后门的检测具有重大价值。相比最新的工作 Karonte 和 SaTC，CMPSeek 方法提高了比较类函数识别的精准率和召回率，具有更好的适用性。实验结果表明，在有源码的情况下，CMPSeek 函数的识别效果虽然不如 FLIRT，但也有着不错的表现；当缺少源码、函数名等信息时，相比 FLIRT 和 SaTC，CMPSeek 在精准率和召回率上都有着更好的结果。

虽然基于静态分析的 CMPSeek 具有快速高效的特点，但是由于静态数据流分析很难做到十分精确，因此难免存在误报和漏报的情况，从而影响函数识别的准确率。在后续的研究中，将尝试结合动态分析的方式来提高函数识别的准确率。

参 考 文 献

[1] PIERLUIGI P. Netgear, Linksys and many other Wireless Routers have a backdoor [EB/OL]. (2014-01-04). http://securityaffairs.co/wordpress/20941/hacking/netgear-linkys-routers-back-door.html.

[2] MATHIEU S. Reverse Engineering a D-Link Backdoor [EB/OL]. (2013-10-14). https://hackaday.com/2013/10/14/reverse-engineering-a-d-link-backdoor/.

[3] OPERATOR8203. SSH Backdoor for FortiGate OS Version 4.x up to 5.0.7[EB/OL]. (2016-01-26). https://seclists.org/fulldisclosure/2016/Jan/26.

[4] ZYXEL N. Zyxel security advisory for hardcoded credential vulnerability| Zyxel[EB/OL]. (2021-01-08). https://www.zyxel.com/support/CVE-2020-29583.shtml.

[5] HD M. CVE-2015-7755: Juniper ScreenOS Authentication Backdoor[EB/OL]. (2015-12-20). https://www.rapid7.com/blog/post/2015/12/20/cve-2015-7755-juniper-screenos-authentication-backdoor/.

[6] IDA Pro. Hex-ray. IDA F. L. I. R. T. Technology: In-Depth[EB/OL]. (2021-07-30). https://hex-rays.com/products/ida/tech/flirt/in_depth/.

[7] REDINI N, MACHIRY A, WANG R, et al. Karonte: Detecting insecure multi-binary interactions in embedded firmware[C]// Proceedings of 2020 IEEE Symposium on Security and Privacy (SP). Piscataway, NJ: IEEE, 2020: 1544-1561.

[8] CHEN L, WANG Y H, CAI Q P, et al. Sharing More and Checking Less: Leveraging Common Input Keywords to Detect Bugs in Embedded Systems[C]// Proceedings of 30th USENIX Security Symposium (USENIX Security 21). Berkeley, CA: USENIX Association, 2021: 303-319.

[9] SHOSHITAISHVILI Y, WANG R, HAUSER C, et al. Firmalice-Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware[C]// NDSS. 2015: 1.1-8.1.

[10] THOMAS S L, CHOTHIA T, GARCIA F D. Stringer: measuring the importance of static data comparisons to detect backdoors and undocumented functionality[C]// European Symposium on Research in Computer Security. Cham: Springer, 2017: 513-531.

[11] YIN X K, LU B, CAN R J, et al. Memcpy-Like Function Identification Technique with Control Flow and Data Flow Analysis [J/OL]. Journal of Computer Research and Developmen. [2022-06-14]. http://kns.cnki.net/kcms/detail/11.1777.TP.20220304.1646.003.html.

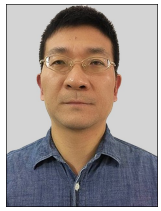
[12] ANGR. Python bindings for Valgrind's VEX IR [EB/OL]. (2021-05-23). https://github.com/angr/pyvex/.

[13] IDA Pro. Hex-ray. IDAPython documentation [EB/OL]. (2021-07-30). https://www.hex-rays.com/wp-content/static/products/ida/support/idapython_docs/.

[14] NETWORKX developers. Network analysis in python [EB/OL]. https://networkx.org/.

[15] STACK O. Where Developers Learn, Share & Build Careers [EB/OL]. https://stackoverflow.com/.

HU An-xiang, born in 1996, master. His main research interests include cyber security and embedded network device.



LIU Sheng-li, born in 1973, Ph.D, professor. His main research interests include network device security and network attack detection.

（责任编辑：喻藜）