软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

函数级数据依赖图及其在静态脆弱性分析中的应用*

陈 千[1,2,3], 程 凯[1,2], 郑尧文[1,2], 朱红松[1,2], 孙利民[1,2]

[1](中国科学院大学网络空间安全学院 北京 100049)

[2](中国科学院信息工程研究所物联网信息安全技术北京市重点实验室 北京 100093)

[3](北京奇虎科技有限公司 北京 100015)

通讯作者: 朱红松, E-mail: zhuhongsong@iie.ac.cn

摘 要: 数据流分析是二进制程序分析的重要手段, 但传统数据依赖图(DDG)构建的时间与空间复杂度高, 限制了可分析代码的规模.本文提出了函数级数据依赖图(FDDG)的概念, 并设计了函数级数据依赖图的构建方法.在考虑函数参数及参数间相互依赖关系的基础上, 将函数作为整体分析, 忽略函数内部的具体实现, 显著缩小了数据依赖图规模, 降低了数据依赖图生成的时空复杂度. 实验结果表明, 与开源工具 angr 中的 DDG 生成方法相比, FDDG 的生成时间性能普遍提升了 3 个数量级.同时, 将 FDDG 应用于嵌入式二进制固件脆弱性分析, 实现了嵌入式固件脆弱性分析原型系统 FFVA, 在对 D-Link、NETGEAR、EasyN、uniview 等品牌的设备固件分析中, 发现了 24 个漏洞, 其中 14 个属于未知漏洞, 进一步验证了 FDDG 在静态脆弱性分析中的有效性.

关键词: 数据流分析;函数级数据依赖图;脆弱性分析;固件

中图法分类号: TP311

中文引用格式: 陈千,程凯,郑尧文,朱红松,孙利民.函数级数据依赖图及其在静态脆弱性分析中的应用.软件学报. http://www.jos.org.cn/1000-9825/ 0000.htm

英文引用格式: Chen Q, Cheng K, Zheng YW, Zhu HS, Sun LM. Function-level Data Dependency Graph and its Application in Static Vulnerability Analysis. Ruan Jian Xue Bao/Journal of Software, 2018 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

Function-level Data Dependency Graph and its Application in Static Vulnerability Analysis

CHEN Qian[1,2,3], CHENG Kai[1,2], ZHENG Yao-Wen[1,2], ZHU Hong-Song[1,2], SUN Li-Min[1,2]

[1](School of Cyber Security, University of Chinese Academy of Science, Beijing 100049, China)

[2](Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, Chinese Academy of Science, Beijing 100093, China)

[3](Qihoo 360 Technology Co. Ltd., Beijing 100015, China)

Abstract: Data flow analysis plays an important role in binary code analysis. Due to the high time and space complexity when constructing the traditional data dependency graph (DDG), the size of the binary code analyzed is limited. This paper has introduced a novel graph model, function-level data dependency graph (FDDG), and proposed the corresponding construction method. While considering the function parameters and the relationships between them, the function is regarded as a whole, thus ignoring the details inside the function. As a result, the size of the data dependency graph is reduced significantly. Also, the time and space are saved greatly. According to the experimental results, the time performance of the method is enhanced by about three orders of magnitude compared to the method in angr. As an instance, FDDG is used to analyze the vulnerability of embedded firmware, and a firmware vulnerability

* 基金项目: 国家自然科学基金(U1766215, U1636120); 中国科学院信息工程研究所国际合作项目(Y7Z0451104)

Foundation item: National Natural Science Foundation of China (U1766215, U1636120); International Cooperation Project of Institute of Information Engineering, Chinese Academy of Science (Y7Z0451104)

收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00

CNKI 在线出版时间: 0000-00-00

analysis prototype system called FFVA is implemented. By analyzing firmware from real embedded devices with the FFVA system, a total of 24 vulnerabilities have been found in the devices from D-Link, NETGEAR, EasyN, uniview and so on, of which 14 are unknown vulnerabilities, thus validating the effectiveness of function-level data dependency graph in static vulnerability analysis.

Key words: data flow analysis; function-level data dependency graph; vulnerability analysis; firmware

数据流分析是一种用来获取相关数据沿着程序执行路径流动的信息分析技术[1]，分析对象是程序执行路径上的数据流动或可能的取值，最初被广泛应用于程序的编译优化过程.作为程序静态分析的辅助支撑技术，该技术也逐渐被运用于程序切片、变量别名分析[2,3,4]、隐私泄露检测[5,6,7]等中.

由于程序数据流的某些特点或性质与程序漏洞紧密相关，也可以将数据流分析技术用于程序漏洞的检测中[8].比如对 SQL 注入、系统命令注入等漏洞进行检测，主要关心数据的流动或数据的性质，即需要知道某个变量的取值是否来源于某个非可信的数据源.文献[9]采用上下文敏感的数据流分析方式来发现 Web 应用中的脆弱点，适用于 SQL 注入、XSS 和命令注入等污点类漏洞的检测.而针对缓冲区溢出漏洞的检测，还需要知道某个程序变量可能的取值范围.目前，数据流分析已被广泛应用于各种面向源代码的漏洞检测分析工具中，典型的工具包括 FindBugs[10]、Fortify SCA[11]、Coverity[12]、IBM Appscan Source Edition[13]和 Pinpoint[14]等.类似的，数据流分析技术也可以用于二进制代码的漏洞检测与分析中，如 CodeSonar[15].

在进行数据流分析时，由于构建传统数据依赖图(Data Dependency Graph,简称 DDG)的时间与空间复杂度高，严重限制了可分析代码的规模，如何对数据依赖图进行改进，对于提升静态分析效率具有重要意义.本文提出了函数级数据依赖图(Function-level Data Dependency Graph,简称 FDDG)的概念，设计了 FDDG 的构建方法，在保证数据流准确性的同时，缩小数据依赖图的规模，显著降低数据依赖图的构建时间.实验结果表明，基于 FDDG 对嵌入式固件进行脆弱性分析，在保证分析能力的情况下，能够极大地提升分析效率.

1 函数级数据依赖图

1.1 数据流分析

数据流分析主要关注程序执行路径上的数据流流动或可能的取值，其目的是在程序的一定范围内，确定变量定义和引用间的关系.变量定义指程序中的某条语句对变量进行赋值，除定义之外的其他变量出现称为变量引用.

数据依赖表示程序中对某变量进行引用的语句(或基本块)对定义该变量语句的依赖，即一种“定义-引用”依赖关系.若 a 和 b 分别是程序控制流图中的两个节点，v 为程序中的一个变量，当节点 a 和 b 满足以下条件时，称节点 b 关于变量 v 直接数据依赖于节点 a：(1)节点 a 对变量 v 进行定义；(2)节点 b 中引用了变量 v；(3)节点 a 到节点 b 之间存在一条可执行路径，且在此路径上不存在其他语句对变量 v 进行定义.

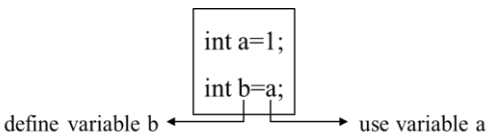


Fig. 1 Example of variable definition and use

图 1 变量定义与引用示例

数据依赖图 DDG[16]是一种表示程序语句(或基本块)间数据依赖关系的有向图.根据数据依赖关系可以构建数据依赖图 DDG=(V, E)，其中 V 表示程序中所有语句对应节点的集合，E 表示语句之间数据依赖关系的集合.图 2为代码片段及其对应的数据依赖图示例.对于二进制代码而言，数据依赖图中的节点均为指令，因此本文将这种传统的数据依赖图称为指令级数据依赖图.

1.2 函数级数据依赖图定义

指令级数据依赖图在数据流分析中应用广泛，但其资源开销限制了可分析代码的规模和分析的效率.鉴于此，本文提出函数级数据依赖图 FDDG=(FV, FE)的概念，通过降低构图粒度提升分析效能.与指令级数据依赖图不同的是，FV 表示程序中函数对应节点的集合，FE 表示函数之间参数依赖关系的集合.

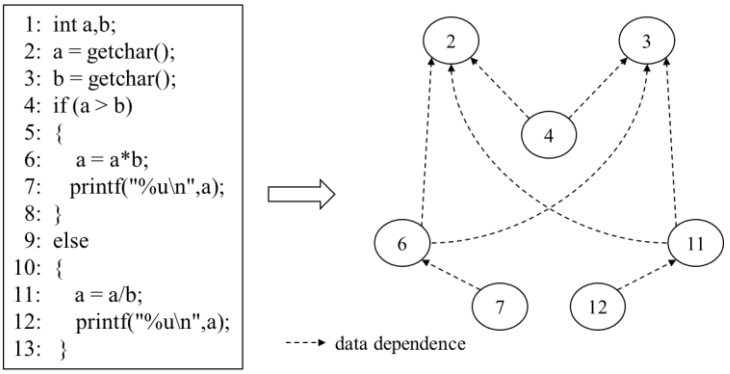


Fig. 2 Example of code snippet and its data dependency graph

图 2 代码片段及其数据依赖图示例

1.2.1 函数级数据依赖图中的节点

在函数级数据依赖图中，节点代表函数，其记录了函数参数或返回值之间的“定义-引用”关系，同时包含函数参数的大小、函数名等信息.节点可以采用(func_name,<[def_arg1,def_arg2,…],[use_arg1,use_arg2,…]>)这种形式来表示，其中，func_name 为函数名称，<[def_arg1,def_arg2,…],[use_arg1,use_arg2,…]>表示函数参数或返回值之间的“定义-引用”关系，[def_arg1,def_arg2,…]为定义参数的集合，[use_arg1,use_arg2,…]为引用参数的集合.

对于常见的函数，均可以采用上述形式进行表示.根据函数功能的不同，可以分为如下几类：

(1) retval=strstr(arg1,arg2,…)类函数：这类函数实质上是对返回值 retval 进行定义，而对 retval 的定义又引用了函数参数 arg1,arg2,…，由于返回值 retval 可能会在后续的数据流中用到，因此可以表示为(strstr,<[retval],[arg1,arg2,…]>)，其中[retval]为定义参数的集合，[arg1,arg2,…]为引用参数的集合；

(2) memcpy(arg1,arg2,arg3,…)类函数：这类函数实质上是对某个参数如 arg1 进行定义，而对该参数的定义又引用了其他参数如 arg2,arg3,…，同时这类函数没有返回值或返回值不会体现在后续的数据流中，因此可以表示为(memcpy,<[arg1],[arg2,arg3,…]>)，其中[arg1]为定义参数的集合，[arg2,arg3,…]为引用参数的集合；

(3) retval=strcpy(arg1,arg2,…)类函数：这类函数实质上是对某个参数如 arg1 进行定义，而对该参数的定义又引用了其他参数如 arg2,…，同时也对返回值 retval 进行了定义，而且返回值可能会体现在后续的数据流中，因此可以表示为(strcpy,<[arg1,retval],[arg2,…]>)，其中[arg1,retval]为定义参数的集合，[arg2,…]为引用参数的集合；

(4) system(arg1,…)类函数：这类函数只是引用函数参数如 arg1,…，同时该函数没有返回值或返回值不会体现在后续的数据流中，因此可以表示为(system,<[],[arg1,…]>)，其中[]为定义参数的集合，[arg1,…]为引用参数的集合.

除了程序中的函数外，也可以将程序中的某些语句片段(或某条语句)抽象成函数，进而表示为 FDDG 中的节点.最典型的就是实现循环拷贝的语句片段，如图 3所示，其中将实现循环拷贝的代码片段抽象成 loop_copy 函数，同时保存循环间隔和循环终止条件等信息.这种处理方法拓展了函数级数据依赖图的表达能力.

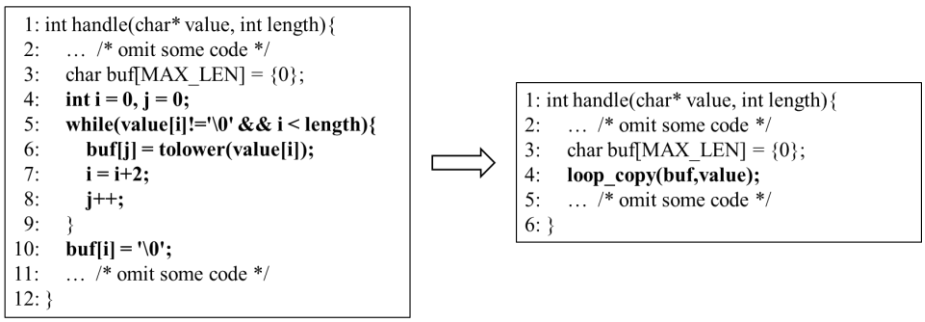


Fig. 3 Example of abstracting a code snippet into a function

图 3 代码片段抽象为函数示例

1.2.2 函数级数据依赖图中的边

函数级数据依赖图中的边表示函数之间参数的依赖关系，主要包括“引用”和“消灭”两种关系.若 a 和 b 分别为程序 P 中的两个函数，v 为两个函数共同的参数，这里将函数的返回值也看作函数的参数，当 a 和 b 满足以下条件时，称函数 b 和函数 a 之间关于参数 v 存在“引用”关系：

(1) 函数 a 对参数 v 进行定义；

(2) 函数 b 引用参数 v；

(3) 函数 a 到函数 b 之间存在一条可执行路径，且此路径上不存在其他对参数 v 的定义.

类似地，当 a 和 b 满足下列条件时，称函数 b 和函数 a 之间关于参数 v 存在“消灭”关系：

(1) 函数 a 对参数 v 进行定义；

(2) 函数 b 对参数 v 进行重新定义；

(3) 函数 a 到函数 b 之间存在一条可执行路径，且此路径上不存在其他对参数 v 的定义.

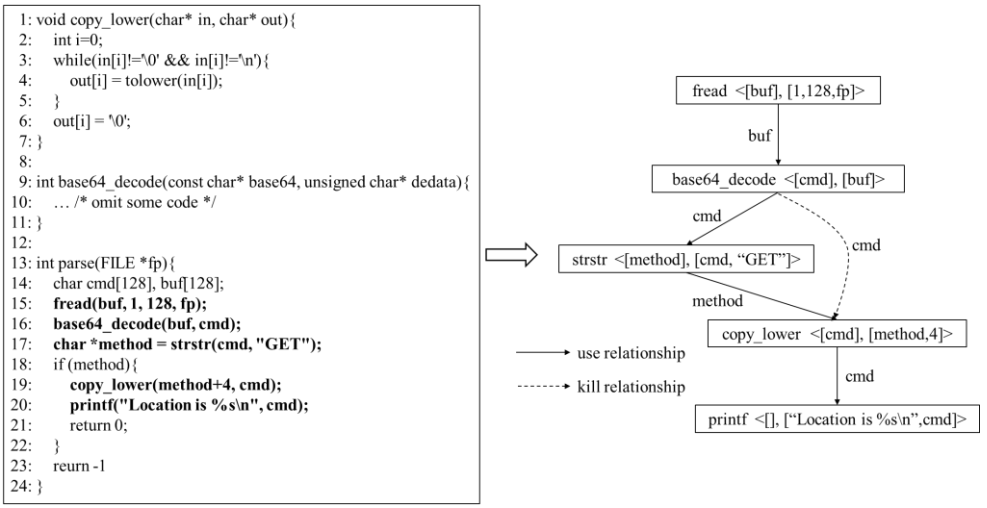


Fig. 4 Example of code snippet and its function-level data dependency graph

图 4 代码片段及其函数级数据依赖图示例

图 4所示为代码片段及其对应的函数级数据依赖图示例.其中，fread 函数对参数 buf 进行定义，base64_decode 函数引用了参数 buf，同时两个函数之间存在一条可执行路径，因此 base64_decode 函数和 fread

函数之间关于参数 buf 存在“引用关系”.同理，strstr 函数和 base64_decode 函数、printf 函数和 copy_lower 函数之间关于参数 cmd 均存在“引用”关系.另外，copy_lower 函数对参数 cmd 进行重新定义，而 base64_decode 函数和 copy_lower 函数之间存在一条可执行路径，且路径上没有对参数 cmd 的其他定义，因此 copy_lower 函数和 base64_decode 函数之间关于参数 cmd 存在“消灭”关系.

1.3 函数级数据依赖图特性分析

指令级数据依赖图 DDG 以语句为节点,在构建过程中会考虑所有的变量,即在所有相关变量之间建立“定义-引用”依赖关系，关系精确、完整，适用于程序切片、程序数据结构恢复等应用，但在程序静态脆弱性分析方面却不高效.一方面，DDG 将变量与变量所处的语境(如变量为 strcpy()函数的参数)分开，变量的上下文信息变得不明确；另一方面，考虑所有变量会造成 DDG 的规模极大，而很多变量信息对脆弱性分析没有帮助，导致后续的数据依赖分析效率低下，浪费计算时间和空间.

函数级数据依赖图 FDDG 以函数为节点，在构建过程中仅考虑函数之间参数的相互依赖关系，得到的依赖图粒度较“粗”，难以用于程序切片.但 FDDG 将变量与其所处的语境保存在一起，同时保留了变量路径分析所需要的全部信息，适用于程序静态脆弱性分析，如对缓冲区溢出和命令注入等漏洞进行检测.此外，由于 FDDG 的规模比较小，数据流比较清晰，因此对参数进行回溯效率很高.

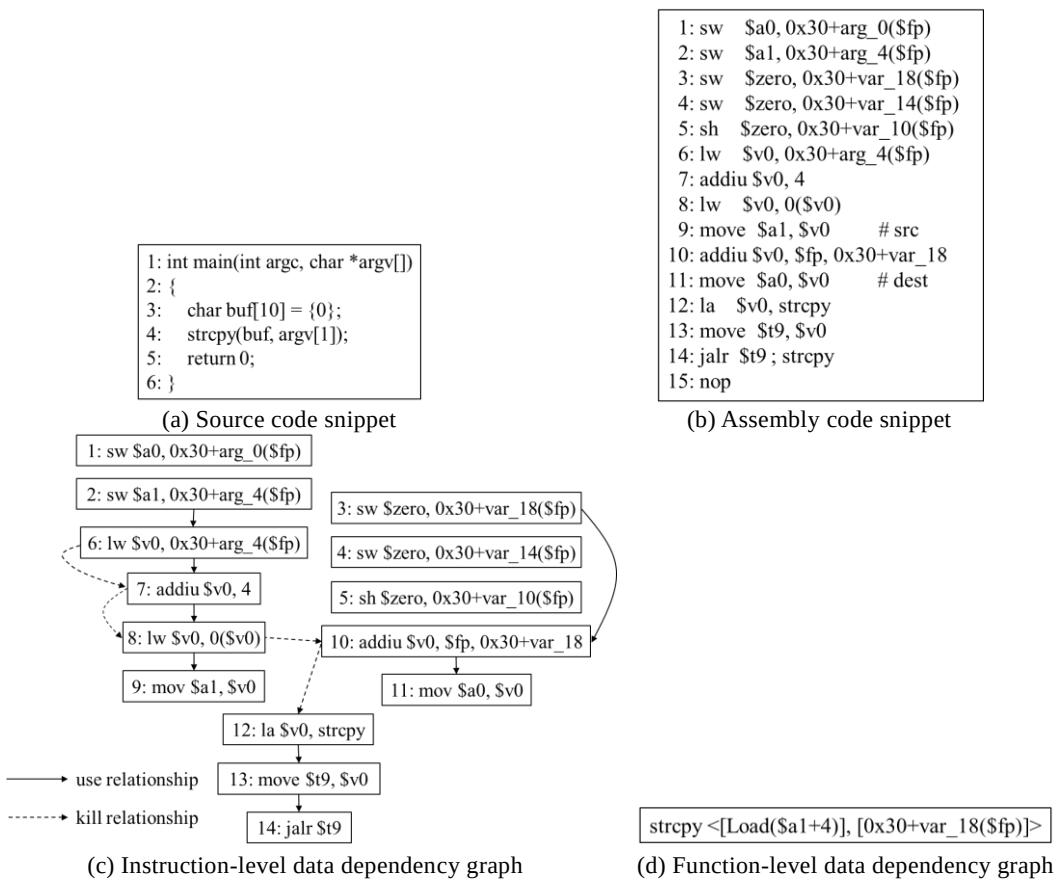


Fig. 5 Example of the two kinds of data dependency graph

图 5 两种数据依赖图示例

图 5所示为针对同一段汇编代码构建的两种数据依赖图示例.由图 5 可知，指令级数据依赖图 DDG 的规

模明显大于函数级数据依赖图 FDDG.同时，在 DDG 中，函数与函数参数是分离的；而 FDDG 则直接将函数参数和函数保存在一起.此外，针对图 5-(a)中的 strcpy(buf, argv[1])语句，在进行静态脆弱性分析时，需要对参数 buf 和 argv[1]进行溯源，以获取参数指向的内存空间大小、参数来源等信息.基于图 5-(c)所示的 DDG 进行回溯，查找路径包括“11 -> 10”和“9 -> 8 -> 7 ->6 ->2”；而在图 5-(d)所示的 FDDG 中，函数参数中已经包含了所需的信息，不用进行回溯.

当然，函数级数据依赖图的应用不局限于脆弱性分析，在很多软件代码宏观特征分析如软件代码相似性检测、恶意代码家族判定中都有广泛的应用前景.

2 函数级数据依赖图构建

为保证数据流分析的准确率，通常需要进行流敏感的分析，即考虑程序语句可能的执行顺序，因而构建 FDDG 需要控制流分析的支持.

控制流分析基于程序的控制流图(Control Flow Graph,简称 CFG)，用于确定一个程序的控制结构，可用于探究程序可能的执行路径、提取循环分析结构等.在构建程序控制流图的过程中，记录函数参数之间的“定义-引用”关系以及函数名、参数大小等信息，并将这些信息保存在对应的控制流图节点中.

在得到函数调用图和程序控制流图后，构建函数级数据依赖图 FDDG 的方法如下述算法 1 表述，具体流程如下：

(1) 按后序方式对函数调用图进行遍历；
(2) 针对函数调用图中的每个函数，对其控制流图进行层次遍历；
(3) 针对单个函数控制流图中的节点，获取节点中保存的函数参数关系记录，根据函数之间参数的依赖关系，先后在当前函数和全局变量定义列表中查找函数参数定义中引用参数的定义，若查找成功则将对参数进行定义的函数和对其引用参数进行定义的函数作为节点、参数之间的依赖关系作为边添加到数据依赖图中，然后跳转到步骤(5)；
(4) 将对该参数进行定义的函数作为节点添加到数据依赖图中.若在当前函数和全局变量定义列表中均无法找到该定义中引用参数的定义，则将该参数的定义更新到所有直接调用当前函数的父函数中；
(5) 若当前函数的控制流图中某个节点的出度为 0，则需要将函数内对函数参数的定义更新到所有直接调用该函数的父函数中，同时将对全局变量的定义更新到全局变量定义列表中；
(6) 若函数控制流图遍历完毕则跳转到步骤(7)，否则跳转到步骤(2)；
(7) 若函数调用图遍历完毕则停止，否则跳转到步骤(1).

算法 1. 函数级数据依赖图构建算法.

输入：函数调用图 CG，控制流图 CFG

输出：函数级数据依赖图 FDDG

```
① GenerateFDDG(CG,CFG)；
② for each node ni in CG
③  for each block_node bi in CFGni
④    for each dst_def_info ddi in bi
⑤      sd ← search_src_def(ddi)；/*在局部或全局变量定义列表中查找引用参数的定义*/
⑥      if sd
⑦        FDDG.add_edge(sd, ddi)；
⑧      else
⑨        FDDG.add_node(ddi)；
⑩        save_forward_def_info(ni, ddi)；/*保存参数用于后续在父节点中继续查找*/
⑪      end if
```

```
⑫        save_def_info(ni, ddi);
⑬      end for
⑭      for each node si in successors(bi)
⑮        add_def_info(si); /*将 bi 中的信息添加到后继节点中*/
⑯      end for
⑰      if out_degreebi == 0
⑱        for each node pi in predecessors(ni)
⑲          update_def_infos(pi); /*将 ni 中的信息更新到父节点中*/
⑳        end for
㉑      end if
㉒    end for
㉓    for each node pi in predecessors(ni)
㉔      search_forward_def(pi); /*对于未查找到的定义，在父节点中进行查找*/
㉕    end for
⑳end for
```

其中，在步骤(3)中，针对某个关键参数的定义，如果该定义中引用参数的类型为整数类型，则不查找引用参数的定义，只有当其类型为堆/栈变量、返回值、函数参数或全局变量时才进行查找.此外，在全局变量定义列表中查找其引用参数的定义时，如果存在多个对引用参数的定义，则在对关键参数进行定义的函数和每个对其引用参数进行定义的函数之间都建立一条边.在步骤(5)，由于函数内部对函数参数的定义会影响父函数的行为，当遇到出度为 0 的节点时，需要更新函数参数定义，同时也要更新全局变量定义列表.

3 函数级数据依赖图在静态脆弱性分析中的应用

函数级数据依赖图 FDDG 保存了函数参数及函数之间参数的依赖关系，所以最直观的应用是分析函数参数在程序调用过程中的传递路径.本节将以固件中二进制程序的静态脆弱性分析为例分析 FDDG 的可用性.

3.1 二进制程序静态脆弱性分析

二进制程序静态脆弱性分析的基本流程如图 6所示.其中，通过控制流分析可以得到程序的控制流图 CFG 和函数调用图(Call Graph,简称 CG)，通过数据流分析可以得到数据依赖图 DDG.在此基础上，根据漏洞模式或规则对程序中的潜在脆弱点进行分析，以判断程序是否存在安全缺陷.

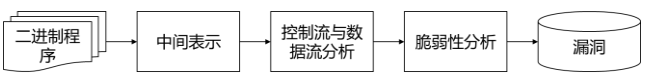


Fig. 6 Flow of static vulnerability analysis

图 6 静态脆弱性分析流程

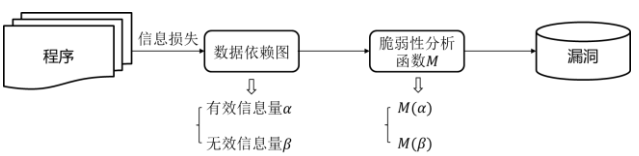


Fig. 7 A vulnerability mining model based on data dependency graph

图 7 基于数据依赖图的漏洞挖掘模型

在进行静态代码分析时，由于存在函数间接调用、指针别名等机制，构建 DDG 的过程中会出现信息损失.基于 DDG 进行静态脆弱性分析，实质是对 DDG 中包含的信息进行挖掘，从而判断是否存在漏洞，整个

过程可以用图 7所示的模型表示.为了便于描述，将程序中包含的与脆弱点相关的信息称为有效信息，与脆弱点无关的信息称为无效信息.

由图 7 可知，脆弱性分析的结果可以看作是将脆弱性分析函数 M 作用于有效信息量 α 和无效信息量 β 上的输出.通常，当包含的有效信息越多时，发现漏洞的可能性越大；当包含的无效信息越多时，检测到不是漏洞的可能性也越大.因此可以将 M(α)与有效信息量 α、M(β)与无效信息量 β 之间的关系简化为线性关系：$M(\alpha)= k_1 \alpha$，$M(\beta)= k_2 \beta$.

在理想情况下，脆弱性分析函数 M 可以根据有效信息检测出所有的漏洞.但实际上，通常会存在根据有效信息检测出不是漏洞的情形，和根据无效信息检测出漏洞的情形，即所谓的漏报和误报.因此可以对 M(α)、M(β)进一步细分，如表 1所示.

Table 1 The relationship between the results of vulnerability analysis and information quantity

表 1 脆弱性分析结果与信息的对应关系

M(α, β)	*有效信息*	*无效信息*
判断是漏洞	$k_{11}\alpha$	$k_{21}\beta$
判断不是漏洞	$k_{12}\alpha$	$k_{22}\beta$

对于特定的脆弱性分析函数而言，其对信息进行分析的能力是恒定的.也就是说，系数 k_{11}、k_{12}、k_{21}、k_{22} 只与脆弱性分析函数有关，而与信息量无关.在表 1中，在进行脆弱性分析时，通常希望 $k_{11}\alpha$ 越大越好，$k_{12}\alpha$ 和 $k_{21}\beta$ 越小越好.由于 k_{11}、k_{12} 和 k_{22} 是恒定的，所以漏报和误报实际上只和数据依赖图 DDG 中包含的有效信息量和无效信息量有关.

3.2 固件脆弱性静态分析系统

为验证 FDDG 在程序脆弱性静态分析中的效能，本文在开源框架 angr[17,18]的基础上，结合污点分析、符号执行等技术，实现了一个固件脆弱性静态分析原型系统 FFVA.

固件通常是对嵌入式设备软件系统的一种称谓，其由固件头、启动引导程序、操作系统内核、根文件系统以及附加数据等组成.与传统软件类似，固件中也存在安全缺陷或漏洞.由于很难获取固件源码，因此嵌入式设备脆弱性分析的重点研究对象是固件文件系统中的二进制应用程序.

在固件二进制程序中，常见的程序漏洞包括缓冲区溢出、格式化字符串漏洞、命令注入、认证缺失、硬编码/弱密钥等.通常，缓冲区溢出、格式化字符串漏洞和命令注入这三类漏洞的形成原因主要是与不安全函数调用有关，可以尝试基于 FDDG 对这三类漏洞进行检测.以 strcpy 函数为例，如果源缓冲区的大小超过目的缓冲区，且源缓冲区的内容为外部可控，将导致缓冲区溢出漏洞，这里源缓冲区的大小及内容来源是评判是否存在漏洞的关键，适用于 FDDG 分析.

Table 2 Top 10 best practices for embedded application security provided by OWASP

表 2 OWASP 嵌入式应用安全 Top10 实践清单

Rank	*Name*
1	Buffer and Stack Overflow Protection
2	Injection Prevention
3	Firmware Updates and Cryptographic Signatures
4	Securing Sensitive Information
5	Identity Management
6	Embedded Framework and C-Based Hardening
7	Usage of Debug Code and Interfaces
8	Transport Layer Security
9	Data collection Usage and Storage –Privacy
10	Third Party Code and Components

表 2所示为 OWASP 嵌入式应用安全项目[19]提供的 Top10 清单.由表 2可知，缓冲区溢出和命令注入漏洞对嵌入式设备的威胁较大.对这两类漏洞进行检测，FFVA 系统的主要思想如下：将这两类漏洞视为一种特殊的“Source-Sink”问题[20,21]，分析程序中所有 Source 点到 Sink 点之间的数据流，然后对 Sink 点的安全性进行判断，同时进行路径敏感性分析.其中 Source 点表示程序中的外部数据输入点如网络流读取、文件读取和用户输入等，Sink 点表示程序中可能会存在安全风险的代码片段，如对 strcpy()等不安全函数的调用.

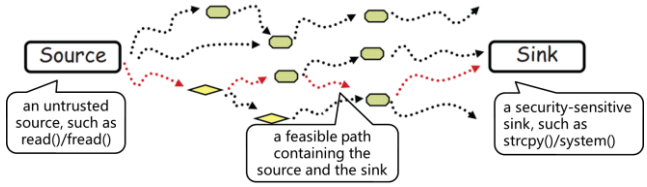


Fig. 8 Example of “Source-Sink” problem

图 8 “Source-Sink”问题示意图

4 实验与结果

利用 FFVA 系统，本节通过两个实验来评估函数级数据依赖图 FDDG 构建方法的效率和在固件静态脆弱性分析中的有效性：

- FDDG 构建时间性能评估实验：对不同固件中的二进制程序分别构建 FDDG 和 DDG，对比两种依赖图的构建时间；
- FDDG 在固件静态脆弱性分析中的有效性评估实验：运用 FFVA 系统对二进制程序进行脆弱性分析，尝试发现程序中存在的安全缺陷或漏洞.

实验采用的软硬件环境如表 3所示.

Tabel 3 Experimental environment

表 3 实验软硬件环境

Type	*Specification*
CPU	Intel(R) Xeon(R) CPU E5-2687W v3 @ 3.10Ghz, 10 Cores
Num. of CPU	4
Memory Size	126G
OS	Ubuntu 16.04.3 LTS x86_64
Software Module	Python 2.7.12 angr 7.7.12.16

对固件进行脆弱性分析，主要分析对象是固件中提供网络服务的程序模块，如 HTTP、UPnP 等.因为一旦在这些程序中发现漏洞，其很可能会被远程利用，进而带来严重的安全隐患.考虑到不同指令架构、不同品牌、文件大小以及是否有真实可用设备等因素，本文从路由器和摄像头设备固件中最终选取表 4中给出的 13 个二进制程序样本作为分析对象.

4.1 FDDG构建时间性能评估

本文采用 FFVA 系统构建函数级数据依赖图，然后利用 angr 框架构建指令级数据依赖图，对两种依赖图的构建时间进行对比.为了保证两种方案分析函数的一致性，本文采用程序的入口地址作为分析的起始地址，将从入口地址可达的所有非系统函数作为待分析函数列表.

Table 4 Details of binary samples

表 4 二进制程序样本相关信息

Name	*Size(k)*	*Arch*	*Brand*	*Model*
wlancmd	38	MIPS	Huawei	HG532e
stupid-ftpd	53	MIPS	D-Link	DIR-882_A1
upnp	90	MIPS	Huawei	HG532e
jjhttpd	118	MIPS	D-Link	DIR820LB1
lighttpd	145	ARM	Honeywell	HIVDC-F100V
cgibin	150	ARM	D-Link	DIR-890L
cgibin	155	MIPS	D-Link	DIR-645
miniupnpd	165	MIPS	D-Link	DIR-882_A1
upnpd	213	MIPS	NETGEAR	DGN2200
setup.cgi	324	MIPS	NETGEAR	DGN1000
httpd	994	MIPS	NETGEAR	DGN2200
ipc_server	2048	ARM	EasyN	B18EN
mwareserver	4812	ARM	uniview	IPC_6201

由于控制流分析采用并行化加速，CFG 的构建时间在整个脆弱性分析中的占比很小，因此数据依赖图的构建时间将直接影响脆弱性分析的效率.由表 5可知，与 DDG 相比，FDDG 的构建时间基本在 1 分钟内，除 wlancmd 这种分析函数个数少、内部逻辑简单的程序样本外，构建时间性能普遍提升约 3 个数量级.同时，构建时间与分析函数的数量、样本内部的流程复杂度密切相关.以两种硬件架构(ARM 和 MIPS)的 cgibin 样本为例，对 ARM 架构的 cgibin 分析了 356 个函数，多于 MIPS 架构，FDDG 生成时间也长一些，但 FDDG 的构建时间差异(3 倍)小于 DDG 的构建时间差异(11 倍).

Table 5 Comparison of the data dependency graph scale

表 5 数据依赖图构建时间对比

Name	*Size(k)*	*Model*	*Number of functions analyzed*	*Time consumed(min)*			*Time performance improved(time)*
				CFG	*DDG*	*FDDG*	
wlancmd	38	HG532e	7	0.07	0.03	0.002	**15**
stupid-ftpd	53	DIR-882_A1	24	0.16	6.99	0.005	**1500**
upnp	90	HG532e	56	0.43	113.54	0.038	**3000**
jjhttpd	118	DIR820LB1	45	0.47	120.14	0.027	**4400**
lighttpd	145	HIVDC-F100V	136	0.98	468.02	0.091	**5100**
cgibin	150	DIR-890L	356	1.24	600.72	0.115	**5600**
cgibin	155	DIR-645	88	0.90	53.21	0.038	**1400**
miniupnpd	165	DIR-882_A1	147	1.02	728.91	0.080	**9100**
upnpd	213	DGN2200	114	0.65	346.93	0.108	**3200**
setup.cgi	324	DGN1000	39	0.92	17.17	0.016	**1050**
httpd	994	DGN2200	168	1.59	831.41	0.138	**6000**

Note: 1. DDG corresponds to the method in angr, while FDDG corresponds to the method this paper proposed;

2. The results of time performance improvement have been rounded up.

从实现原理上看，angr 框架中的 DDG 构建方法考虑了程序中的所有变量，当一个函数内部存在循环时，由于很多变量之间存在循环依赖，故需要频繁对变量的数据依赖进行更新，耗费大量的时间.本文提出的 FDDG 构建方法，仅关心与特定函数(如不安全函数)相关的参数，不考虑无关变量，分析变量大大减少，大大降低了在循环内对变量进行数据依赖更新的频度.本文对比了两种数据依赖图中节点与边的规模，如表 6 所示，节点与边的数量分别相差约 3 个数量级.

总的来说，本文提出的 FDDG 构建方法极大地降低了分析所需的时间和空间开销，实现在相同计算资源条件下分析更大规模的程序，同时适用于对批量固件的快速分析.

Table 6 Comparison of the data dependency graph scale

表 6 数据依赖图规模对比

Name	*Size(k)*	*Model*	*Number of functions analyzed*	*Number of nodes*		*Number of edges*	
				DDG	*FDDG*	*DDG*	*FDDG*
stupid-ftpd	53	DIR-882_A1	24	102037	57	120825	12
upnp	90	HG532e	56	494854	372	585385	266
jjhttpd	118	DIR820LB1	45	544243	198	612616	157
lighttpd	145	HIVDC-F100V	136	2318991	428	2646642	491
cgibin	150	DIR-890L	356	2845062	1223	3132910	742
cgibin	155	DIR-645	88	634415	502	766330	205
miniupnpd	165	DIR-882_A1	147	3184594	697	3490395	505
upnpd	213	DGN2200	114	1850608	1291	2047508	1242
setup.cgi	324	DGN1000	39	157698	458	185527	177
httpd	994	DGN2200	168	2517499	801	2795711	566

Note: DDG corresponds to the method in angr, while FDDG corresponds to the method this paper. proposed.

4.2 固件脆弱性分析系统有效性评估

在前面分析中强调函数级数据依赖图保留了关键变量等有效信息，但能否在固件静态脆弱性分析中发挥效用，需要通过实验进行评估.

本文首先利用 FFVA 系统对表 4中的二进制程序样本进行分析，然后借助逆向分析工具如 IDA Pro[22]，对系统的分析结果进行人工审查，利用真实设备或 QEMU 模拟器[23]-[24]对脆弱点进行验证.实验结果如表 7所示，共发现漏洞 24 个，14 个属于未知漏洞.部分未知漏洞已经提交给厂商并得到确认，剩余的未知漏洞则利用真实设备或通过固件仿真的方式进行了验证.

Table 7 Results of vulnerability analysis on binary samples

表 7 二进制程序脆弱性分析结果

Name	*Size(k)*	*Model*	*Number of function analyzed*	*Time consumed (min)*	*Number of possible vulnerabilities found*	*Number of real vulnerabilities found*
wlancmd	38	HG532e	86	0.52	9	**4**
stupid-ftpd	53	DIR882_A1	56	0.46	6	**0**
upnp	90	HG532e	298	3.32	1	**1**
jjhttpd	118	DIR820LB1	281	3.25	1	**0**
lighttpd	145	HIVDC-F100V	327	2.42	0	**0**
cgibin	150	DIR-890L	445	2.77	5	**2**
cgibin	155	DIR-645	318	2.30	7	**4**
miniupnpd	165	DIR-882_A1	229	2.03	2	**0**
upnpd	213	DGN2200	268	4.47	7	**0**
setup.cgi	324	DGN1000	732	11.54	19	**5**
httpd	993	DGN2200	796	30.72	14	**2**
ipc_server*	2048	B18EN	383	3.62	11	**5**
mwareserver*	4812	IPC_6201	430	5.67	10	**1**

Note: As to ipc_server and mwareserver binary, only part of functions have been analyzed.

在表 7中，以 Honeywell 摄像头固件中的 lighttpd 程序为例，其主要负责处理与 HTTP 协议相关的请求，在其中发现的脆弱点数量为 0.借助 IDA Pro 工具对该程序进行逆向分析，发现程序中几乎没有使用 strcpy()、sprintf()等不安全函数，大部分的拷贝操作使用 memcpy()函数进行完成，并且对拷贝长度进行校验.通过对 stupid-ftpd 和 miniupnpd 程序进行逆向分析，发现其很可能是直接来源于开源软件，对脆弱点进行人工验证后没有发现漏洞.

由表 7可知，采用 FFVA 系统对二进制程序进行静态脆弱性分析时，其结果同样存在误报.从代码分析的角度，存在误报的主要原因是由于函数回调机制和间接调用的存在，导致进行数据依赖分析时会出现“断链”现象，即从潜在脆弱点无法直接回溯到外部输入点如 recv/recvfrom 等，进而无法判断参数的来源，造成对潜在脆弱点安全评估的不准确.此外，函数内由于不同的执行路径可能会对应不同返回值，对多个返回值的处理不当，也会对数据依赖分析造成影响.

在表 7中，分析二进制程序所耗费的时间包括程序分析和结果可视化两部分.以 NETGEAR httpd 为例，由于该程序内部函数较多且流程比较复杂，整体耗费时间约为 30 分钟，但真正进行程序分析所耗费的时间大约为 7 分钟，占比为 23.3%.二进制程序分析所耗时间占比如图 9 所示，由图 9 可知，大部分分析样例的整体耗费时间低于 5 分钟，而对于耗时超过 5 分钟的分析样例，真正进行程序分析所耗费的时间在整体中的占比均低于 35%.

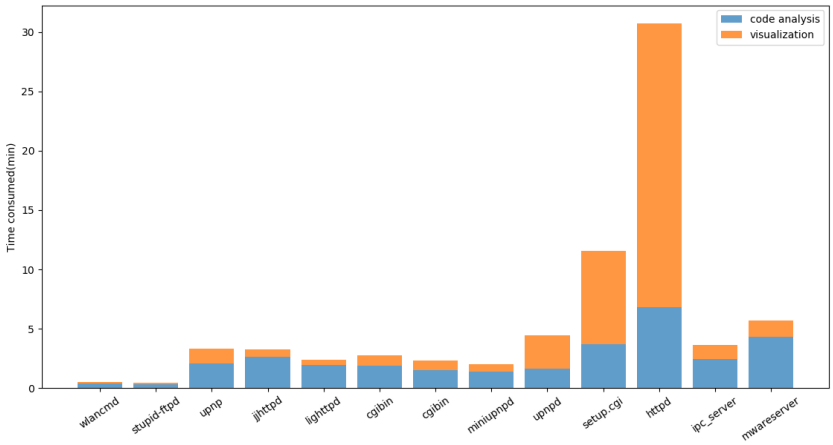


Fig. 9 Proportion of time consumed in analyzing vulnerability of binary samples

图 9 二进制程序脆弱性分析所耗时间占比

综上，利用 FFVA 系统对固件进行静态脆弱性分析，虽然结果存在误报，但其时间开销较低，同时能够发现程序中存在的安全缺陷或漏洞，表明了函数级数据依赖图在静态脆弱性分析中的有效性.

5 总结与展望

本文提出了函数级数据依赖图 FDDG 的概念，同时设计了 FDDG 的构建方法.与指令级数据依赖图相比，该图的粒度更“粗”，数据流更清晰，同时包含丰富的变量及语义信息.进一步，基于 angr 框架，在 FDDG 的基础上实现了一个固件脆弱性静态分析原型系统 FFVA.实验结果表明，FDDG 能够高效地应用在程序静态脆弱性分析中，并在 D-Link、NETGEAR、EasyN、uniview 等品牌的设备中发现了 24 个漏洞，其中 14 个为未知漏洞.本文验证了 FDDG 在静态分析中的有效性，但对 FDDG 是否会因为忽略信息导致脆弱性分析不全的问题只做了定性分析，后续将尝试通过模型方法论证.

References:

[1] Aho A V, Lam M S, Sethi R, et al. Compilers: Principles, Techniques, and Tools (2nd Edition)[M]. Addison-Wesley Longman Publishing Co. Inc. 2006.

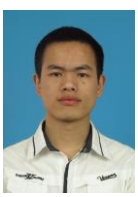
[2] Livshits V B, Lam M S. Finding security vulnerabilities in java applications with static analysis[C]. usenix security symposium, 2005: 18-18.

[3] Whaley J, Lam M S. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams[J]. programming language design and implementation, 2004, 39(6): 131-144.

[4] Jian L I, Liu J. Flow-sensitive Interprocedural Pointer Alias Analysis[J]. Computer Applications, 2004, 24(3):112-114.
[5] Doh K G, Shin S C. Detection of information leak by data flow analysis.[J]. Acm Sigplan Notices, 2002, 37(8):66-71.
[6] Dhavale S V, Lokhande B. Comnoid: Information Leakage Detection using Data Flow Analysis on Android Devices[J]. International Journal of Computer Applications, 2016, 134(7): 15-20.
[7] Liu X, Liu J, Wang W, et al. Discovering and understanding android sensor usage behaviors with data flow analysis[J]. World Wide Web-internet & Web Information Systems, 2017(11):1-22.
[8] Wu S Z, Guo T, Dong G W. Software Vulnerability Analyses[M]. Beijing: Science Press, 2014.
[9] Jovanovic N, Kruegel C, Kirda E. Pixy: A static analysis tool for detecting web application vulnerabilities[C]//Security and Privacy, 2006 IEEE Symposium on. IEEE, 2006: 6 pp.-263.
[10] FindBugs. http://www.ibm.com/developerworks/java/library/j-findbug1.
[11] Fortify. http://fortify-sca.software.informer.com.
[12] Coverity.https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html.
[13] IBM Appscan. https://www.ibm.com/security/application-security/appscan.
[14] Pinpoint. https://www.sourcebrella.com/.
[15] CodeSonar. https://www.grammatech.com/products/codesonar.
[16] Ferrante J, Ottenstein K J, Warren J D. The program dependence graph and its use in optimization[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1987, 9(3): 319-349.
[17] Shoshitaishvili Y, Wang R, Salls C, et al. (State of) The Art of War: Offensive Techniques in Binary Analysis[C]. Proceedings of the 2016 IEEE Symposium on Security and Privacy. 138-157.
[18] angr, a binary analysis framework[Online]. Available:http://angr.io/index.html.
[19] OWASP Foundation, Inc. OWASP Embedded Application Security[EB/OL].[2018-01-02].https://www.owasp.org/index.php/OWASP_Embedded_Application_Security.
[20] Schwartz E J, Avgerinos T, Brumley D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)[C]. Security and privacy (SP), 2010 IEEE symposium on. IEEE, 2010: 317-331.
[21] Wang T L. Research on binary-executable-oriented software vulnerability detection [D]. Beijing: Peking University, 2011.
[22] Hex-Rays. https://www.hex-rays.com/.
[23] Bellard F. QEMU, a fast and portable dynamic translator[C]. USENIX Annual Technical Conference, FREENIX Track. 2005, 41: 46.
[24] Wu S H, Wang W, Zhao X. Revealing the art of 0 day mining in home router[M]. Beijing: Publishing House of Electronics Industry, 2015.

附中文参考文献:

[4] 李健,刘坚. 流敏感的跨过程指针别名分析[J]. 计算机应用, 2004, 24(3): 112-114.
[8] 吴世忠,郭涛,董国伟. 软件漏洞分析技术[M]. 北京:科学出版社,2014.
[21] 王铁磊. 面向二进制程序的漏洞挖掘关键技术研究[D]. 北京大学, 2011.
[24] 吴少华,王炜,赵旭. 揭秘家用路由器 0day 漏洞挖掘技术[M]. 北京:电子工业出版社,2015.



陈千(1993-),男,硕士,主要研究领域为嵌入式设备安全.

朱红松(1973-),男,博士,研究员,博士生导师,主要研究领域为物联网安全、网络攻防、安全大数据分析与测评.

程凯(1991-),男,博士生,主要研究领域为IoT 安全、二进制固件的脆弱性分析.

孙利民(1966-),男,博士后,研究员,博士生导师,主要研究领域为物联网及其安全、工业控制系统安全、区块链安全.

郑尧文(1990-),男,博士生,主要研究领域为 IoT 安全、固件逆向分析与漏洞挖掘.