

基于双向切片的软件漏洞分析技术研究

于 璐¹, 吴礼发¹, 庄洪林¹, 沈 毅²

(1 解放军理工大学, 江苏 南京 210007; 2 解放军电子工程学院, 安徽 合肥 230037)

摘 要: 当前对二进制软件漏洞的分析多基于传统的逆向工程, 这种方法很大程度上依赖分析人员的经验, 效率低。由此针对二进制软件的实际特点对已有切片技术进行改进, 对软件的执行记录进行双向切片, 获得导致异常的关键指令序列, 只需对关键序列进行少量人工分析即可分析定位异常代码, 提高漏洞分析效率。

关键词: 二进制软件; 漏洞分析; 自动化; 双向切片

中图分类号: TP393

文献标识码: A

文章编号: 1000-7180(2011)08-0173-03

Bidirectional-Slicing Based Software Vulnerability Analysis

YU Lu¹, WU Li-fa¹, ZHUANG Hong-lin¹, SHEN Yi²

(1 Institute of Command Automation, PLA University of Science & Technology, Nanjing 210007, China;

2 Electronic Engineering Institute of PLA, Hefei 230037, China)

Abstract: Nowadays the vulnerability analysis for binary software is mainly based on the traditional reversing engineering, which depends on the analyzers' experience and is of low efficiency. This paper makes improvement on existing slicing technology and implementing bidirectional slicing to the binary software trace and obtaining key instruction sequences that lead to the exception. With little human analysis the fault localization can be made, which can improve the vulnerability analysis efficiency.

Key words: binary software; vulnerability analysis; automation; bidirectional slicing

1 引言

程序切片最早由 Weiser^[1]提出, 它基于程序的数据或控制依赖关系, 获得影响某给定变量的语句集合。切片技术被广泛应用于程序调试、代码聚合度量、程序理解、程序正确性分析和软件安全性分析等方向。

本文将切片技术应用于对二进制软件的漏洞分析中。通过记录软件执行过程, 对执行记录进行双向切片, 获得与异常最相关的指令序列及其他辅助信息, 在少量人工分析的参与下, 有效定位异常代码来辅助漏洞分析, 提高漏洞分析的自动化水平和效率。

2 双向切片概述

双向切片技术的实现基于目标软件的执行记录内容^[2], 将复杂度较低的数据依赖关系作为切片技

术的依据^[3]。后向切片在程序执行异常时, 由直接导致异常的变量开始, 得到影响该变量的所有语句。前向切片对指令序列继续筛选, 得到受外界输入影响的语句进行重点分析。在实现上对双向切片进行优化, 具体执行流程图如图 1 所示。

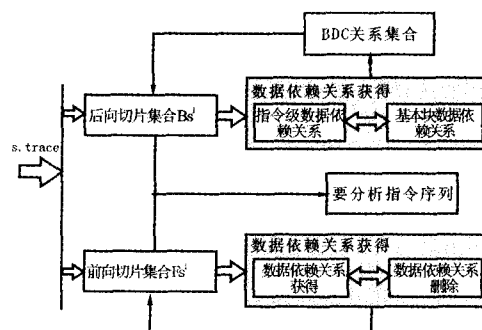


图 1 双向切片的具体流程图

3 后向切片

对导致异常的直接变量进行切片获得与其具有数据依赖关系的变量和指令集合. 进行后向切片时, 定义基本块级别和指令级别不同粒度的依赖关系. 基本块级别的数据依赖关系通过数据依赖方程获得, 自定义指令级别的依赖关系, 最后完成切片算法.

3.1 数据依赖方程的建立

将指令中的操作数统称为变量, 变量的影响范围多在跳转或者调用指令执行到另外一个基本块时结束. 基本块间变量的数据传递关系可以通过求解“到达一定值”数据流方程^[4]得到.

数据流方程的解不仅能获得基本块间变量的数据依赖关系, 还能直观表示某变量的影响范围是否结束, 作为后续切片依据.

3.2 指令级别数据依赖关系的定义

由于汇编代码中的指令种类有限, 指令级别的数据关系更依赖具体指令.

使用 $\text{Relation}(i)$ 表示指令 i 处操作数的依赖关系, 使用 $\text{Re} \leftarrow \text{ReR}_1 \mid \text{ReR}_2 \mid \dots$ 表示, 表示的是依赖关系: ReR_1 、 ReR_2 等依赖于 Re . 对于 $\text{mov reg}_1, \text{mem}$, 影响寄存器 reg 的值包括 mem 地址及其指向的值, 因此定义 $\text{reg}_1 \leftarrow \text{mem} \mid [\text{mem}]$.

3.3 后向切片的实现算法

使用 ADFDSA (Advanced Data-Flow Dynamic Slicing Algorithm) 描述切片算法, 算法使用递归搜索集合 V 中需要记录的变量, 根据数据依赖关系, 将需要继续回溯切片的变量和已记录影响目标变量的值分开记录, 以减少冗余度. 具体算法如下所示:

输入: Er 表示导致异常的直接变量集合; CompList 记录要继续分析的变量值和所在行数;

输出: V 是确定对目标值有影响的变量及行数值的集合;

变量说明: $\text{line}(v)$ 是 v 所在指令行数; $\text{Block}(i)$ 表示行 i 的指令所在的基本块; $\text{RCHin}(Bi)$, ReR 和 $\text{RCHout}(Bi)$, ReR 表示基本块 Bi 输入和输出的位向量表中 ReR 的位值; $\text{Relation}(i)$ 表示指令 i 处的数据依赖关系, 使用 $\langle \text{Re}, \text{ReR} \rangle$ 表示.

1. ADFDSA($\text{CompList}, i$)
2. for $\langle x, \text{line}(x) \rangle$ in CompList
3. for $\langle \text{Re}, \text{ReR} \rangle$ in $\text{Relation}(i)$
4. if $x == \text{Re}$
5. if $\text{Re} == \text{ReR}$

6. $V. \text{update}(v, i)$
7. else
8. $V. \text{add}(\text{ReR}, i)$
9. if (any $\langle \text{Re}_1, \text{ReR}_1 \rangle$ in $\{V - \langle x, \text{line}(x) \rangle\} \& \& x! = \text{Re}$)
10. $V. \text{delete}(x, \text{line}(x))$
11. if ins i is exit node
12. if $\text{RCHin}(\text{Block}(i)), \text{ReR} = 1 \& \& \text{RCHout}(\text{Block}(i)), \text{ReR} = 0$
13. $\text{CompList. delete}(\text{ReR})$
14. ADFDSA($\text{CompList}, i-1$)

第 4~10 行代码说明如果 CompList 中 x 依赖其本身, 则将对记录中的行数更新, 否则向 V 中添加新的依赖变量, 如果 x 完全依赖于另一变量, 则将 x 的值从列表中删除. 11~13 行在基本块出口指令处根据输入输出的位向量判断变量的影响范围是否结束, 如果结束, 则将其从对应的列表中删除.

4 前向切片

作为后向切片的辅助, 前向切片只需记录执行指令序列中受输入数据影响的操作数. 同时为了降低复杂度和资源消耗, 只对受输入影响的操作数标记而不需另外记录.

前向切片在实现上的不同在于其存在数据标记的删除. 将指令分为三个集合: 数据传送和一般运算指令集 (NormalIns)、导致标记消除的特殊指令集 (SpeciIns) 和其他的指令集 (Remainder). NormalIns 中指令与后向切片中定义的依赖关系相同; 而 SpeciIns 中的指令会导致操作数的值不再受之前操作影响, 需适时对标记进行删除; Remainder 中的指令不需进行任何处理. 前向切片 FSA (Forwarding Slicing Algorithm) 的算法描述如下:

变量说明: $\text{Ins}(i)$ 表示第 i 个指令; $\text{Relation}(i)$ 表示第 i 个指令对应的数据依赖关系

输入: 汇编指令集 $\text{NormalIns}, \text{SpeciIns}, \text{Remainder}$

输出: V 表示标记为 1 的变量组成的集合

- 1 FSA(V, i) // 对第 i 条指令的处理
- 2 for $\text{Relation}(i)$
- 3 if $\text{Ins}(i) \text{Remainder}$
- 4 FSA($V, i+1$) // 不做任何处理
- 5 else if $\text{Relation}(i), \text{Re. flag} == 1$
- 6 if $\text{Ins}(i) \text{SpeciIns}$
- 7 $V. \text{remove}(\text{Relation}(i), \text{Re})$
- 8 else
- 9 $V. \text{add}(\text{Relation}(i), \text{ReR})$ // 添加标记

```

10 V.update()
11 Relation(i), ReR.flag = 1
12 FSA(V, i + 1)

```

5 实验分析

5.1 简单实例验证

使用一个简单缓冲区溢出程序说明双向切片的过程,并通过对执行记录的分析验证切片的有效性.关键代码如下:

```

void copy(char *input) {.....
    strcpy(output, input);}
int main() {.....
    copy(input);.....}

```

如果输入参数 input 的长度过长就会覆盖函数返回地址导致异常.使用超长字符串(12 个“a”)作为输入,异常指令为 ret 0x0012ff84[0x61616161].进行双向切片可得到指令序列如下所示,指令前的数字表示该指令的执行序列号:

```

(00000040) mov ecx, DWORD PTR [ebp - 0x34]
(00000044) shr ecx, 0x2
(00000045) rep movs DWORD PTR es:[edi],

```

```

DWORD PTR ds:[esi] .....

```

```

(00001062) mov BYTE PTR [esi], al

```

分析序号 40 的指令,其 ecx 值为 0x0c, ecx 控制的是指令 rep movs DWORD PTR es:[edi], DWORD PTR ds:[esi] 的循环次数,而 esi 和 edi 分别记录变量 input 和 output 的地址,可以判断 ecx 就是输入字符串的长度.因此导致异常的是 40 处 ecx 的值与相应的 esi 和 edi 值,验证了漏洞产生的原因.

此外通过执行序号可以看出,序号 40 到 1062 的指令序列,对其进行双向切片得到的待分析指令只有 8 个,可以看出这种方法可以大大降低分析工作量.

5.2 双向切片的有效性验证

将已公布的常用软件漏洞作为研究的目标,记录其执行的指令序列并记录单独使用前向、后向和双向切片得到的指令条数.后向切片技术基于迭代方法,使用 RBS 表示后向切片方法中的迭代次数, NBS 则表示后向切片多次迭代后得到的所有指令数目;FS 表示的是前向切片获得的受输入影响的变量数目;BFS 则表示将两种切片方法结合得到的指令数目. Org 表示目标软件执行的指令总数目.得到的实验数据如表 1 所示.

表 1 相关软件切片结果

漏洞编号	Org	RBS	NBS	FS	BFS	BFS/Org(e-5)
CVE-2008-6415	45 019 952	584	4 529	1 587 924	3 144	6.983
CVE-2007-1404	151 321 654	46	79 564	2 456 211	1 055	6.971
CVE-2010-2204	1 173 735	22	1 031	255 458	1 005	85.62
CVE-2010-2862	725 894 584	58	37 255	4 547 885	10 224	1.408

6 结束语

本文针对二进制软件漏洞分析中人工分析效率低下的问题,对软件的执行指令序列进行双向切片,实现对漏洞较为自动化分析.在下一步工作中,可以将对执行记录的比较方法加入到切片中,从而进一步筛选切片得到的指令序列,降低分析的工作量.

参考文献:

- [1] Mark Weiser. Program slicing[C]// Proceedings of the 5th International Conference on Software Engineering. Piscataway, NJ, USA: IEEE, 1981:439-449.
- [2] Zhang X, Gupta N. Whole Execution Traces[J]. IEEE/ACM 37th International Symposium on Microarchitec-

ture, 2004:105-116.

- [3] Zhang X, Gupta R, Zhang Y. Precise dynamic slicing algorithms[C]// IEEE/ACM international conference on software engineering. Portland, Oregon: IEEE, 2003:319-329.
- [4] Steven S. Muchnick. Advanced Compiler Design and Implementation [M]. San Francisco, USA: Mprgan Kaufmann Publishers, 1997.

作者简介:

于璐女,(1985-),硕士研究生.研究方向为信息与网络安全.

吴礼发男,(1968-),教授.研究方向为网络信息安全.

庄洪林男,(1967-),高工.研究方向为网络安全.

沈毅男,(1985-),助教.研究方向为网络安全.