基于安全风险的 RTL 级硬件木马验证研究

赵剑锋[1,2], 史 岗[2]

[1] 中国科学院大学 网络空间安全学院 北京 中国 100049
[2] 中国科学院信息工程研究所 第五实验室 北京 中国 100093

摘要 信息时代使得信息安全变得日益重要。攻击方为了获取想要的信息, 除了使用软件方面的手段, 如病毒、蠕虫、软件木马等, 也使用硬件手段来威胁设备、系统和数据的安全, 如在芯片中植入硬件木马等。如果将硬件木马植入信息处理的核心--处理器, 那将风险更高、危害更大。然而, 硬件木马位于信息系统底层核心的层面, 难以被检测和发现出来。硬件木马是国内外学术界研究的热点课题, 尤其是在设计阶段结合源代码的硬件木马检测问题, 是新问题, 也是有实际需要的问题。在上述背景下, 围绕源代码中硬件木马的检测和验证展开了研究。基于硬件木马危害结果属性, 在学术上提出基于安全风险的模型和验证规则, 给出相应的描述形式, 从理论上说明安全验证规则在减少验证盲目性、缩小可疑代码范围、提高评估效率的作用, 实验表明, 基于安全风险规则的验证, 可以避免验证的盲目性和测试空间向量膨胀的问题, 有效验证疑似硬件木马的存在和危害, 对源代码安全评估是有一定效果的。

关键词 芯片; RTL 级硬件木马; 安全风险; 验证规则
中图法分类号 TP309.2 DOI 号 10.19363/J.cnki.cn10-1380/tn.2022.12.08

Research on RTL Level Hardware Trojan Verification Based on Security Risk

Zhao Jianfeng[1,2], Shi Gang[2]

[1] School of Cybersecurity, University of Chinese Academy of Sciences, Beijing 100049, China
[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

Abstract With the advent of the information age, information security has become increasingly important. In order to obtain the desired information, attackers not only use software means, such as viruses, worms, software trojans, but also use hardware means to threaten the security of devices, systems and data, such as hardware Trojans embedded in chips. If the hardware Trojan horse is embedded in the processor, which is the core of information processing, the risk will be higher and the harm will be greater. However, the hardware Trojan horse is located at the bottom of the information system core level, which is difficult to detect and discover. Hardware Trojan Horse is a hot topic in academic circles at home and abroad. Especially in the design stage, the problem of hardware Trojan Horse detection combined with source code is not only a new problem, but also a necessary one. This paper is based on the above background and combined with the actual needs of domestic chip RTL source code security risk assessment to carry out related work, mainly for the detection and verification of hardware Trojan in RTL source code. The main contents and contributions of this paper are as follows. Aiming at the problem that RTL level hardware Trojan has not yet given its characteristic attributes academically, the description form of hardware Trojan's attribute is given Based on the harm result attribute of Hardware Trojan, the model and verification rules based on security risk are put forward academically, and the corresponding description form is given, this paper theoretically explains the role of security verification rules in reducing the blindness of verification, reducing the scope of suspicious code, and improving the efficiency of evaluation, it can avoid the blindness of verification and the expansion of test space vector, and effectively verify the existence and harm of suspected hardware Trojans, which is effective for RTL source code security assessment.

Key words chip; RTL level hardware trojan; security risk; verification rules

1 引言

信息安全深刻影响着世界的政治、经济、军事等领域。信息安全包括软件安全和硬件安全, 除了人们熟知的软件安全威胁, 如病毒、木马和蠕虫外, 攻击方已经渗透到硬件安全, 尤其作为信息处理核心

通讯作者: 赵剑锋, 博士研究生, 讲师, Email: zhaojianfeng@iie.ac.cn。
本课题得到国家“核高基”科技重大专项基金项目(No.2013ZX01029003-001); 国家“八六三”高技术研究发展计划基金项目(No.2012AA01A401)资助。
收稿日期: 2020-05-03; 修改日期: 2020-06-12; 定稿日期: 2022-12-07

的处理器芯片的安全, 更是重中之重[1-4]。

由芯片的设计与制造的流程可以看出, 芯片生产设计制造的复杂程度为芯片安全埋下了隐患, 导致芯片的安全性变得越来越脆弱。

2007 年, IBM 沃森研究中心联合伍斯特理工学院首次提出了硬件木马的概念[5]。之后, 经过国外相关研究人员, 如 D. Jia[6], X. Wang 和 M.Tehranipoor, Yier Jin[7]等以及国内(如西北工业大学、中国科学院信息工程研究所等高校和研究机构)硬件木马研究人员[8-9]的多方面深入地拓展研究, 形成了硬件木马的一般概念: 硬件木马主要是指在集成电路芯片(可以在设计、制造、生产的各个阶段)中故意植入含有恶意功能的逻辑电路, 并且某种特定条件下触发该恶意逻辑功能, 以达到攻击方非法的目的(如信息泄漏、拒绝服务、毁坏芯片、系统权限提升等)。

硬件木马研究作为国内外学术界关注的焦点, 涉及多个研究方面, 包括硬件木马设计、检测、验证及防范等[10-15]。其中, 关于第三方源代码安全评估问题(由于芯片设计与制造过程相分离, 导致第三方源代码安全不可控, 特别是对于国防、政府、金融、交通等敏感领域而言, 更是如此), 是热点问题, 同时也是难点问题。

文章围绕 RTL(Register Transfer Level)级源代码(硬件描述语言书写的)中硬件木马的问题展开了研究。主要贡献: 针对 RTL 级硬件木马尚未在学术上给出一般安全风险模型和指导规则, 结合对标准集的分析结果, 提出相应的模型和定义, 给出安全风险模型和描述形式, 在安全风险模型的基础上, 提出了安全风险验证规则, 从理论上说明基于安全风险验证规则, 可以减少验证盲目性、缩小可疑代码范围、提高源代码评估效率。实验表明, 基于安全风险的验证, 对 RTL 级源代码评估是有一定效果的。

文章主要包括以下部分: 第 2 小节, 给出当前常见的评估验证方法; 第 3 小节, 在硬件木马集的基础上抽象出硬件木马一般模型和定义、安全风险模型, 并提出安全风险验证规则, 指导验证工作; 第 4 小节给出实验; 最后是总结与展望部分, 对研究内容进行总结, 并提出了下一步的研究方向。

2 相关方法

如今, 集成电路设计与制造技术发展越来越快, 规模也是越来越大, 要对日益庞大的集成电路进行完整地验证, 其难度也是与日俱增。有研究数据指出, 要开发一款芯片, 验证付出的时间占整个工程周期的 3/4 左右, 而验证人员也比设计人员要多一倍。验证时间的长短和效率在一定程度上决定芯片的上市日期, 也决定了企业、公司的前途和命运, 这就对验证工作的效率提出了迫切的需求。

当前, 验证方法主要分成形式化方法和非形式化方法。

形式化方法是严格基于数学理论的方法, 采用相应形式化语言来描述目标系统性质, 根据严格的数学符号和相应法则, 对系统结构、行为进行简洁高效的描述、分析、推理和演算, 以此为系统属性的说明、开发、验证设计了框架, 能够辅助发现目标系统设计需求中的不一致性、不完备性等非正常情况。形式化方法主要包括三种方式, 分别是等价性验证、模型检验和定理证明。

形式化等价检查提供了一种等价检测机制, 能够验证 RTL 与 RTL, RTL 与 netlist 或者 netlist 与 netlist 之间的等价性。该方法的优点是利用逻辑等价性加快验证速度, 同时能够保证能够提供 100%的覆盖率, 该方法最大的优点是不需要外部输入的测试向量, 故而可以节约测试时间, 缩短测试周期。这种方法在工业界已经得到广泛的应用。如乘法器件、GPU 等中的应用。模型验证就是比较已设计的产品的行为和用户定义的逻辑特征是否相符合。模型检验的方法目前已经开始为工业界所接受, 其优点是完全自动化, 当系统不满足给定的性质时, 检验结果会给出反例, 来帮助验证人员发现错误。定理证明一般采用交互式的定理辅助证明器来对系统问题进行抽象描述, 并以数学公式定理的方式表达系统的功能和安全性。采用数学定理推导演算的方法进行验证, 目前未被业界广泛采用。

非形式化方法主要是模拟验证方法, 内建自测方法。

模拟验证是集成电路设计验证经常采用的方法[16]。其原理是: 通过对待验证电路施加激励, 将得到的响应与预期结果比较, 来发现设计中的问题。

内建自测方法, 简称为 BIST(Built-In Self-Test), 实质是在设计一个芯片时, 在其内部加外额外的逻辑功能模块[17], 通过额外的逻辑功能模块可以监测芯片内部的信号状态正常与否。在芯片实际运行当中, 安全的芯片通过 BIST 电路生成一个签名, 如果芯片中含有硬件木马, 则会生成另外一个不同的签名。

最近几年, 提出了不少新的验证研究方法, 如西北工业大学研究团队提出的门级信息流方法, 还有统计学习方法[18]、SVM(Support Vector Machine)[19]、机器学习[20-21]、自学习[22]、深度学习[23]、

多层神经网络[24]、人工免疫系统及行为分类[24]等多种方法。

对于 RTL 级源代码安全性评估和验证而言, 当前方法存在的主要问题有:

(1) 缺乏一般的安全风险模型和指导规则, 对于庞大的 RTL 级源代码安全性评估和验证而言, 缺少针对性, 即哪些是评估的重点区域, 哪些地方存在安全风险, 不知道这些, 会导致安全评估和验证具有盲目性[25-29];

(2) 形式化方法中的等价性检查仅仅验证 RTL 与 RTL, RTL 与 netlist 或者 netlist 与 netlist 之间的等价性, 模型检验存在状态空间爆炸的问题, 定理证明方法尚未被业界广泛采用;

(3) 非形式化方法中的模验证方法, 主要从正向来验证芯片的逻辑功能是否满足要求, 其目的是验证芯片是否满足设计功能, 需要庞大的测试向量, 测试时间长; 内建自测方法需要在芯片中加入额外逻辑功能模块, 增加硬件开销;

(4) 门级信息流方法、机器学习方法及自学习方法等, 需要将 RTL 源代码转换成门级网表, 使其逻辑锥庞大, 同样存在着验证空间过大的问题。

基于以上方法存在的问题, 在面临 RTL 级源代码安全评估和验证时, 试图在分析现有硬件木马集的基础上, 给出硬件木马的一般模型, 提炼出安全风险模型, 以此得出面向安全风险的验证规则, 来指导 RTL 级源代码的安全性评估和验证工作。

3 安全风险模型及规则

3.1 安全风险模型及规则提出原因

(1) 在芯片设计过程中, 除了自主开发源代码以外, 会购买集成大量的第三方的 IP 核(Intellectual Property core), 包括含 RTL 源码的 IP 核。在对其进行安全评估时, 面临的源文件多, 源代码量大, 如果只是单纯采取传统验证分析方法, 必然导致大量的人力、物力、财力消耗, 并且验证效果也难以保证。因此, 如果能提出相应的安全风险模型和指导规则, 对于 RTL 源代码安全性评估和验证具有重要的工程意义。

(2) 从硬件木马攻击者角度出发, 硬件木马攻击者为了达到攻击目的, 必然要设计隐藏性很好的木马, 才能有效保护自己, 待到特定条件时达到攻击的目的。它的触发和激励的位长有关系, 即位数长, 硬件木马被触发的概率小; 位数短, 被触发的概率大。如果激励空间不是很大, 在计算量可忍受范围内, 则通过全覆盖可以测出。如果激励空间范围大, 如在 2^{128} 个向量中取某个值, 假设向量空间的取值是服从均匀分布的, 则一次触发硬件木马的概率为 2^{128} 分之一, 这个概率几乎是 0。为了发现这种木马, 就要产生很大的测试向量, 这在时间上耗费太大, 通过正向测试的方法难以发现这样的硬件木马, 那么如果通过对源代码的分析, 结合安全风险模型和指导规则, 并且根据查找到的有关参量位产生测试激励, 它比盲目产生随机测试激励针对性更强。

3.2 基于标准集分析结果

Trust-Hub 含有 52 不同模块硬件木马样本, 包括 21 个 AES 加密系列模块实例, 10 个 RS232 系列模块, 7 个 MC8051 系列模块, 4 个 PIC16F84 系列模块, 4 个 RSA 系列模块, 3 个 B19 系列模块, 2 个 wb_conmax 系列模块和 1 个 memctrl 模块。

Trust-Hub 标准集是研究硬件木马检测和验证的样本库, 通过对硬件样本集的分析来抽象硬件木马的一般概念和安全风险模型, 并在此基础上提出安全风险验证规则, 以指导 RTL 级源代码安全评估和验证工作。

对 Trust-Hub 标准集统计分析可以得到硬件木马所处位置、危害结果及特征表现的数据。

从表 1 可以看出, 硬件木马直接依附于 I/O 端口的占到 69.2%, 依附于状态机的占到 15.4%, 依附于特殊寄存器(如计数器、特权模式设置寄存器等)占到 15.4%。

表 1 硬件木马所处位置统计数据[30]

Table 1 Statistical data about location of hardware trojan[30]

所处位置	百分比/%
I/O 端口	69.2
状态机	15.4
特殊寄存器	15.4

在表 2 中, 统计出四种危害结果, 包括信息泄漏、功能改变、拒绝服务和破坏芯片, 其所占百分比分别为 38.5%、46.2%、9.6%、5.8%, 在后续的论述中, 结合现实情况, 加入了系统权限提升这一危害。

表 2 硬件木马危害结果统计数据[30]

Table 2 Statistical harm data of hardware trojan[30]

危害结果	百分比/%
信息泄漏	38.5
功能改变	46.2
拒绝服务	9.6
破坏芯片	5.8

表 3 给出了危害结果与所处位置之间的统计关系。

表 3 危害结果与所处位置的统计关系数据[30]

Table 3 Relationship between location and harm result[30] (%)

所处位置	信息泄漏	功能改变	拒绝服务	破坏芯片
I/O 端口	70.0	60.0	100.0	33.3
状态机	0.0	25.0	0.0	66.7
特殊寄存器	30.0	15.0	0.0	0.0

表 4 给出了常见硬件木马的书写特征表现形式, 可以作为验证可疑硬件木马的参考标准。

表 4 Trust-Hub 硬件木马特征[31]

Table 4 Characteristics of Trust-Hub hardware trojan[31]

类别	特征表现
1	子模块时钟信号被数值驱动, 造成工作不正常
2	被驱动信号出现子模块时钟信号
3	always 块敏感列表中的信号出现概率小或条件一直满足
4	if 语句条件中出现特定序列值
5	驱动信号列表出现多个信号且夹杂着降低出现概率的符号
6	计数器出现特殊序列数值

3.3 基本定义和安全风险模型

根据 3.2 节分析结果, 可以给出一个一般性的硬件木马模型如下图。

根据图 1 模型, 可以给出如下相关定义。

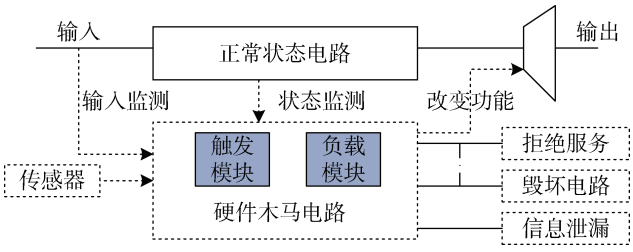

图 1 硬件木马模型示意图

Figure 1 Schematic diagram of hardware trojan model

定义 1:

芯片 IC 可以抽象为一个非空集合, IC={I, O, N, A, NF, AF}; 其中:

I: 表示输入端口集合, $I=\{I_1, I_2, \ldots, I_n\}$, 其中下标 n 表示输入端口个数;

O: 表示输出端口集合, $O=\{O_1, O_2, \ldots, O_m\}$, 其中下标 m 表示输出端口个数;

N: 表示正常状态的电路集合, $N=\{N_1, N_2, \ldots, N_k\}$, 其中下标 k 表示正常电路集合个数;

A: 表示硬件木马电路集合, $A=\{A_1, A_2, \ldots, A_l\}$, 其中下标 l 表示硬件木马电路集合个数;

NF: 表示芯片正常的功能集合, $NF=\{NF_1, NF_2, \ldots, NF_r\}$, 其中下标 r 表示正常电路功能个数;

AF: 表示硬件木马的危害功能集合, $AF=\{AF_1, AF_2, \ldots, AF_s\}$, 其中下标 s 表示硬件木马电路功能个数。

定义 2:

设 X 为要安全评估和验证的 RTL 级源代码, 同样为一个非空集合 X={I, O, N, A, NF, AF}, 对其进行安全评估和验证, 依据定义 1, 当满足:

A=∅, 且 AF=∅时, 说明源代码安全, 否则是不安全的, 其中∅表示空集。

定义 3: 将定义 1 中危害功能集合 $AF=\{AF_1, AF_2, \ldots, AF_s\}$定义为安全风险集合, 每一个危害功能与一个安全风险相对应, 结合 3.2 节分析及现实情况, 安全风险集合 $S=\{I_{leakage}, F_{change}, D_{service}, C_{destroy}, S_{empower}\}$, 其中:

$I_{leakage}$ 表示信息泄漏安全风险;

F_{change} 表示功能改变安全风险;

$D_{service}$ 表示拒绝服务安全风险;

$C_{destroy}$ 表示破坏芯片安全风险;

$S_{empower}$ 表示提升权限安全风险。

定义 4: 将 3.2 节中表 3.1 中硬件木马位置定义为安全风险位置, 安全风险位置集合 $L=\{P_{io}, S_{machine}, R_{in}\}$, 其中:

P_{io} 表示处于输入或输出端口位置;

$S_{machine}$ 表示处于状态机位置;

R_{in} 表示处于内部特殊寄存器位置。

定义 5: P_i 表示安全风险位置集合 L 中第 i 个元素出现的概率, 则由概率论基础知识可知, $0 \leqslant P_i \leqslant 1$, 其中, $1 \leqslant i \leqslant |L|$, 其中| |表示取集合 L 元素的个数。

在定义 5 的基础上, 借助于信息熵的概念, 提出安全风险熵 $H_i = -\log_2(1/P_i)$;

H_i 取值大表示安全风险位置出现的概率小, 检测和验证相对困难;

H_i 取值小表示安全风险位置出现的概率大, 检测和验证相对容易。

定义 6: 定义安全风险评估分类集合 E 为定义 3 中的安全风险集合 S 与定义 4 中的安全风险位置集合 L 的笛卡尔积, 集合 E 中每一元素表示安全风险评估的一个类别, 即安全风险位置是什么, 对应的安全风险危害是什么。

$$E = S \otimes L = \begin{bmatrix} (I_{leakage}, P_{io}) & (I_{leakage}, S_{machine}) & (I_{leakage}, R_{in}) \\ (F_{change}, P_{io}) & (F_{change}, S_{machine}) & (F_{change}, R_{in}) \\ (D_{service}, P_{io}) & (D_{service}, S_{machine}) & (D_{service}, R_{in}) \\ (C_{destroy}, P_{io}) & (C_{destroy}, S_{machine}) & (C_{destroy}, R_{in}) \\ (S_{empower}, P_{io}) & (S_{empower}, S_{machine}) & (S_{empower}, R_{in}) \end{bmatrix}$$

硬件木马在位置、危害、特征表现上各不相同, 根据 3.2 节标准集分析结果及以上定义, 可以给出以下安全风险模型及相应的形式描述。

(1) 安全风险模型一

如图 2 所示, 图中带箭头实线表示正常在状态转换, 虚线表示转移到安全风险状态。

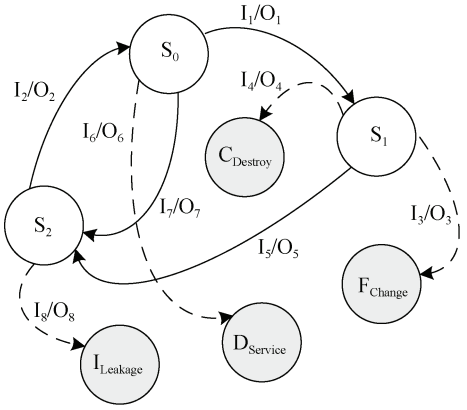

图 2 安全风险模型一

Figure 2 Schematic diagram of security risk model one

图 2 安全风险模型一的形式化表示为一个四元组(S, I, O, T), 其中:

S: 有穷非空状态集{S_0, S_1, S_2, $I_{leakage}$, F_{change}, $D_{service}$, $C_{destroy}$ }, 其中 S_0, S_1, S_2 表示正常状态, $I_{leakage}$ 表示信息泄漏安全风险状态, F_{change} 表示功能改变安全风险状态, $D_{service}$ 表示拒绝服务安全风险状态, $C_{destroy}$ 表示破坏芯片安全风险

I: 有穷非空输入集{I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8 }

O: 有穷非空输出集{ O_1, O_2, O_3, O_4, O_5, O_6, O_7, O_8 }

T: 表示在输入 I 集中某元素时, S 状态集中某两个状态之间的转换函数

=>: 表示状态转换

状态描述如下:

S_0+ I_1 => S_1+ O_1 表示当前状态为 S_0, 在输入 I_1 的情况下, 转换到状态 S_1, 输出为 O_1;

S_1+ I_5 => S_2+ O_5 表示当前状态为 S_1, 在输入 I_5 的情况下, 转换到状态 S_2, 输出为 O_5;

S_0+ I_7 => S_2+ O_7 表示当前状态为 S_0, 在输入 I_7 的情况下, 转换到状态 S_2, 输出为 O_7;

S_2+ I_2 => S_0+ O_2 表示当前状态为 S_2, 在输入 I_2 的情况下, 转换到状态 S_0, 输出为 O_2;

S_1+ I_4 => $C_{Destroy}$+ O_4 表示当前状态为 S_1, 在输入 I_4 的情况下, 转换到状态 $C_{Destroy}$, 输出为 O_4;

S_1+ I_3 => F_{Change}+ O_3 表示当前状态为 S_1, 在输入 I_3 的情况下, 转换到状态 F_{Change}, 输出为 O_3;

S_0+ I_6 => $D_{Service}$+ O_6 表示当前状态为 S_0, 在输入 I_6 的情况下, 转换到状态 $D_{Service}$, 输出为 O_6;

S_2+ I_8 =>$I_{Leakage}$+ O_8 表示当前状态为 S_2, 在输入 I_8 的情况下, 转换到状态 $I_{Leakage}$, 输出为 O_8。

(2) 安全风险模型二

如图 3 所示, 图中带箭头实线表示正常在状态转换, 虚线表示转移到非法状态。

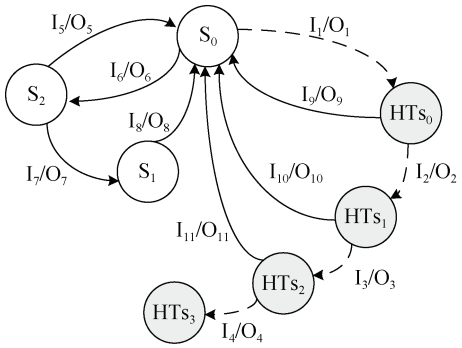

图 3 安全风险模型二

Figure 3 Schematic diagram of security risk model two

图 3 安全风险模型二的形式化表示为一个四元组(S, I, O, T), 其中:

S: 有穷非空状态集{S_0, S_1, S_2, HT_{s0}, HT_{s1}, HT_{s2}, HT_{s3} }, 其中 S_0, S_1, S_2 表示正常状态, HT_{s0}, HT_{s1}, HT_{s2} 为过渡态, 不具有危害, 只有连续依次经过 HT_{s0}, HT_{s1}, HT_{s2} 三种状态, 才能达到安全风险状态 HT_{s3}={$I_{leakage}$, F_{change}, $D_{service}$, $C_{destroy}$, $S_{empower}$},

$I_{leakage}$ 表示信息泄漏安全风险状态, F_{change} 表示功能改变安全风险状态, $D_{service}$ 表示拒绝服务安全风险状态, $C_{destroy}$ 表示破坏芯片安全风险, $S_{empower}$ 表示提升权限安全风险

I: 有穷非空输入集{I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8 , I_9, I_{10}, I_{11}}

O: 有穷非空输出集{ O_1, O_2, O_3, O_4, O_5, O_6, O_7, O_8 , O_9, O_{10}, O_{11}}

T: 表示在输入 I 集中某元素时, S 状态集中某两个状态之间的转换函数

=>: 表示状态转换

状态描述如下:

S_0+ I_1 => HT_{s0}+ O_1 表示当前状态为 S_0, 在输入 I_1 的情况下, 转换到状态 HT_{s0}, 输出为 O_1; 在这种情况下, 如果下一个输入为 I_9, 则又转入到正常状态 S_0;

HT_{s0}+ I_2 => HT_{s1}+ O_2 表示当前状态为 HT_{s0}, 在输入 I_2 的情况下, 转换到状态 HT_{s1}, 输出为 O_2; 在这种情况下, 如果下一个输入为 I_{10}, 则又转入到正常状态 S_0;

HT_{s1}+ I_3 => HT_{s2}+ O_3 表示当前状态为 HT_{s1}, 在输入 I_3 的情况下, 转换到状态 HT_{s2}, 输出为 O_3; 在这种情况下, 如果下一个输入为 I_{11}, 则又转入到正常状态 S_0;

注意: 不存在以下转换关系

HT_{s2}+ I_4 => HT_{s3}+ O_4 表示当前状态为 HT_{s2}, 在输入 I_4 的情况下, 转换到状态 HT_{s3}, 输出为 O_4;

只有在当前状态为 S_0 时, 连续输入 I_1、I_2、I_3、I_4 时, 才从 S_0 => HT_{s0}=> HT_{s1} => HT_{s2}=> HT_{s3}, 这时才达到安全风险状态。

以下四种状态均为正常状态之间的转换关系:

S_0+ I_6 => S_2+ O_6 表示当前状态为 S_0, 在输入 I_6 的情况下, 转换到状态 S_2, 输出为 O_6;

S_2+ I_5 => S_0+ O_5 表示当前状态为 S_2, 在输入 I_5 的情况下, 转换到状态 S_0, 输出为 O_5;

S_2+ I_7 => S_1+ O_7 表示当前状态为 S_2, 在输入 I_7 的情况下, 转换到状态 S_1, 输出为 O_7;

S_1+ I_8 => S_0+ O_8 表示当前状态为 S_1, 在输入 I_8 的情况下, 转换到状态 S_0, 输出为 O_8。

(3) 安全风险模型三

如图 4 所示, 图中带箭头实线表示正常在状态转换, 虚线表示转移到非法状态。

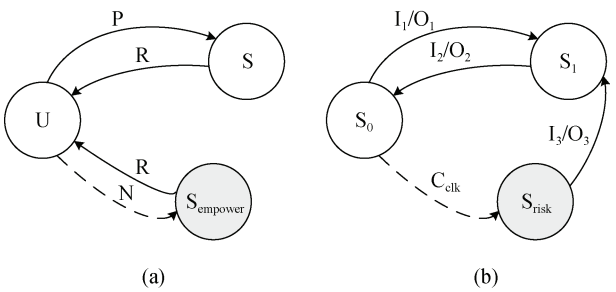

图 4 安全风险模型三

Figure 4 Schematic diagram of security risk model three

图 4(a)表示系统提权模型, 其形式化表示为一个七元组 (U, S, $S_{empower}$, T, P, R, N), 其中:

U: 用户态

S: 正常情况下的系统态或内核态

T: 表示状态之间的转换关系

P: 表示 U 通过系统调用指令达到系统态 S

R: 表示从系统态 S 返回到用户态 U

N: 表示非正常系统调用, 可以通过后门指令或普通用户指令组合实现

=>: 表示状态转换

状态描述如下:

U+P=> S, 表示当前为用户态 U, 在 P 的作用下, 转换到正常系统态 S;

S+R=> U, 表示当前为正常系统 S, 在 R 的作用下, 转换到用户态 U;

U+N=> $S_{empower}$ 表示当前为用户态 U, 在 N 的作用下, 转换到非正常系统态 $S_{empower}$, 达到权限提升的目的;

$S_{empower}$+R=> U, 表示当前状态为 $S_{empower}$, 在 R 的作用下, 转换到用户态 U。

图 4(b)表示内部计数器触发硬件木马模型, 形式化表示为一个四元组(S, I, O, T), 其中:

S: 有穷非空状态集{S_0, S_1, S_{risk}}, 其中 S_0, S_1, S_2 表示正常状态, S_{risk} 表示安全风险状态

I: 有穷非空输入集{I_1, I_2, I_3 , C_{clk}}, 其中 C_{clk} 表示对时钟计数到某值

O: 有穷非空输出集{ O_1, O_2, O_3 }

T: 表示在输入 I 集中某元素时, S 状态集中某两个状态之间的转换函数

=>: 表示状态转换

状态描述如下:

S_0+ I_1 => S_1+ O_1 表示当前状态为 S_0, 在输入 I_1 的情况下, 转换到状态 S_1, 输出为 O_1;

S_1+ I_2 => S_0+ O_2 表示当前状态为 S_1, 在输入 I_2 的情况下, 转换到状态 S_0, 输出为 O_2;

S_0+ C_{risk} => S_{risk} 表示当前状态为 S_0, 在输入时钟计数到 C_{risk} 的情况下, 转换到安全风险状态 S_{risk};

S_{risk}+ I_3 => S_1+ O_3 表示当前状态为 S_{risk}, 在输入 I_3 的情况下, 转换到状态 S_1, 输出为 O_3。

3.4 安全风险验证规则

根据 3.2 节表 1 分析结果, 得知硬件木马直接依附于 I/O 端口的占到 69.2%, 依附于状态机的占到 15.4%, 依附于特殊寄存器(如计数器、特权模式设置寄存器等)占到 15.4%; 再结合 3.3 节相关定义和安全风险模型, 提出以下安全风险验证规则。

(1) 安全风险验证规则一

对于可疑硬件木马位置直接依附位于端口的, 可以扫描出端口相关源代码语句进行验证 [31]。

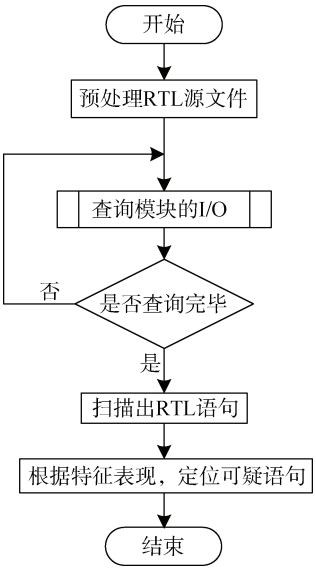

图 5　端口语句扫描流程图

Figure 5　I/O port statement scan flow chart

(2) 安全风险验证规则二

对于可疑硬件木马位置位于状态机的, 可通过相应工具提取出源文件中的状态机和相关输入变量进行验证。

(3) 安全风险验证规则三

对于可疑硬件木马位置位于内部寄存器的, 则重点关注内部计数器寄存器、特殊寄存器(如用户态到系统态转换的寄存器)。

定义 1: 在安全风险规则一、二、三下指导下扫描出的 RTL 源代码为可疑硬件木马语句。

定义 2: 可疑硬件木马语句中含有的输入变量为观测变量。

定义 3: 对于一条可疑硬件木马语句, 如果其作用效果可以造成安全风险中的一种或多种, 那么就说这条语句是硬件木马语句, 否则即为误判。

命题 1: 可疑硬件木马语句中含有的输入变量有助于减少搜索向量空间。

证明: 假设输入变量 V[N-1:0]的长度是 N, 则其可以表示的向量空间范围为 2^N。N 越大, 向量空间范围越大。但是当输入变量中的某一位是确定的, 则其向量空间范围降为 2^{N-1}。如果有两位是确定的, 则其向量空间范围降为 2^{N-2}。依次类推, 当所有位都是确定的, 则向量空间范围变为 1, 即只含有一个向量。

如果扫描出的可疑硬件木马语句中含有输入端口变量, 则根据含有的输入端口变量产生向量。提出如下算法。

3.5　基于安全风险的验证流程

基于安全风险模型和规则, 结合仿真、模型检验等验证方法[32-41], 给出了基于安全风险的验证流程图。

表 5　基于特定位向量产生算法伪代码

Table 5　Vector generation based on special bit location

Algorithm vector_generation

Input: 扫描到的可疑 RTL 语句
Output: 测试向量
1: 初始化: 定义输入变量长度为 N, 所有的输入变量的位均为 X
2: if(可疑 RTL 语句中不含有输入端口变量**)**
根据传统方法产生随机测试向量
else(含有输入变量**)**
{
for(i=0;i<N;i++)
从低位到高位查找包含有哪些位
}
3: 根据 RTL 语句中找出的含有输入变量的位, 将输入变量相应的位确定为 0 和 1, 其他位随机生成值, 则产生相应的测试向量
4: Output 测试向量

流程图主要内容如下:

(1) 模型与规则选取

选取第三方 IP 核 RTL 源代码, 根据结合安全风险模型和规则, 确定待验证的源代码范围。

(2) 验证方法与激励选取

根据选取的验证对象, 确定是采用模拟仿真的方法、还是形式化验证方法(模型检验)。如果是模拟仿真的方式则需要产生激励, 根据搜索到的 RTL 源代码去指导产生相应的激励。

(3) 具体验证

依据上一步骤产生的激励去验证从正常状态到异常状态之间转换是否成立, 如成立, 说明含有硬件木马风险, 否则认为是安全的。

4　实验

实验意义: 根据标准测试集的分析结果, 目前 RTL 级硬件木马主要存在于输入输出端口、状态机和内部寄存器(如计数器、系统提权寄存器), 位于端口的占到 69.2%, 如 AES-T100、AES-T1000、AES-T1300 和 AES-T1400 模块等; 直接依附于状态机的占到 15.4%, 如 RS232-T600、RS232-T700、RS232-T900 和 RS232-T901 模块等; 剩下的 15.4%是依附在内部寄存器, 如 RS232-T200、RS232-T300、AES-T1200、AES-T1500 等模块。因此在进行实验时, 主要针对这三种情况进行实验, 实验具有代表性, 可以覆盖典型的或大部分 RTL 级硬件木马的问题, 对于工程实践应用具有一定的参考价值和借鉴意义。

实验对象: 由于 Trust-Hub 标准测试是有一定局限性(每个测试用例的代码量少, 并且样本也少等)

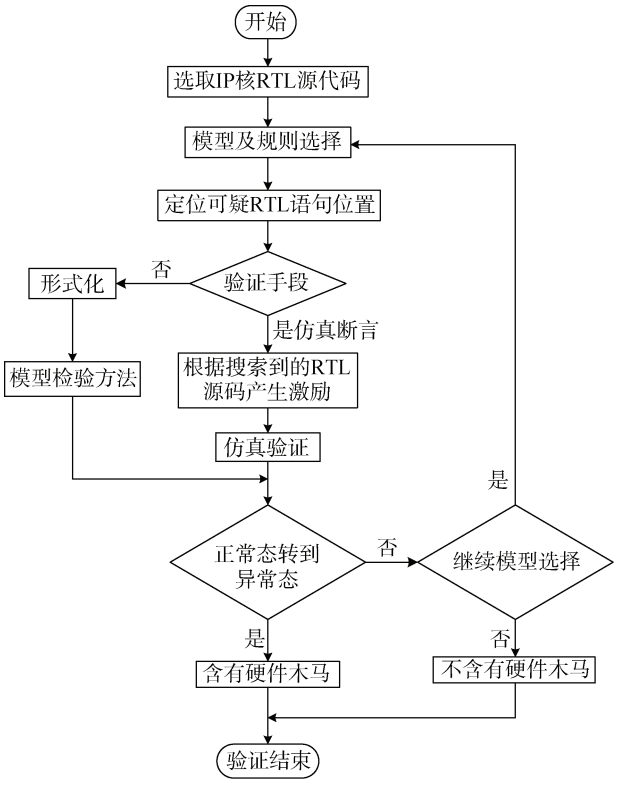

图 6 基于安全风险验证流程图

Figure 6 Verification flowchart based on Security risk

的, 测试集的特点如下: 首先是测试集系列(如 21 个 AES 测试用例或 10 个 RS232 测试用例系列等)案例相似度过高, 硬件木马设计变化较小; 其次是测试集中的每个案例 RTL 源代码量比较小, 行数不多, 代码书写特征变化形式少; 再次是输入输出端口数目较少, 硬件木马的设计位置相对固定。以上这些特点在实验时会导致误报率和漏报率等数据的可信性欠缺, 使得实验数据代表性减弱, 这与工程实际当中的源代码(如代码量大、行数多、文件数量多、代码书写形式丰富等)安全评估工作环境是有区别的, 所以这些特点反映了 Trust-Hub 测试集是有一定局限性的。

正是由于 Trust-Hub 硬件木马集是有一定局限性(代码量少和样本少等)原因, 考虑到工程实际情况, 拟结合常见的开源处理器的 RTL 级源代码, 针对常见的典型的三种安全风险位置, 设计了三个实验作为对应。

实验环境: 操作系统为 win7 professional service pack 1, 实验软件为 modelsim 10.1a, Debussy, SVA 断言, 及模型检测工具 EMBC 等。

4.1 实验一

(1) 实验一针对依附于端口位置硬件木马(安全风险模型一和验证规则一)。

下面实验对象是开源处理器 OR1200 的 ALU 算术逻辑运算单元, 它主要完成或、异或、移位、位扩展、加、减等运算。对于乘除运算, 则有专门的 mult_mac 模块来完成, 它把结果传送给 ALU [35]。OR1200 开源处理器的 ALU 运算单元源文件是 or1200_alu.v。

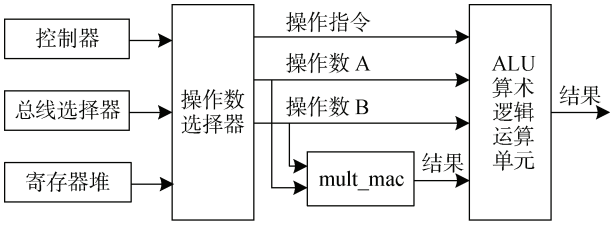

图 7 ALU 算术逻辑运算单元连接示意图

Figure 7 Arithmetic logic unit connection diagram

(2) 在 or1200_alu. v 源文件中加入硬件木马(模仿 Trust-Hub 硬件木马特征书写)。将原来的 result=a^b, 修改为 result=(~(a[31]^b[20]))?a^b:0。该硬件木马的功能是, 将 ALU 算术逻辑运算单元输入的操作数 a 的 a[31]位与输入操作数 b 的 b[20]位相异或, 异或完之后取反, 如果结果是 1, 则硬件木马功能不触发, 输出 a^b 结果正确; 如果是 0, 则触发硬件木马, a^b 结果取 0。

(3) 通过端口扫描出相关可疑硬件木马语句, result=(~(a[31]^b[20]))?a^b:0, 如果按照传统的激励产生方法则需要在 2^{32} 的向量空间内产生数量庞大的测试向量, 而大多数测试激励对于问题的发现并没有意义, 在这里只需要产生关键比特位 a[31]=0, b[20]=1, 或 a[31]=1, b[20]=0 激励向量就可以了, 因此使得测试激励目的性变强, 测试向量空间缩小。

(4) 通过仿真验证容易发现, 当 a[31]=0, b[20]=1, 或 a[31]=1, b[20]=0 时, 是与原功能相违背的, 因此可以说明这条语句是有安全风险的(改变功能)。

4.2 实验二

(1) 实验二主要针对依附于状态机的硬件木马(安全风险模型二和验证规则二)。

实验对象是 Trust-Hub 标准集的 RS232-T901 源代码 u_xmit 模块电路, 提取状态机得到如下图 7 所示。

(2) 该硬件木马的功能是, 当连续输入 8'hAA, 8'h00, 8'h55 和 8'hFF 四个值时, 造成拒绝服务的安全风险。如果按照传统验证方法, 虽然对于 8 比特长的输入变量来说, 只有 256 个向量取值, 完全可以全覆盖验证; 但是遇到图 8 这种情况是难以奏效的, 因为这是多个值的组合, 而这种排列组合的空间是十

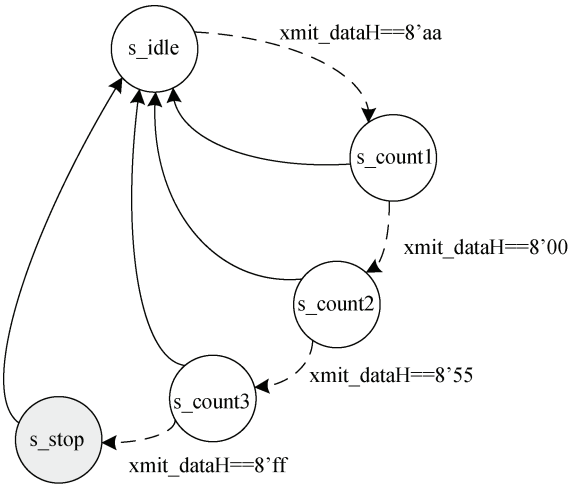

图 8　u_xmit 模块电路 state_DataSend 状态机
Figure 8　state_DataSend state machine of u_xmit

分庞大的。所以采用基于安全风险分析的方法, 提取出相应状态机及有关输入变量取值, 验证时, 主要验证连续输入 8'hAA, 8'h00, 8'h55 和 8'hFF 四个值时的情况, 针对性更强一些。

4.3　实验三

(1) 实验三主要针对依附于特殊寄存器的硬件木马(安全风险模型三和验证规则三)。

OR1200 处理器有两种工作模式, 即用户模式和特权模式。不同的模式, 对特殊寄存器的访问权限不同。OR1200 处理器的特殊寄存器访问指令主要有 l.mfspr 和 l.mtspr 两条指令。其中, l.mfspr 指令用于读取特殊寄存器, l.mtspr 指令用于写入特殊寄存器, l.sys 0x01 是系统调用指令。

对 or1200_ctrl.v 文件中的原代码 sig_syscall <= (id_insn[31:23]=={`OR1200_OR32_XSYNC, 3'b000}) 进行修改为如下所示:

sig_syscall<=(id_insn[31:23]=={'OR1200_OR32_XSYNC,3'b000})||((if_insn[31:26]==6'b111000)&&(id_insn[31:26]==6'b000110))。

or1200_ctrl.v 原代码是未加入硬件木马的, 意思是说遇到系统调用指令 l.sys 时才产生系统调用信号。加入硬件木马时, 则是遇到 l.movhi(操作码为 6'b000110)和 l.add(操作码为 6'b111000)组合指令序列, 考虑到流水线的作用, 当处于译码寄存器 id_insn[31:26] 的是 l.movhi 指令, 取指寄存器 if_insn[31:26]是 l.add 指令, 则触发硬件木马进行系统调用, 以实现提升权限操作。

下边给出了一段设计程序来说明问题。

```
.section .text,"ax"
.org 0x100
.global _start
_start:
l.movhi r0,0   #初始化 r0-r4 寄存器, 全部清零
l.movhi r1,1
l.add r1,r1,r0
l.nop 0x0001
.org 0xc00   #0xc00 就是系统调用入口
l.mfspr r1,r0,0x20
l.mfspr r1,r0,0x40
l.nop 0x0001
```

在以上实验程序中并没有含有系统调用指令 l.sys。在没有硬件木马时, 以上程序不应该产生系统调用操作, 但是在加入硬件木马时, 则会产生与系统调用指令 l.sys 相同的功能, 达到提升权限的目的。

(2) 目前指令集一般都含有上百条、甚至几百条指令, 对一条指令的功能验证是容易的, 但是对于指令集组合所产生的功能进行验证, 这个验证空间是十分庞大的, 为此, 对关键寄存器进行关注是十分必要的。

对于(1)中的这种情况, sig_syscall 是重点关注的寄存器, 设置了如下断言:

@(posedge clk)　$rose(sig_syscall);

来监测系统调用的次数。在没有加入硬件木马时, 断言监测系统调用的次数为 0 次; 在加入硬件木马后, 虽然没有系统调用指令, 但是监测到还是有 1 次系统调用的信号, 这说明是硬件木马触发的。

通过采用安全风险模型和安全风险验证规则, 在以上三个实验基础上进行了拓展, 以开源 CPU 代码 or1200(or1200_alu.v 和 or1200_ctrl.v)、turbo8051(oc8051_alu.v)、M32632(M32632.v)、a-z80(alu.v)的源文件为实验对象(这些源代码更接近于真实情况, 对这些代码仿照标准集木马, 在源代码中插入硬件木马)说明, 如表 6 所示(TPR 表示正检率, FPR 表示误报率, FNR 表示漏报率)。

表 6　基于安全风险验证数据
Table 6　Verification based on Security risk　(%)

文件名称	缩减范围	TPR	FPR	FNR
or1200_alu.v	99.02	85.00	9.67	15.00
or1200_ctrl.v	98.28	90.62	13.92	9.38
oc8051_alu.v	94.69	80.01	9.27	19.99
M32632.v	95.01	85.00	11.32	15.00
alu.v	94.79	88.89	12.80	11.11

4.4　分析与比较

基于安全风险验证可以将评估验证范围缩小, 但是不可避免地存在着误报和漏报的问题。

产生误报的原因是因为在对端口语句搜索的结

果中, 含有和硬件木马接近的表达逻辑, 所以被误识别成硬件木马。例如源文件 or1200_ctrl.v 的误报率比源文件 or1200_alu.v 的高, 主要原因在于 or1200_ctrl.v 是 CPU 的控制器模块, 它里面含有丰富多样的控制信号, 如锁流水线信号 wb_freeze、系统调用信号 sig_syscall、读信号 ex_spr_read、写信号 ex_spr_write, 以上信号都是基本上都是由多个变量的组合逻辑表达式构成, 与硬件木马表现特征近似, 导致误判, oc8051_alu.v、M32632.v、alu.v 也存在同样的问题。产生漏报的原因是主要是对于状态机较多、内部寄存器较多的情况, 安全风险模型和验证规则难以覆盖到所有的状态机和内部寄存器。如 or1200_ctrl.v 文件有数十个状态机和将近四十个内部寄存器, 对于验证而言, 哪些是关键寄存器, 哪些不是关键寄存器, 目前尚欠缺比较明确的划分方式, 本文重点关注计数器和权限提升寄存器, 并且硬件木马采用随机插入方式, 这样导致在验证过程中遗漏相关内部寄存器。

对于 RTL 级硬件木马问题研究, 由于研究人员观察角度不同, 所以采用的方法也不同, 因此, 本文在与其他方法(比较典型的是文献[28]、[29] 、[30])比较时采用了定性分析的方法。

文献[28]通过采用控制流图方法来检测 RTL 级硬件木马, 将 RTL 源代码中的硬件木马抽象成 cheat code、dead machine、ticking timebomb 三类模型, 然后对分析的 RTL 源代码提取出相应的代码段与模型做匹配。这种方法本质是借鉴软件木马匹配检测的方法, 这种方法具有局限性, 主要是因为硬件木马的设计与软件木马具有很大的区别。例如, 软件木马是一个相对完整的程序, 它由一个或多个源文件构成, 通过对整个程序或关键文件进行哈希匹配运算, 即可得到该软件木马的特征。相反地, 硬件木马在 RTL 级上的设计不需要像软件木马一样, 它只需在源文件中添加一条语句(甚至只修改原文件中一条语句的参数), 或几条语句就可以完成植入任务, 进一步而言, 只要修改硬件描述语言中的参数, 而不修改语句的表达式结构, 就在另一个源文件中完成另外一个硬件木马的植入。因此说可以吸取软件木马特征匹配方法的思想, 但是具体到硬件木马检测和验证时, 应该重点关注其结构表达特征。再者, 文献[28]的提取算法(Extraction Algorithm)和检测算法(Detection Algorithm)相对比较复杂, 在提取算法中, 需要提取和生成 RTL 源代码中的控制流图, 并且还要生成硬件木马特征库, 以便于比较检测, 这就涉及分析RTL源代码的逻辑功能, 如if-then(-else)结构, 这将会导致问题复杂化, 不同的 RTL 源代码的控制流图是有很大区别的, 不可避免地导致计算量和耗费开销增加。而本文方法只是将 RTL 源代码作为文本来进行分析, 从文本语义角度进行检测, 对于不同的 RTL 源代码分析方法和复杂程序是一样的, 分析和检测算法相对简单, 适用范围相对广泛, 这与工程实际的要求是相吻合的。

文献[29]主要采用形式化方法, 即模型检验来检测硬件木马, 关于模型检验方法的使用, 会不可避免的遇到状态空间爆炸问题, 虽然对于某些特定问题可以比较好的解决, 但是对于实际工程中文件数量众多、变量众多、状态空间多的情况则难以适应。如对实验二的依附于状态机的硬件木马的检测问题, 及实验三中的组合指令实现系统提权的问题, 则采用模型检测方法难以奏效。文献[29]主要用来检测信息泄漏类型的硬件木马, 没有提及到功能改变、芯片破坏、拒绝服务和系统提权等危害类型。进一步而言, 模型检验方法本质上也是标准的芯片验证方法, 主要从正面验证芯片功能的正确性, 但是, 前而已经提及, 攻击者设计的木马类型多样、隐蔽性强, 一般很难通过常规的、传统的验证方法发现。为此, 本文本质上是利用了硬件木马的分布概率规律, 包括危害类型、安全风险位置、及危害类型和安全风险之间的关系等, 通过利用这些先验知识, 来减少验证的盲目性, 提高检测和验证硬件木马的成功率。

文献[30]设计的硬件木马是通过破坏 RISC 处理器的栈指针(stack pointer)来达到攻击的目的。正常情况下, 主要是 CALL 调用指令、RET 指令和 RESET 指令来修改栈指针, 该文献设计的硬件木马则是, 当指令寄存器的值在 0x4～0xB 范围内, 并且这种情况出现 100 次, 则会触发硬件木马。如果从本文的角度来分析, 则属于第三种安全风险类型, 并且与实验三很类似。在实验三中已经提出, 由于指令集一般有上百条指令甚至几百条指令, 随机组合的空间更为庞大, 那么在文献[30]中也存在同样的问题, 所以按照传统的验证方式是难以奏效的。文献[30]主要是针对一种修改数据的类型的木马, 需要与指令集、软件编程仿真相结合, 相对而言, 硬件木马表现形式比较单一, 危害形式只有一种, 即改变功能, 与工程实际情况差别较大。如果利用本文安全风险模型三, 把栈指针寄存器和指令寄存器当作关键的验证寄存器, 通过对 RTL 源代码的分析和验证, 则评估效率和验证效率应该会有所提高。

总之, 本文的研究主要是面向源代码安全性评估的工作, RTL 级源文件数量比较多, 在这种情况下,

如果采用传统的验证方法难以找到重点, 再加上即使源文件中真得含有安全风险, 一般设计也比较隐蔽, 如果对源代码不加以区分, 对代码做安全评估是有困难的。

因此, 本文在分析硬件木马标准集的基础上, 给出相关定义, 提炼出安全风险模型和安全风险位置, 制定出相应的安全风险验证规则, 在以上先验知识的基础上, 结合 RTL 源文件的语义分析, 避开传统直接验证的方式方法, 这样做可以尽量避免状态空间爆炸、向量空间膨胀等问题, 降低验证人力、物力、财力和时间成本。与其他方法相比较而言, 适合于对外来第三方 IP 核源代码做前期安全性评估, 有助于将可疑代码范围缩小, 降低盲目性, 增强针对性, 提高评估效率。当然, 本文的研究还存在精细程度和深入程度还不够, 如对安全风险规则三当中的重点关注寄存器确定问题, 需要结合相关研究方法进一步提高。

5 总结与展望

当前, 关于第三方 IP 核源代码安全评估问题, 即是否含有硬件木马, 如何进行验证, 是新的研究点, 同时也是难点。

针对目前 RTL 级源代码安全评估及硬件木马验证研究缺少安全风险模型和指导规则, 在对硬件木马集进行分析的基础上, 给出了相关模型、定义和安全风险验证规则。

文章试图在面临 RTL 源代码安全评估时, 提出了基于安全风险模型和安全风险规则的验证评估, 减少验证盲目性, 缩小可疑代码范围, 提高工作效率。实验证明, 安全风险模型和规则对 RTL 级硬件木马验证具有一定的效果。

致 谢 本文是在信息工程研究所第五研究室多位老师指导下完成的, 在此表示诚挚感谢。另外, 对于审稿专家的意见和建议, 表示衷心感谢!

参考文献

[1] Jiang P, Li D J. Information confrontation[M]. Beijing: Tsinghua University Press, 2007: 1-11.
(蒋平, 李冬静. 信息对抗[M]. 北京: 清华大学出版社, 2007: 1-11.)

[2] McAfee Corp. New Paradigm Shift: Comprehensive Security beyond the Operating System. *White paper*, 2012, 2-9.

[3] Kauer B. OSLO: Improving the Security of Trusted Computing[J]. *16th USENIX Security Symposium*, 2007: 229-237.

[4] A.Sanjaya, Hardware Assisted Security: Anticipating Digital Threat and Challenges[C]. *FIRST TC 2012 IDF*, 2012: 1-10.

[5] Agrawal D, Baktir S, Karakoyunlu D, et al. Trojan Detection Using IC Fingerprinting[C]. *2007 IEEE Symposium on Security and Privacy*, 2007: 296-310.

[6] Di J, Smith S. A Hardware Threat Modeling Concept for Trustable Integrated Circuits[C]. *2007 IEEE Region 5 Technical Conference*, 2007: 354-357.

[7] Wang X X, Tehranipoor M, Plusquellic J. Detecting Malicious Inclusions in Secure Hardware: Challenges and Solutions[C]. *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008: 15-19.

[8] Hu W, Oberg J, Irturk A, et al. On the Complexity of Generating Gate Level Information Flow Tracking Logic[J]. *IEEE Transactions on Information Forensics and Security*, 2012, 7(3): 1067-1080.

[9] Hu W, Oberg J, Irturk A, et al. Theoretical Fundamentals of Gate Level Information Flow Tracking[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2011, 30(8): 1128-1140.

[10] Zhou B, Zhang W, Thambipillai S, et al. Cost-Efficient Acceleration of Hardware Trojan Detection through Fan-out Cone Analysis and Weighted Random Pattern Technique[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016, 35(5): 792-805.

[11] Liu C, Cronin P, Yang C M. A Mutual Auditing Framework to Protect IoT Against Hardware Trojans[C]. *2016 21st Asia and South Pacific Design Automation Conference*, 2016: 69-74.

[12] Dupuis S, Ba P S, Di Natale G, et al. A Novel Hardware Logic Encryption Technique for Thwarting Illegal Overproduction and Hardware Trojans[C]. *2014 IEEE 20th International On-Line Testing Symposium*, 2014: 49-54.

[13] Zhou E R, Li S Q, Zhao Z X, et al. Nonlinear Analysis for Hardware Trojan Detection[C]. *2015 IEEE International Conference on Signal Processing*, *Communications and Computing*, 2015: 1-4.

[14] L.W Wang, H.W. Luo and R.H. Yao, A hardware Trojan detection method based on bypass analysis[J] .*Journal of South China University of Technology (NATURAL SCIENCE EDITION)*, 2012, 40(6): 9.
(王力纬, 罗宏伟, 姚若河.基于旁路分析的硬件木马检测方法[J] .*华南理工大学学报(自然科学版)*, 2012, 40(6): 9.)

[15] Kumar P, Srinivasan R. Detection of Hardware Trojan in SEA Using Path Delay[C]. *2014 IEEE Students' Conference on Electrical*, *Electronics and Computer Science*, 2014: 1-6.

[16] Moore T D, Jarvis J L. Failure Analysis and Stress Simulation in Small Multichip BGAs[J]. *IEEE Transactions on Advanced Packaging*, 2001, 24(2): 216-223.

[17] Chakraborty R S, Wolff F, Paul S, et al. MERO: A Statistical Approach for Hardware Trojan Detection[C]. *The 11th International Workshop on Cryptographic Hardware and Embedded Systems*, 2009: 396-410.

[18] Cakır B, Malik S. Hardware Trojan Detection for Gate-Level ICs Using Signal Correlation Based Clustering[C]. *2015 Design*, *Automation & Test in Europe Conference & Exhibition*, 2015: 471-476.

[19] Bao C X, Forte D, Srivastava A. On Application of One-Class SVM to Reverse Engineering-Based Hardware Trojan Detection[C]. *Fifteenth International Symposium on Quality Electronic Design*, 2014: 47-54.

[20] Hasegawa K, Oya M, Yanagisawa M, et al. Hardware Trojans Classification for Gate-Level Netlists Based on Machine Learning[C]. *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design*, 2016: 203-206.

[21] Wang C G, Cai Y C, Zhou Q. Automatic Security Property Generation for Detecting Information-Leaking Hardware Trojans[C]. *2017 IEEE International Conference on Computer Design*, 2017: 321-328.

[22] Lodhi F K, Abbasi I, Khalid F, et al. A Self-Learning Framework to Detect the Intruded Integrated Circuits[C]. *2016 IEEE International Symposium on Circuits and Systems*, 2016: 1702-1705.

[23] Dai Y Y, Braytont R K. Circuit Recognition with Deep Learning[C]. *2017 IEEE International Symposium on Hardware Oriented Security and Trust*, 2017: 162.

[24] Zareen F, Karam R. Detecting RTL Trojans Using Artificial Immune Systems and High Level Behavior Classification[C]. *2018 Asian Hardware Oriented Security and Trust Symposium*, 2019: 68-73.

[25] Zhang J, Xu Q. On Hardware Trojan Design and Implementation at Register-Transfer Level[C]. *2013 IEEE International Symposium on Hardware-Oriented Security and Trust*, 2013: 107-112.

[26] Banga M, Hsiao M S. Trusted RTL: Trojan Detection Methodology in Pre-Silicon Designs[C]. *2010 IEEE International Symposium on Hardware-Oriented Security and Trust*, 2010: 56-59.

[27] Piccolboni L, Menon A, Pravadelli G. Efficient Control-Flow Subgraph Matching for Detecting Hardware Trojans in RTL Models[J]. *ACM Transactions on Embedded Computing Systems*, 2017, 16(5s): 1-19.

[28] Rajendran J, Dhandayuthapany A M, Vedula V, et al. Formal Security Verification of Third Party Intellectual Property Cores for Information Leakage[C]. *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems*, 2016: 547-552.

[29] Rajendran J, Vedula V, Karri R. Detecting Malicious Modifications of Data in Third-Party Intellectual Property Cores[C]. *2015 52nd ACM/EDAC/IEEE Design Automation Conference*, 2015: 1-6.

[30] J.F. Zhao, G. Shi. Cheng. Research on RTL level Hardware Trojan. *Journal of Cyber Security*, 2020.
(赵剑锋, 史岗.RTL 级硬件木马问题研究[J]. *信息安全学报*,2020.)

[31] Cheng X, Li L, Cheng W. A Detection Method of Hardware Trojans Based on RTL[J]. *Microelectronics & Computer*, 2017, 34(3): 56-60.
(成祥, 李磊, 程伟. 基于 RTL 级硬件木马的检测方法[J]. *微电子学与计算机*, 2017, 34(3): 56-60.)

[32] Zhao J F, Shi G. A Survey on the Studies of Hardware Trojan[J]. *Journal of Cyber Security*, 2017, 2(1): 74-90.
(赵剑锋, 史岗. 硬件木马研究动态综述[J]. *信息安全学报*, 2017, 2(1): 74-90.)

[33] Clarke E M. Automatic Verification of Finite-State Concurrent Systems[C]. *Proceedings Ninth Annual IEEE Symposium on Logic in Computer Science*, 2002: 126.

[34] Allen Emerson E, Halpern J Y. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time[J]. *Journal of Computer and System Sciences*, 1985, 30(1): 1-24.

[35] Ehrhard T. Linear logic in computer science[M]. Cambridge: Cambridge University Press, 2004

[36] Lei S L. Step by step “core”: Analysis of internal design of soft core processor[M]. Beijing: Publishing House of Electronics Industry, 2013.
(雷思磊. 步步惊“芯”: 软核处理器内部设计分析[M]. 北京: 电子工业出版社, 2013.)

[37] S. Vijayaraghavan and M. Ramanathan. A practical Guide for System Verilog Assertions[M]. *Springer Ltd publishers*, 2005, 1-100.

[38] Damjan Lampret. OpenRISC 1200 IP Core Specification. http://www.opencore.org. 2017.

[39] OpenRISC 1000 Architecture Manual. http://www.opencore.org. 2017.

[40] Yeung P, Larsen K. Practical Assertion-Based Formal Verification for SoC Designs[C]. *2005 International Symposium on System-on-Chip*, 2006: 58-61.

[41] Karunakaran D K, Mohankumar N. Malicious Combinational Hardware Trojan Detection by Gate Level Characterization in 90nm Technology[C]. *Fifth International Conference on Computing, Communications and Networking Technologies*, 2014: 1-7.

赵剑锋 于 2006 年在北京理工大学通信与信息系统专业获得硕士学位。现在中国科学院信息工程研究所计算机系统结构专业攻读博士学位。研究领域为计算机系统安全。研究兴趣包括: 架构安全。Email: zhaojianfeng@iie.ac.cn

史岗 于 2004 年在中国科学院计算技术研究所获得博士学位。现任中国科学院信息工程研究所第五研究室高级工程师。研究领域为计算机系统安全。研究兴趣包括: 嵌入式系统、信息安全。Email: shigang@iie.ac.cn