

RTL 级硬件木马问题研究

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。信息安全可以分为两类: 软件安全和硬件安全。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, 同样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。在上述背景并结合国内对芯片 RTL 源代码安全风险评估的实际需求展开了相关工作, 围绕 RTL 源代码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属性的问题, 给出硬件木马的属性描述形式, 在形成硬件木马属性的基础上, 以模块端口信号为源, 提出了一种基于信号流向的多叉树分层递归搜索方法, 实验结果表明, 该方法对于依附在端口上的硬件木马的检测是有效的。

关键词 芯片; RTL 级硬件木马; 属性描述; 搜索方法

中图分类号 TP309.2 DOI 号 10.19363/J.cnki.cn10-1380/tn.2023.07.10

Research on RTL level hardware Trojan

Zhao Jianfeng^{1,2}, Shi Gang²

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. Information security can be divided into two categories, Software security and hardware security. 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 is a hot topic in academic circles at home and abroad. Especially in the design stage, the problem of hardware Trojan 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, and a clustering analysis method based on unsupervised learning is proposed to form the definition of hardware Trojan's attributes, i.e. intrinsic attributes, including structural features, and external attributes, including triggering mode, location and harmful results. It provides a basis for the detection and verification of hardware Trojan. On the basis of Forming Hardware Trojan Attribute, a hierarchical recursive search method based on multi-tree of signal flow direction is proposed. The experimental results show that the method is effective for the detection of hardware Trojan attached to ports.

Key words chip; RTL level hardware trojan; attribute description; search method

1 引言

信息时代的来临使得信息安全日益重要。信息安全可以分为软件安全和硬件安全两个方面^[1-6]。攻

击方为了获取想要的信息, 可以说是不择手段。人们比较熟悉的, 如软件病毒、木马、蠕虫等已经对全球的政治、经济、军事等领域造成了难以估量的损失, 在此不需赘述。事实上, 除了软件方面的安全威胁,

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: 2020-03-07; 修改日期: 2020-04-20; 定稿日期: 2023-02-16

硬件同样存在着安全问题, 尤其是对于作为信息处理核心的处理器而言, 更是让人担忧。然而, 硬件木马就是对处理器芯片构成巨大危害的元凶之一。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, 如果被触发, 更会对系统造成难以估计的损失和灾害。

IBM 沃森研究中心和伍斯特理工学院在 2007 年第一次联合提出了硬件木马的概念^[7]。后来经过 D. Jia^[8], X. Wang 和 M. Tehranipoor, Yier Jin^[9]以及国内硬件木马研究人员^[10-11]的研究和扩展, 虽然还未形成统一的概念, 但是总结而言: 硬件木马主要是指在集成电路芯片中故意植入特殊功能的逻辑电路, 在一定的条件下触发该逻辑功能, 以达到攻击利用或破坏芯片的目的。硬件木马可以使得芯片系统功能改变, 造成重要信息泄漏, 或致使系统无法运行等。

硬件木马是伴随着芯片的设计、生产及制造过程而被植入的。一块芯片的诞生, 需要经过设计、综合、仿真、布局、布线、加工、测试、封装和组装等各个环节, 因此, 硬件木马有可能在其中任一个环节中被植入。

硬件木马已经成为国内外学术界研究的热点课题, 尤其是关于硬件木马的检测问题, 更是热点, 同时也是难点。

文章围绕 RTL 源代码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属性的问题, 给出硬件木马的属性描述形式, 在形成硬件木马属性的基础上, 以模块端口信号为源, 提出了一种基于信号流向的多叉树分层递归搜索方法, 实验结果表明, 该方法对于依附在端口上的硬件木马的检测是有效的。

文章主要包括以下部分: 第 2 小节, 给出当前常见的硬件木马研究分析方法; 第 3 小节, 提出威胁模型, 在硬件木马集的基础上抽象出硬件木马属性描述形式, 并对硬件木马集做了分析, 以此来指导检测工作; 第 4 小节, 提出基于信号流向的多叉树搜索分析方法; 第 5 小节给出实验。最后是总结部分, 给出结论, 展望了下一步的研究。

2 相关研究方法

硬件木马研究方法主要分为破坏式和非破坏式两大类。

破坏式方法主要是物理检测技术。物理检测技术^[15]一般对芯片先进行开封, 进而对芯片一层一层地扫描, 依据扫描图像恢复出设计, 最后与原始版

图比较来检测其中是否含有硬件木马。物理检测技术属于破坏式检测方法, 本质上是芯片逆向工程分析方法。如文献[12]和[13]采用的就是物理检测技术。

非破坏式方法分为侵入式(如 DFS, Design for Security 和内建自测技术)与非侵入式(如功能测试技术和旁路分析技术)两种。

侵入式方法中比较典型的是 DFS 安全性设计方法, 它最大限度地减少芯片脆弱性, 从芯片的机密性, 完整性, 可用性、访问控制、认证、防抵赖和隐私保护等方面来设计芯片。

内建自测技术, 其简称 BIST(Built-In Self-Test), 实际上是在一个芯片内部加外额外的功能模块^[14]。这些额外的功能模块可以监测芯片内部的信号状态。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。

非侵入方法主要包含功能测试、旁路分析、机器学习、门级信息流分析及形式化方法等。

功能测试技术是在 ATPG(自动测试图形生成)基础上完成硬件木马的检测^[15]。这种技术的思路是: 通过对芯片的输入端口施加激励变量, 然后观测输出端口信号值, 如果输出值与预计值不同, 则推断芯片中可能含有硬件木马。

旁路分析技术^[16]是基于这样一个事实: 芯片工作时会产生各种旁路信号, 如热信号、电磁辐射信号、功耗信号、以及电路延时信号等。如果芯片中含有硬件木马, 那么当硬件木马运行时, 会对芯片的旁路信号造成影响, 通过观测某种旁路信号的变化, 则可能判断出芯片中是否含有硬件木马。如文献[17]分析的旁路信号是电流。文献[18]分析的旁路信号是瞬时功耗。文献[19]分析的旁路信号是路径时延。

近几年来, 在硬件木马检测研究方法提出了不少新的方法, 引入了如统计学习方法^[20]、SVM(Support Vector Machine)^[21]、机器学习和自学习^[22-24]、深度学习^[25]、多层神经网络和人工免疫系统及行为分类^[26]等多种方法。以文献[20]为例, 它通过计算 Gate-Level 电路输入输出之间的相关值来对硬件木马电路进行区分, 因为正常情况下, 输入输出的相关值要大, 而硬件木马输入端低概率触发导致一般情况下输出值与硬件木马电路几乎不相关, 基于这个原理, 可以把整个电路的 Gate-Level Netlist 的各级输入输出的相关性做上标记, 通过基于密度聚类分析的方法找到硬件木马电路。

门级信息流方法采用格模型(Lattice model)来描述信息流信道与策略。信息流策略采用 $L=(SC, \sqsubseteq)$ 安

全格进行描述, SC 是代表客体安全类集合, \sqsubseteq 是定义在 SC 上的偏序关系, 这种偏序关系限定了不同安全类别之间的数据流向, 规定数据只能在同一个安全类之内, 或从低级别安全类向更高级别安全类流动。采用定义 $L: O \rightarrow SC$ 表示返回某对象 O 的安全类, 例如 $L(a) \sqsubseteq L(b)$ 表示从 a 到 b 流向是安全的。文献[10-11]采用门级信息流分析方法, 该方法对每个数据位分配一个标签(代表此数据单位的属性, 如正确/错误), 采用标签传播策略, 依据操作类型及操作数的标签来确定运算结果的标签, 通过观测运算结果的标签来确定输出结果的属性。

形式化方法是基于数学的方法, 用以描述目标系统性质, 通过严格的数学符号和相应法则, 对系统的结构和行为进行高效简明的描述、分析、推理和演算, 为系统属性的说明、开发及验证设计了框架, 可有助于发现目标系统需求设计中的不一致性、不完备性等情况。形式化方法主要包括等价性验证、模型检验和定理证明三种方式。

当前方法存在的主要问题有:

- (1) 缺乏硬件木马威胁模型和属性的描述;
- (2) 破坏式检测方法需要高精尖的设备, 人力、物力、财力投入巨大, 对一般规模较小的、简单的芯片有一定作用, 对于类似处理器这样比较复杂的芯片, 则难以胜任;
- (3) 非破坏式方法中比较典型的逻辑功能测试方法从正向来验证芯片的逻辑功能是否满足要求, 这种方法主要目的是验证芯片是否满足设计功能, 而硬件木马触发概率极低(上百亿分之一的概率), 从测试向量角度去查找木马的话, 空间过于庞大, 测试时间长;
- (4) DFS 安全设计方法和内建自测方法都是在芯片中加入额外的电路, 以期达到对芯片保护的目的, 这会产生额外的硬件开销;
- (5) 门级信息流的方法需要将 RTL 源代码转换成门级网表, 并且需要对门级网表中加入相应的门级电路, 一个与门需要加入 3 个相应的门级电路作为检测的前提, 使其逻辑锥更为庞大, 同样存在着空间过大的问题;
- (6) 机器学习的方法, 与门级信息流类似, 也是着重分析门级网表层次的电路, 所以与之存在相同的问题。

基于以上存在的问题, 试图给出硬件木马的描述形式, 从 RTL 源代码语义角度出发, 提出基于信号流向的多叉树分层搜索方法来分析 RTL 源文件中的疑似硬件木马语句。

3 硬件木马属性描述形式

3.1 威胁模型

当前主流芯片设计过程中, 除了自主开发外, 会集成大量的第三方的 IP 核, 包括含 RTL 源码的 IP 核, 所以发现并验证 RTL 硬件木马具有重要的工程意义。为了明确本文的研究目标和评价方法的有效性, 本文基于以下威胁模型(Threat Model)开展研究。

威胁是芯片安全性可能被破坏的潜在的行为或事件, 它不是必然发生, 只是存在这种可能性。

威胁模型包含威胁的来源、威胁的效果、威胁的对象三部分。威胁来源主要是自写 RTL 代码、第三方 IP 核代码、库单元、EDA 工具等; 威胁的效果主要是拒绝服务、系统提权、功能改变、信息泄漏、破坏芯片等; 威胁的对象是芯片的设计生产的各个阶段, 包括算法模型、RTL 设计、门级网表、电路版图、封装制造等。

对于芯片而言, 可以画出如图 1 所示的威胁模型示意图。

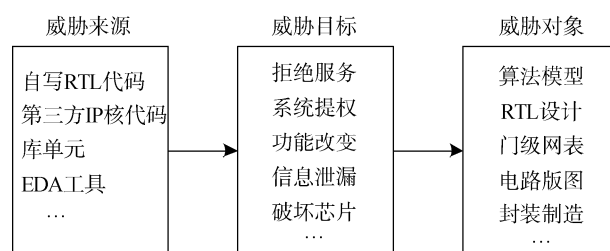


图 1 威胁模型示意图
Figure 1 Treat Model Diagram

文中研究的威胁来源是第三方 IP 核代码, 威胁效果包括拒绝服务, 系统提权, 功能改变, 信息泄漏, 威胁目标是 RTL 设计阶段。即研究的前提是: 假设第三方 IP 核 RTL 源代码是不安全的, 它有可能在 RTL 设计阶段, 植入硬件木马, 以此达到拒绝服务、系统提权、功能改变或信息泄漏的攻击效果。其他的威胁来源、威胁目标和威胁对象不在研究范围之内, 或者说假定它们都是没有问题的。

3.2 硬件木马与软件木马分析方法的不同

要对以 RTL 源代码的形式存在硬件木马进行分析, 首先就要知道硬件木马在源代码中的表现形式, 具体行为以及产生的后果。

由于目前还没有明确提出在 RTL 源代码基础上的硬件木马一般属性, 因此, 本文希望借鉴形式化描述的方法, 来构建 RTL 硬件木马集的描述, 并给出一般属性定义。在分析检测方面借鉴了软件木马处理的思想。软件特征匹配方法主要包括 HASH

匹配、文件内容匹配、二进制代码匹配等,但这些匹配方式不能直接拿来用到 RTL 源代码上。主要原因在于:

软件木马是一个完整的程序,该程序由一个或多个源文件构成,每个源文件由一个或多个函数构成,每个函数由若干程序设计语言的语句组成。软件木马的功能需要整个程序中的每一条语句按既定设计运行,才能达到预期目的。通过 HASH 匹配或其他匹配方式,对整个程序,或其关键文件进行运算,可以得到该软件木马的特征。当软件木马在网络不同的主机上运行时,其程序代码是一样的,语句运行顺序是一样的,危害行为也是一样的,通过直接匹配可以进行检测和查杀。软件木马具有传染性,可以从一个主机传染到另外一个主机,但是其文本特征在短期内是稳定的,所以通过直接 HASH 匹配、文本匹配或二进制代码匹配是可行的。

然而,硬件木马却不一样。它在 RTL 级上的设计不需要像软件木马一样,由多个源文件,或许多条描述语句来完成,相反地,它只需要在整个 RTL 源文件中添加一条语句(甚至只需要改变原文件中某条语句的参数或逻辑表达结构),或几条语句就可以完成植入,并且用同样的方法可以在不同的芯片 RTL 源文件中设计出相同功能的硬件木马,只需要改变硬件描述语言中的参数,而不修改语句的表达式结构,就可以完成另外一个木马的植入。再者,硬件木马不具有传染性,一款芯片中的 RTL 硬件木马只对这款芯片有影响,不会传染到其他芯片设计中。如果直接对 RTL 源文件进行 HASH 运算,或直接字符串、二进制串匹配是难以发现的,所以说可以借鉴软件木马特征匹配的思想,但是具体到硬件木马在 RTL 级上的表现时,应该关注其结构表达特征,如是组合逻辑的还是时序逻辑的,这是由硬件木马本身是数字电路的本质决定的。

另外,硬件木马的触发概率极低(几十亿甚至几百亿分之一的概率),而软件木马一般存在于网络上的许多主机上,它触发运行的概率要大得多。

总之,可以借鉴软件木马检测的思想,但不能直接照搬软件木马的检测分析方法。

文中所讨论的 RTL 硬件木马主要是指数字硬件木马,它属于数字电路的范畴,由组合逻辑(常见的如多路器、数据通路开关、加法器和乘法器等)或时序逻辑(常见的有计数器、数据流动控制逻辑、运算控制逻辑、指令分析和操作控制逻辑等)构成的。本文试图从 RTL 源代码字符语义角度,通过分析 RTL 源代码硬件木马标准集,分析 RTL 源码的一般属性。

3.3 基本定义

Trust-Hub 含有不同模块硬件木马样本,共有 52 个硬件木马实例,其中 AES 加密模块实例有 21 个,RS232 模块有 10 个,MC8051 模块有 7 个,PIC16F84 模块有 4 个,RSA 模块有 4 个,B19 模块有 3 个,wb_conmax 有 2 个,memctrl 模块有 1 个。

Trust-Hub 硬件木马集是研究硬件木马检测和验证的样本,通过对硬件样本集的分析来抽象硬件木马集的一般属性,通过分析硬件木马的一般属性来指导 RTL 级硬件木马的检测与验证工作。

定义 1: HT 是一个 RTL 级硬件木马的非空集合 $HT = \{ht_1, ht_2, ht_3, \dots, ht_n\}$ 。

定义 2: P_i 表示集合 X 中第 i 个元素出现的概率,则由概率论基础知识可知, $0 \leq P_i \leq 1$, 其中, $1 \leq i \leq |X|$, 其中 $|X|$ 表示取集合 X 元素的个数。

定义 3: 对于集合 HT 中的每一个元素,定义一个四元组来描述集合 HT 中的第 i 个元素的属性 $A_i = \langle F(ht_i), L(ht_i), T(ht_i), R(ht_i) \rangle$, $1 \leq i \leq n$, 其中:

F: 是一个 RTL 硬件木马特征非空集合, $F = \{F(ht_i)\}$, 这个特征集合元素的个数不一定与 HT 集合元素个数相同,即 $1 \leq |F| \leq |HT|$, $|F|$ 表示取集合的元素个数, ht_i 与 ht_j ($i \neq j$) 是 HT 集合中不同的硬件木马元素,但是可以拥有相似的表述特征。

L: 是一个 RTL 硬件木马所处位置的非空集合, $L = \{L(ht_i)\}$, 这个所处位置集合元素的个数不一定与 HT 集合元素个数相同,即 $1 \leq |L| \leq |HT|$, ht_i 与 ht_j ($i \neq j$) 是 HT 集合中不同的硬件木马元素,但是可以拥有相似的位置。

T: 是一个 RTL 硬件木马触发方式的非空集合, $T = \{T(ht_i)\}$, 这个触发方式集合元素的个数不一定与 HT 集合元素个数相同,即 $1 \leq |T| \leq |HT|$, ht_i 与 ht_j ($i \neq j$) 是 HT 集合中不同的硬件木马元素,但是可以拥有相似的触发方式。

R: 是一个 RTL 硬件木马危害结果的非空集合, $R = \{R(ht_i)\}$, 这个危害结果集合元素的个数不一定与 HT 集合元素个数相同,即 $1 \leq |R| \leq |HT|$, ht_i 与 ht_j ($i \neq j$) 是 HT 集合中不同的硬件木马元素,但是可以拥有相似的危害结果。

将硬件木马的逻辑表述特征 F 定义为硬件木马在 RTL 级别上的内在属性,指导检测工作。

将硬件木马的位置属性 L、触发属性 T、危害结果属性 R 定义为硬件木马在 RTL 级别上的外在属性,用于指导验证工作。

定义 4: 硬件木马集的分析是,对于定义 3 中的集合 $\{F, L, T, R\}$ 某个元素,本质上也是一个集合设

为 X , 集合 Y 是 X 的某一个非空子集, 满足:

$$Y \subset X, \text{ 且 } Y \neq \emptyset$$

则分析就是满足如下条件的得到的集合:

(1) $Y_1 \cup Y_2 \cup \dots \cup Y_k = X$, 其中 K 代表集合 X 分类结果为 K 个。

(2) $Y_i \cap Y_j = \emptyset (i \neq j)$, 表示不同分类之间无交集。

定义 5:

定义一个 HTKripke 结构为一个七元组。

$HTKripke = \langle S_T, S_F, S_0, R_T, R_F, AP, L \rangle$, 其中:

S_T 是合法状态的有限集合;

S_F 是非法状态的有限集合, 即处于硬件木马工作状态状态的集合;

$S_0 \subseteq (S_T \cup S_F)$ 是初始状态集合;

$R_T \subseteq S_T \times S_T$ 是转移关系;

$R_F \subseteq S_T \times S_F$ 是从合法状态转移到硬件木马工作状态的转移关系;

AP 是所有原子命题和它们的否定命题的集合;

$L: (S_T \cup S_F) \rightarrow 2^{AP}$ 是标记函数, 该函数把状态 $s \in (S_T \cup S_F)$ 映射为在 $(S_T \cup S_F)$ 中为真的原子命题集合, 该集合是 AP 的一个子集。对于一个状态没有包含的原子命题在该状态的值不确定。

其中 $S_T = \{s_{T0}, s_{T1}, s_{T2}\}$, $S_F = \{s_{F0}, s_{F1}, s_{F2}\}$, $S_{T0} = \{s_{T0}\}$, $AP = \{a, b, c, d, e, f, \neg a, \neg b, \neg c, \neg d, \neg e, \neg f\}$, $R_T = \{ \langle s_{T0}, s_{T1} \rangle, \langle s_{T2}, s_{T0} \rangle, \langle s_{T0}, s_{T1} \rangle, \langle s_{T1}, s_{T2} \rangle \}$, $R_F = \{ \langle s_{T0}, s_{F0} \rangle, \langle s_{T1}, s_{F0} \rangle, \langle s_{T2}, s_{F0} \rangle, \langle s_{T0}, s_{F1} \rangle, \langle s_{T1}, s_{F1} \rangle, \langle s_{T2}, s_{F1} \rangle, \langle s_{T0}, s_{F2} \rangle, \langle s_{T1}, s_{F2} \rangle, \langle s_{T2}, s_{F2} \rangle \}$, $L(s_{T0}) = \{a, b\}$, $L(s_{T1}) = \{a, c\}$, $L(s_{T2}) = \{b, \neg c\}$, $L(s_{F0}) = \{d, e\}$, $L(s_{F1}) = \{d, f\}$, $L(s_{F2}) = \{e, \neg f\}$ 。图 2 中带箭头实线表示合法转移, 虚线表示转移到硬件木马工作状态。

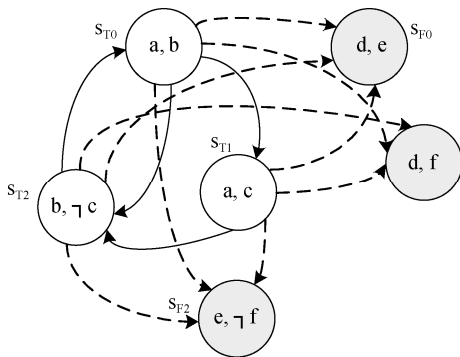


图 2 HTKripke 结构示意图

Figure 2 Schematic diagram of HTKripke

对硬件木马集的研究, 主要就是根据硬件木马集合的属性, 分析硬件木马在特征表现、所处位置、触发方式、危害结果的数据, 为硬件木马检测和验证

提供理论依据。

3.4 基于硬件木马集的分析结果

根据 3.3 相关定义, 对 Trust-Hub 硬件木马集用 SPSS 统计分析可以得到触发方式、危害结果、所处位置及特征表现的数据。

从表 1 可以看到 combinational logic(组合逻辑)触发方式占到 11.5%, specific combination of data(特殊序列组合)占到 65.4%, counter(计数器)和 state machine(状态机)方式分别占到 7.7%和 15.4%。

表 1 Trust-Hub 硬件木马触发方式统计
Table 1 Trust-Hub Hardware Trojan Triggering Mode

触发方式	百分比
组合逻辑	11.5%
特殊序列组合	65.4%
计数器	7.7%
状态机	15.4%

在表 2 中, 是以硬件木马各种危害结果属性聚类的结果, 信息泄漏占 38.5%, 功能改变占 46.2%, 拒绝服务占 9.6%, 剩下的 5.8%为芯片破坏所占的比重。

表 2 Trust-Hub 硬件木马危害结果统计
Table 2 Harm result of Trust-Hub Hardware Trojan

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 是以硬件木马所处位置属性为主聚类的结果, 端口占到 69.2%, 内部寄存器占到 30.8%, 对 RTL 源代码中的端口进行硬件木马检测分析是十分必要的。

表 3 Trust-Hub 硬件木马所处位置统计
Table 3 Location of Trust-Hub Hardware Trojan

所处位置	百分比
端口	69.2%
内部寄存器	30.8%

表 4 分析得出信息泄漏的硬件木马有 70%直接依附于端口, 剩下 30%是通过内部寄存器间接作用于端口。功能改变的硬件木马 60%直接依附于端口, 40%部分存在于内部寄存器, 而破坏芯片和拒绝服务型的硬件木马则全部依附于端口。

表 4 Trust-Hub 硬件木马危害与位置的关系

Table 4 Relationship between location and harm result

所处位置	信息泄漏	功能改变	拒绝服务	破坏芯片
端口	70.0%	60.0%	100.0%	100.0%
内部	30.0%	40.0%	0.0	0.0

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。RTL 硬件木马本质上是数字电路, 即是由组合逻辑或时序逻辑来设计的。表 5 给出了常见硬件木马的特征表现形式, 可以作为可疑硬件木马的判断标准。

表 5 Trust-Hub 硬件木马特征表现^[27-29]

Table 5 Features of Trust-Hub Hardware Trojan

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
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	计数器出现一个难以达到的数值

虽然每个硬件木马的危害、位置、特征各不相同, 但是分析发现硬件木马主要是直接依附于输入输出端口, 小部分是通过内部寄存器间接作用于输入输出端口, 都是待到特定条件时触发。根据安全风险分析结果, 给出主要的安全风险模型及相应的描述。

(1) 拒绝服务模型

拒绝服务的安全风险, 主要针对总线, 本质是总线控制端口。其抽象模型示意图如 3 所示。图中带箭头实线表示正常在状态转换, 虚线表示转移到拒绝服务状态。

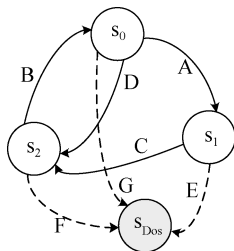


图 3 拒绝服务模型示意图

Figure 3 Schematic diagram of Denial of Service

图 3 拒绝服务模型的形式化表示为一个五元组

$Dos(J, K, T, S, F)$, 其中:

J: 有穷非空状态集 $\{S_0, S_1, S_2, S_{Dos}\}$, 其中 S_0, S_1, S_2 表示正常状态, S_{Dos} 表示拒绝服务状态

K: 有穷非空输入集 $\{A, B, C, D, E, F, G\}$

T: 表示在输入 K 集中某元素时, J 状态集中某两个状态之间的转换函数

\Rightarrow : 表示状态转换

状态描述如下:

$S_0+A \Rightarrow S_1$ 表示当前状态为 S_0 , 在输入 A 的情况下, 转换到状态 S_1 ;

$S_0+D \Rightarrow S_2$ 表示当前状态为 S_0 , 在输入 D 的情况下, 转换到状态 S_2 ;

$S_0+G \Rightarrow S_{Dos}$ 表示当前状态为 S_0 , 在输入 G 的情况下, 转换到状态 S_{Dos} ;

$S_1+E \Rightarrow S_{Dos}$ 表示当前状态为 S_1 , 在输入 E 的情况下, 转换到状态 S_{Dos} ;

$S_1+C \Rightarrow S_2$ 表示当前状态为 S_1 , 在输入 C 的情况下, 转换到状态 S_2 ;

$S_2+B \Rightarrow S_0$ 表示当前状态为 S_2 , 在输入 B 的情况下, 转换到状态 S_0 ;

$S_2+F \Rightarrow S_{Dos}$ 表示当前状态为 S_2 , 在输入 F 的情况下, 转换到状态 S_{Dos} 。

(2) 系统提权模型

系统提权的安全风险, 主要对芯片内部的关键寄存器。其抽象模型示意图如 4 所示。图中带箭头实线表示从用户状态 U 到系统状态 S 的正常转换, 虚线表示从用户状态 U 到系统状态 S 的非正常转换, 即非法系统提权。

图 4 系统提权模型的形式化表示为一个六元组 $Utos(U, S, T, P, R, N)$, 其中:

U: 用户态

S: 系统态或内核态

T: 表示 U 和 S 之间的转换关系

P: 表示 U 通过正常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

\Rightarrow : 表示状态转换

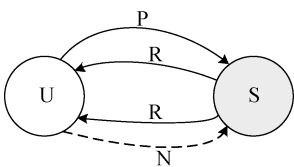


图 4 系统提权模型示意图

Figure 4 Schematic diagram of Systematic empowerment

状态描述如下:

$U+P \Rightarrow S$ 表示当前状态为 U , 在 P 的作用下, 转换到状态 S ;

$S+R \Rightarrow U$ 表示当前状态为 S , 在 R 的作用下, 转换到状态 U ;

$U+N \Rightarrow S$ 表示当前状态为 U , 在 N 的作用下, 转换到状态 S 。

(3) 改变功能模型

改变功能的安全风险, 主要针对输入输出端口。其抽象模型示意图如 5 所示。图中左半部分带箭头实线表示在正常状态之间转换, 右半部分带箭头实线表示在功能改变后各状态之间的转换, 带箭头虚线从正常工作状态表示转移到功能改变状态。

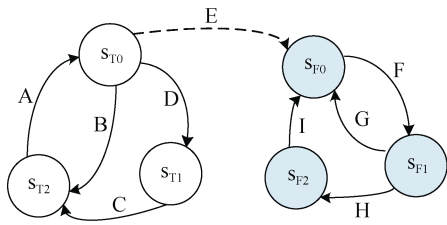


图 5 改变功能模型示意图

Figure 5 Schematic diagram of Change function

图 5 改变功能模型的形式化表示为一个三元组 $C_F(S, K, T)$, 其中:

S : 有穷非空状态集 $\{S_{T0}, S_{T1}, S_{T2}, S_{F0}, S_{F1}, S_{F2}\}$, 其中 S_{T0}, S_{T1}, S_{T2} 表示正常状态, S_{F0}, S_{F1}, S_{F2} 表示功能改变后的工作状态

K : 有穷非空输入集 $\{A, B, C, D, E, F, G, H, I\}$

T : 表示在输入 K 集中某元素时, S 状态集中某两个状态之间的转换函数

\Rightarrow : 表示状态转换

状态描述如下:

$S_{T0}+D \Rightarrow S_{T1}$ 表示当前状态为 S_{T0} , 在输入 D 的情况下, 转换到状态 S_{T1} ;

$S_{T0}+B \Rightarrow S_{T2}$ 表示当前状态为 S_{T0} , 在输入 B 的情况下, 转换到状态 S_{T2} ;

$S_{T1}+C \Rightarrow S_{T2}$ 表示当前状态为 S_{T1} , 在输入 C 的情况下, 转换到状态 S_{T2} ;

$S_{T2}+A \Rightarrow S_{T0}$ 表示当前状态为 S_{T2} , 在输入 A 的情况下, 转换到状态 S_{T0} ;

$S_{T0}+E \Rightarrow S_{F0}$ 表示当前状态为 S_{T0} , 在输入 E 的情况下, 转换到状态 S_{F0} ;

$S_{F0}+F \Rightarrow S_{F1}$ 表示当前状态为 S_{F0} , 在输入 F 的情况下, 转换到状态 S_{F1} ;

$S_{F1}+G \Rightarrow S_{F0}$ 表示当前状态为 S_{F1} , 在输入 G 的

情况下, 转换到状态 S_{F0} ;

$S_{F1}+H \Rightarrow S_{F2}$ 表示当前状态为 S_{F1} , 在输入 H 的情况下, 转换到状态 S_{F2} ;

$S_{F2}+I \Rightarrow S_{F0}$ 表示当前状态为 S_{F2} , 在输入 I 的情况下, 转换到状态 S_{F0} 。

(4) 信息泄漏模型

信息泄漏的安全风险, 主要依附在输出端口。其抽象模型示意图如 6 所示。

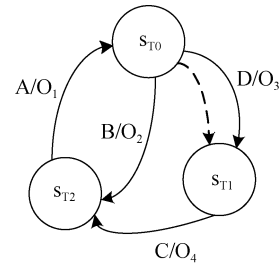


图 6 信息泄漏模型示意图

Figure 6 Schematic diagram of Information Leakage

图中带箭头实线表示在正常状态之间的转换, 和实线并行的虚线表示有潜在的侧信道将信息泄漏出去。

图 6 信息泄漏模型的形式化表示为一个三元组 $I_L(S, K, O, T)$, 其中:

S : 有穷非空状态集 $\{S_{T0}, S_{T1}, S_{T2}\}$, 其中 S_{T0}, S_{T1}, S_{T2} 表示正常状态

K : 有穷非空输入集 $\{A, B, C, D\}$

O : 有穷非空输出集 $\{O_1, O_2, O_3, O_4\}$

T : 表示在输入 K 集中某元素时, S 状态集中某两个状态之间的转换函数, 并产生输出集 O 中的元素

\Rightarrow : 表示正常转换

$-->$: 表示有信息泄漏的转换

状态描述如下:

$S_{T0}+D \Rightarrow S_{T1} + O_3$ 表示当前状态为 S_{T0} , 在输入 D 的情况下, 转换到状态 S_{T1} , 输出为 O_3 ;

$S_{T0}+D --> S_{T1} + O_3$ 表示当前状态为 S_{T0} , 在输入 D 的情况下, 转换到状态 S_{T1} , 输出为 O_3 , 这是一种隐藏的侧信道将信息泄漏出去;

$S_{T1}+C \Rightarrow S_{T1} + O_4$ 表示当前状态为 S_{T1} , 在输入 C 的情况下, 转换到状态 S_{T2} , 输出为 O_4 ;

$S_{T2}+A \Rightarrow S_{T0} + O_1$ 表示当前状态为 S_{T2} , 在输入 A 的情况下, 转换到状态 S_{T0} , 输出为 O_1 ;

$S_{T0}+B \Rightarrow S_{T2} + O_2$ 表示当前状态为 S_{T0} , 在输入 B 的情况下, 转换到状态 S_{T2} , 输出为 O_2 。

4 基于信号流向的多叉树分层搜索方法

目前, 对于芯片的 RTL 源代码, 包括第三方 IP

核安全风险评估工作^[27-29], 已经展开相关研究, 文献[39]提出了控制流图的方法来检测 RTL 中的硬件木马, 它将 RTL 中的硬件木马抽象成几种控制流模型, 根据不同的模型来检测不同的硬件木马; 文献[40]采用形式化验证的方法来检测第三方 IP 核 RTL 中的信息泄漏类型的硬件木马; 文献[41]提出了针对处理器关键寄存器的模型检验方法, 来验证关键寄存器是否存在修改数据的硬件木马。在相关研究的基础上以及方法的启发下, 试图从另外一种途径来检测 RTL 中的硬件木马, 提出了基于信号流向的多叉树分层搜索方法, 原因如下:

(1) 在对硬件木马标准集进行分析时发现, 有 69.2% 的硬件木马与模块端口有关, 而信号是通过端口流入和流出的。

(2) Verilog 设计语言本身的特点^[30-38]。如采用“自顶向下、逐步细化”的层次化模块化设计方法, 其实这就是一种多叉树式的数据结构形式。RTL 文件是由 Verilog 基本元素构成的, 这些构成元素主要包括基本设计单元, 即模块, 模块分为描述接口和描述逻辑功能两部分。

(3) 硬件木马要对芯片施加影响, 在 RTL 源代码中必然有相应的语句来完成木马的功能, 并且该语句必然存在于当前模块或调用的某一个模块当中。只要顺着芯片输入或输出端口, 从顶层端口往下层进行相应分析, 则可以找出可疑的模块及 RTL 源代码语句。

4.1 传统多叉树搜索方法

传统多叉树搜索方法主要是遍历多叉树(Traversing Binary Tree): 是指按指定的规律对多叉树中的每个结点访问一次且仅访问一次。主要包括先序遍历、中序遍历、后序遍历。以图 7 多叉树为例, 以某一关键词为搜索根据, 进行先序遍历, 则查找结点的顺序为 A、B、H、I、N、O、P、C、J、D、E、K、L、F、M, 无论是哪种遍历方式, 都需要把所有的结点访问一次。

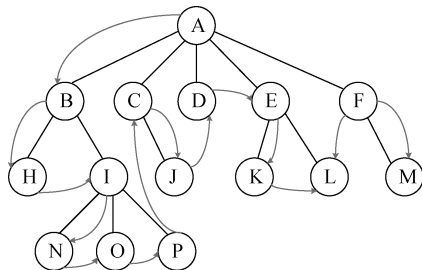


图 7 多叉树先序遍历示意图

Figure 7 Pre-ordered Traversal diagram of multi-fork tree

(1) 空间复杂度

假设任意一个结点的最大孩子结点数为 N ;

假设多叉树的最大深度为 K 。

则一棵深度为 K 的多叉树的结点个数为

$$\sum_{i=0}^{K-1} N^i = \frac{(N^K - 1)}{(N - 1)}。由于每个结点的孩子结点数为 N , 所以每个结点需要 N 个指针变量, 由此得出, 指针变量的空间个数为 $N \times \sum_{i=0}^{K-1} N^i = N \times \frac{(N^K - 1)}{(N - 1)}$ 。由此$$

可见, 当 N 和 K 较大时, 需要占用比较大的额外空间。

实际上, 由于 RTL 源文件中的模块调用关系构成的多叉树基本上不会为满树, 占用的空间不会很多。

(2) 时间复杂度

由以上假设可知, 多叉树的最大深度为 K , 每个结点的最大孩子结点数为 N , 要查找一个结点, 在最糟糕的情况下, 需要进行 $K \times N$ 次比较, N 和 K 都是取有限值的常数, 所以搜索时间复杂度为 $O(1)$ 。同理, 查找 n 个结点, 在最坏的情况下, 搜索平均时间复杂度为 $O(n)$ 。

4.2 基于信号流向的多叉树搜索方法

4.2.1 信号流向

数字芯片从整体来看, 可以抽象成一个多输入输出的系统, 如图 8 所示。在图 8 中, 输入信号为 $f_1(k)$, $f_2(k)$, ..., $f_p(k)$, 输出信号为 $y_1(k)$, $y_2(k)$, ..., $y_q(k)$, $\{x_i(k)\}$ 表示芯片处理过程中间的信号。

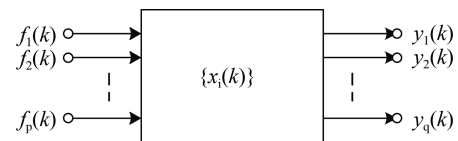


图 8 数字芯片抽象模型图

Figure 8 Digital chip abstract model diagram

如果从芯片内部看, 可以把信号流向做出抽象, 以二输入二输出端口为例, 抽象成如图 9 所示。

在图中, 输入信号为 $f_1(k)$ 、 $f_2(k)$, 输出信号为 $y_1(k)$ 、 $y_2(k)$, $x_1(k)$ 、 $x_2(k)$ 、 $x_3(k)$ 、 $x_4(k)$ 为中间信号, 从左向右的箭头表示信号前向方向, 从右向左的箭头表示后向方向, 即信号有反馈。

如果硬件木马要对输入输出端口施加影响, 那么必然对信号的流向或信号的取值造成影响, 以达到攻击的目的, 因此为了对影响端口的硬件木马进行跟踪查找, 可以顺着信号的流向来进行分析。

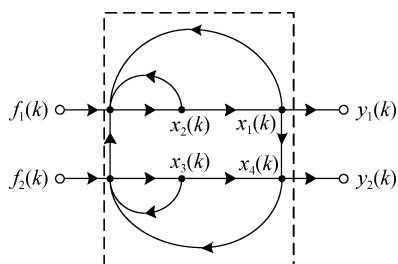


图 9 数字芯片信号流向模型图

Figure 9 Digital chip signal flow model diagram

4.2.2 基于信号流向的多叉树搜索方法

面向信号流向的多叉树搜索方法与图 7 所示不同。以图 10 为例说明, 其中每个结点代表 RTL 源文件中的一个模块, 搜索关键词是某端口名称, 图 10 所示的搜索方法只查找与端口相关的结点, 而不访问无关的结点, 可以减少搜索结点数。每次搜索只是沿着根结点, 顺着相关路径查找, 不需要全部遍历。

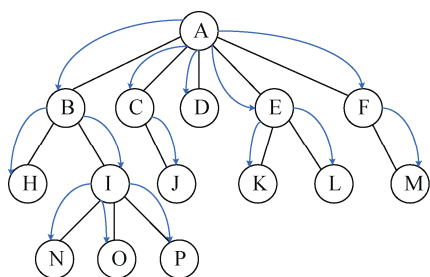


图 10 基于信号流向的多叉树搜索示意图

Figure 10 Pre-ordered Traversal diagram of multi-fork tree based on signal flow

命题: 假设多叉树有 n 个结点, 如果要搜索 m 个关键词, 则面向信号流向的多叉树搜索方法优于传统多树搜索遍历方法。

证明: 对于传统多叉树搜索遍历方法, 要查找一个关键词, 需要搜索整个多叉树的结点, 即, 查找一个关键词, 搜索结点个数为 n , 查找 m 个关键词的全部搜索结点个数为 $m \cdot n$ 个。

对于信号流向的多叉树搜索, 每查找一个关键词, 只需要查找与本关键词相关的结点, 设查找第 k 关键词结点的个数 L_k , 则 $1 \leq L_k \leq n$, (其中, $1 \leq k \leq m$), 查找全部关键词的个数为 $L_1 + L_2 + \dots + L_k + L_m =$

$$\sum_{k=1}^m L_k \leq m \cdot n. \text{ 命题得证。}$$

主要以开源 CPU 代码 or1200、turbo8051、sparc64socT1-CPU、openfire2、m32632、minirisc、fwrisc 进行了实验, 实验证明, 该方法是可行的、有

效的。

下表 6 列出了传统搜索方法与基于信号流向的搜索方法的比较情况。通过比较发现, 基于信号流向的搜索方法在搜索次数上优于传统方法。

表 6 传统搜索与信号流向搜索方法比较

Table 6 Comparison of traditional search and signal

名称	文件数	端口数	flow	
			传统搜索次数	本文方法
or1200	79	56	4424	112
turbo8051	29	19	551	38
sparc64soc	126	13	1638	26
openfire2	27	14	378	28
m32632	16	37	592	74
minirisc	7	7	49	14
fwrisc	7	9	63	18

RTL 源代码“自顶向下、逐步细化”, 及层次化调用的特点, 决定了整个 RTL 源代码文件的结构是一个树形结构, 并且是一个多叉树结构(因为一个顶层模块有可能调用 2 个以上的模块)。一个复杂的电路模型可以通过多个 Verilog HDL 模块构成, 每一个模块又可以由多个子模块来成, 这种模块间分层次调用的结构可以描述极其复杂的电路设计。因此多叉树搜索原理是先读取 RTL 源文件, 构造出模块之间的逻辑关系, 即多叉树这种数据结构。每个多叉树结构的结点主要包括模块名称、所处深度、父结点指针、子结点指针、输入输出端口名称等信息。构造完多叉树之后, 以顶层模块的某一关注的输入或输出端口名称为关键词, 对多叉树进行层次化搜索, 找出本模块与此端口相关的 RTL 语句, 找出下一调用模块中与该端口例化相关 RTL 语句, 直到查找完为止。给出分析流程图如图 11 所示。

从流程图 11 可以看出, 在开始之后, 先读取 RTL 源文件, 进行预处理操作, 去掉 RTL 源文件中的注释等冗余信息。然后通过构建多叉树, 利用多叉树分层搜索方法查找某端口的相关 RTL 语句, 查到相关语句之后与表 5 的特征表现作对比, 看是否符合表 5 中的硬件木马表现形式, 最终定位出可疑的 RTL 硬件木马语句。

基于信号流向的多叉树搜索的伪代码如表 7 所示。

5 实验

(1) 实验对象

Ttrust-Hub 硬件木马数据集及开源 CPU 代码。

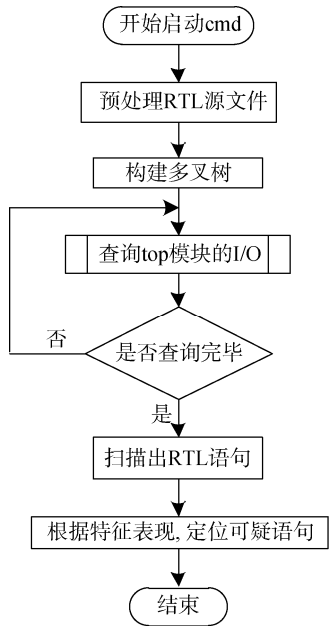


图 11 RTL 级硬件木马分析流程图

Figure 11 RTL Hardware Trojan Analysis Framework

表 7 基于信号流向的多叉树搜索伪代码

Table 7 Algorithm of multi-fork tree search pseudo code

Algorithm	multifork tree search
Input:	搜索关键词为顶层 top 模块的输入输出端口名称
Output:	与关键词相关的 RTL 语句
1: (Initialization)	预处理分析 RTL 源文件模块及各模块之间的调用关系.
2: (construct multi-tree)	以顶层模块 top module. 为根结点, 根据预处理中的各模块的调用关系构建多叉树. 定义多叉树结点的结构体变量 Struct node {module name, input port, output port, file name, parent node, children node}
3: (searching)	traverse the multifork tree with port name as keyword to find port related statements.
4: Output	输出与端口名称相关的可疑 RTL 语句

(2) 评价指标

对于一句 RTL 源代码, 要么是硬件木马语句 (positive) 或不是 (negative)。这样地话会出现四种情况。如果一个是硬件木马并且也被判为硬件木马, 则称为真正类 (True positive), 如果不是硬件木马而被判为硬件木马, 则称之为假正类 (False positive)。同样地, 如果不是硬件木马而被判为不是, 称之为真负类 (True negative), 不是硬件木马而被判断为硬件木马则为假负类 (false negative)。

TP: 正确肯定的数目;

FN: 漏报, 没有正确找到的匹配的数目;

FP: 误报, 给出的匹配是不正确的;

TN: 正确拒绝的非匹配对数;

TPR (True Positive Rate) 所有正类中, 有多少被预测成正类 (正类预测正确), 给出定义如下:
 $TPR = TP / (TP + FN)$;

TNR (True Negative Rate) 所有反类中, 有多少被预测成反类 (反类预测正确), 给出定义如下:
 $TNR = TN / (FP + TN)$;

FPR (False Positive Rate) 所有反类中, 有多少被预测成正类 (正类预测错误) 即误报率, 给出定义如下:
 $FPR = FP / (FP + TN)$;

FNR (False Negative Rate) 所有正类中, 有多少被预测成反类 (反类预测错误) 即漏报率, 给出定义如下:
 $FNR = FN / (TP + FN)$ 。

在实际中用到 FPR 和 FNR 比较多, 因此在以下实验中, 主要关注这两个指标。

(3) 实验

先以 Trust-Hub 硬件木马集的 AES 和 RS232 系列测试集做了实验分析。通过实验发现, 基于信号流向的、从模块端口自顶向下的分析方法, 对于依附在模块端口上的硬件木马是比较有效的, 而对于内部寄存器的木马则有很大的局限性, 所以还需要结合其他方法来弥补该方法的不足之处。

Trust-Hub 硬件木马集每个案例包含两部分源代码, 分别在 TjFree 和 TjIn 文件夹下。凡是 TjFree 文件夹下的源文件表示是干净源代码, 即没有插入 RTL 硬件木马的源代码文件, 而 TjIn 文件夹下则是含有 RTL 硬件木马的源代码文件。

Trust-Hub 硬件木马集是有一定局限性 (代码量少和样本少等) 的, 原因在于测试集源代码本身的特点: 一是测试集系列 (如 AES 或 RS232 系列等) 案例过于相似, 导致样本空间小; 二是测试集中的每个案例代码量比较小, 代码书写特征表现不丰富; 三是端口数目少, 硬件木马的设计位置相对固定, 以至于不容易出现误报和漏报的情况, 这与实际源代码 (代码量大、文件多、代码形式表现丰富) 安全评估工作环境是有差别的, 反映了 Ttrust-Hub 测试集的局限性。

为此, 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, 有助于检验本文方法, 并比较客观地分析 FPR 和 FNR 情况)。实验分两步进行, 第一步, 对未插入硬件木马时的源代码分析, 主要观察误报率; 第二步, 对端口和部分内部寄存器插入硬件木马 (模仿 Trust-Hub 硬件木马特征风格书写) 的情况进行分析, 主要观察漏报率。

表 8 AES 测试集实验结果

Table 8 Analysis of RTL source code of Trust-Hub AES		
基准代码	模块端口	可疑硬件木马语句定位
AES-T100	capacitance	lfsr.v 的 13 行和 17 行, 35 行到 105 行
AES-T200	capacitance	lfsr.v 的 13 行和 17 行, 35 行到 105 行
AES-T300	state	TSC.v 的 31、35、39、43、47、51 行
AES-T400	state	Trojan_Trigger.v 的 48 行
AES-T500	capacitance	TSC.v 的 13、17 行
AES-T600	state	Trojan_Trigger.v 的 48 行
AES-T700	capacitance	lfsr.v 的 12 行和 17 行
AES-T900	capacitance	lfsr.v 的 13 行和 17 行
AES-T900	capacitance	lfsr.v 的 13 行和 17 行
AES-T1600	capacitance	lfsr.v 的 13 行和 17 行

表 9 RS232 测试集实验结果

Table 9 Analysis of RTL source code of Trust-Hub RS232		
基准代码	模块端口	可疑硬件木马语句定位
RS232-T100	rec_readyH	u_rec.v 的 58 行
RS232-T300	xmit_dataH	xmit.v 的 61 行和 176 行
RS232-T400	rec_dataH	uart.v 的 86 行
RS232-T500	xmit_doneH	xmit.v 的 185 行
RS232-T700	xmit_doneH	xmit.v 的 189、196、202、208、214、226 行
RS232-T800	rec_readyH	u_rec.v 的 54 行
RS232-T900	xmit_doneH	xmit.v 的 187、195、202、208、214、220 行
RS232-T901	xmit_doneH	xmit.v 的 191、198、205、211、217、223 行

表 10 or1200_alu.v 未插入硬件木马时的实验结果
Table 10 Result of or1200_alu.v without Hardware Trojan

端口名称	扫描结果	判定结果	FPR
A	3	0	*
B	3	0	*
mult_mac_result	5	2	*
macrc_op	1	0	*
alu_op	13	0	*
alu_op2	3	0	*
comp_op	3	0	*
cust5_op	1	0	*
cust5_limm	1	0	*
result	13	3	*
flagforw	2	0	*
flag_we	2	0	*
ovforw	2	0	*
ov_we	2	0	*
cyforw	3	0	*
cy_we	3	0	*
carry	1	0	*
flag	1	0	*
共计个数	62	5	8.10%

表 11 or1200_alu.v 插入硬件木马时的实验结果
Table 11 Result of or1200_alu.v with Hardware Trojan

端口名称	加入个数	发现个数	FNR
A	1	1	*
B	1	1	*
mult_mac_result	1	1	*
macrc_op	1	1	*
alu_op	1	1	*
alu_op2	1	1	*
comp_op	1	1	*
cust5_op	1	1	*
cust5_limm	1	1	*
result	1	1	*
flagforw	1	1	*
flag_we	1	1	*
ovforw	1	1	*
ov_we	1	1	*
cyforw	1	1	*
cy_we	1	1	*
carry	1	1	*
flag	1	1	*
comp_a	1	0	*
comp_b	1	0	*
共计个数	20	18	10.00%

由于实验相关源文件端口较多不再一一列出, 直接给出结果如表 12 和表 13 所示。

表 12 未插入硬件木马时的实验结果

Table 12 Experiment result without Hardware Trojan				
文件名称	端口数	扫描结果	判定结果	FPR
or1200_alu.v	18	62	5	8.10%
or1200_ctrl.v	59	158	21	13.30%
oc8051_alu.v	13	248	22	8.80%
M32632.v	37	212	23	10.80%
alu.v	40	164	20	12.19%

表 13 插入硬件木马时的实验结果

Table 13 Experiment result with Hardware Trojan			
文件名称	加入木马数	判定结果	FNR
or1200_ctrl.v	20	18	10.00%
or1200_ctrl.v	64	59	7.81%
oc8051_alu.v	15	13	13.30%
M32632.v	40	35	12.50%
alu.v	45	41	8.89%

(4) 分析

产生误报的原因是因为在对端口信号搜索的结果中, 含有和硬件木马接近的表达逻辑, 单独从特征上是区分不开的, 所以被识别成硬件木马。例如 or1200_ctrl.v 的误报率比 or1200_alu.v 的高, 原因是 or1200_ctrl.v 是 OR1200 CPU 的控制逻辑模块, 它里面含有比较多的控制信号, 如 wb_freeze 锁流水线信号、sig_syscall 系统调用信号、ex_spr_read 读信号、ex_spr_write 写信号, 这些信号都是由多个变量的组合逻辑构成, 在表达上与硬件木马相似, 很容易被误判。另外, 产生漏报的原因是: 分析是由模块端口入手的, 主要搜索到与模块端口相关联的语句, 而模块内部定义的寄存器等关联不到(包括涉及状态机寄存器中的硬件木马), 所以导致漏报。

总之, 基于信号流向的、从模块端口自顶向下的分析方法, 对于依附在模块端口上的硬件木马是比较有效的, 而对于内部寄存器的木马则有一定的局限性, 所以还需要结合其他方法来弥补该方法的不足之处。

(5) 与现有研究方法比较

由于观察问题的角度和采用的方法不同, 与其他方法比较时采用了定性分析。

文献[39]采用控制流图方法来检测 RTL 级硬件木马, 并且将木马抽象成 cheat code、dead machine、ticking timebomb 三种模型, 然后对 RTL 源代码提出

相应的代码段与模型做匹配。这种方法其实是借鉴软件木马匹配检测的方法, 在 3.2 节中已经分析这种方法的局限性, 原因在于硬件木马的设计与软件木马具有很大的不同。

文献[40]采用形式化方法, 即模型检验来检测硬件木马, 这不可避免的存在状态空间爆炸问题。对于文件数量多、变量多、状态空间多的情况不合适。

文献[41]主要是针对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, 研究的木马表现形式相对而言比较单一。

本文的研究方法主要是面向源代码安全性评估的工作, 一般来说面临源文件数量比较多、文件中变量比较多、端口也比较多的情况, 在这种情况下, 如果采用仿真或形式化方法是难以对代码做整体评估的。因此, 本文只是把所面对的 verilog 文件当作文本来对待, 结合硬件木马集的逻辑特征表现形式及 verilog 本身的特点, 试图用一种比较通用性较强的方法找出潜在的可疑的语句, 不存在状态空间爆炸、向量空间膨胀等问题。与其他方法相比较而言, 适合于对外来第三方源代码做前期安全性评估, 将可疑代码范围缩小, 提高评估效率。当然, 与以上三篇文献相比存在不足之外, 如对某一类型的硬件木马分析的精细程度和深入程度还不够, 需要结合相关研究成果进一步挖掘潜在的研究点。

6 总结与展望

当前, 关于硬件木马的检测方法多集中在硅后, 如旁路分析、或物理检测等, 而对于 RTL 源代码中加入硬件木马进行检测研究的方法相对而言比较缺乏。

针对目前 RTL 级硬件木马研究缺少描述形式及抽象模型定义, 基于硬件木马集, 给出了 RTL 级硬件木马的相关定义和安全威胁模型。

针对源代码进行检测主要是传统的仿真验证方法, 其缺点是难以遍历所有向量, 并且运行时间开销大。

文章试图针对在 RTL 源代码中加入硬件木马的情况, 提出了一种分析方法。实验证明, 该方法对在设计阶段加入到源代码中的硬件木马具有一定的效果。

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[C]. *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, 2007: 1-9.
- [4] A.Sanjaya. Hardware Assisted Security: Anticipating Digital Threat and Challenges[C]. *FIRST TC 2012 IDF*, 2012, 1-10.
- [5] Security issues, <http://www.cctime.com/html/2014-12-10/201412101135225626.htm>, Dec, 2014.
- [6] Global information Security issues, <http://sec.chinabyte.com/162/13439662.shtml>, May, 2015.
- [7] Agrawal D, Baktir S, Karakoyunlu D, et al. Trojan Detection Using IC Fingerprinting[C]. *2007 IEEE Symposium on Security and Privacy*, 2007: 296-310.
- [8] Di J, Smith S. A Hardware Threat Modeling Concept for Trustable Integrated Circuits[C]. *2007 IEEE Region 5 Technical Conference*, 2007: 354-357.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] 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.
(王力伟, 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.)
- [19] 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.
- [20] Cakir 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.
- [21] 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.
- [22] 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.
- [23] 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.
- [24] 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.
- [25] Dai Y Y, Brayton R K. Circuit Recognition with Deep Learning[C]. *2017 IEEE International Symposium on Hardware Oriented Security and Trust*, 2017: 162.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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, SS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.)
- [30] Clarke E M, Emerson E A, Sistla A P. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications[J]. *ACM Transactions on Programming Languages and Systems*, 1986, 8(2): 244-263.
- [31] Emerson E A, Halpern J Y. Decision Procedures and Expressive-

- ness in the Temporal Logic of Branching Time[J]. *Journal of Computer and System Sciences*, 1985, 30(1): 1-24.
- [32] Ehrhard T. Linear logic in computer science[M]. Cambridge: Cambridge University Press, 2004.
- [33] Lei S L. Surprise the core step by step: Analysis of the 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.)
- [34] S. Vijayaraghavan and M. Ramanathan. A practical Guide for System Verilog Assertions[M]. *Springer Ltd publishers*, 2005, 1-100.
- [35] Damjan Lampret. OpenRISC 1200 IP Core Specification. <http://www.opencore.org>. 2017.
- [36] OpenRISC 1000 Architecture Manual. <http://www.opencore.org>. 2017.
- [37] Yeung P, Larsen K. Practical Assertion-Based Formal Verification for SoC Designs[C]. *2005 International Symposium on System-on-Chip*, 2006: 58-61.
- [38] 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.
- [39] 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.
- [40] 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.
- [41] 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.



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