doi:10.3772/j.issn.1002-0470.2017.06.003

基于静态特征的硬件木马检测①

赵跃辉②* 沈海华*,**

(*中国科学院大学计算机与控制学院 北京 101408)

(**中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)

摘 要 分析了集成电路全球化设计、制造致使集成电路易被植入硬件木马(HT)从而使其存在遭受恶意攻击隐患的硬件安全形势,以及现有硬件木马检测方法的技术特点,在此基础上提出了一种基于静态特征的硬件木马检测新方法——HTChecker。HTChecker 基于硬件木马的静态特征利用子图同构技术来检测木马。与其他的检测方法相比,它可以快速精确地找出已知特征的硬件木马。为了不受限于机器内存的大小,该方法借助图数据库来存储电路,这样它对超大规模的电路也可以进行检测。使用 ISCAS'89 和 Open-Cores benchmark 电路对 HTChecker 进行了评估,木马电路被随机地插入到这些电路中。实验结果显示 HTChecker 可以快速精确地找出木马,并且不需要"Golden Chip"的辅助。HTChecker 可以有效地处理实际的 VLSI 设计。

关键词 硬件安全,静态特征,硬件木马(HT),木马检测,子图同构

0 引 言

集成电路设计复杂性的急剧增长,推动着半导体产业走向垂直专业化分工,集成电路设计的各个阶段被分解和外包给外面的公司,这种地理上分散的芯片设计极易受到恶意活动的影响,为攻击者在芯片中植入硬件木马(hardware Trojan, HT)创造了可能[1-5]。为确保电路和系统安全,必须研究先进的硬件木马检测方法。然而硬件木马检测相当困难,原因之一来自当前芯片和系统前所未有的设计规模和复杂度。本研究给出了一种基于静态特征的硬件木马检测方法——HTChecker,并对其进行了详细论述和实验验证。

1 相关研究

在过去的几年中,已出现了很多硬件木马检测方法。一些方法是基于传统的验证和测试技术,当硬件木马的大小与待检测电路的大小没有量级上的差异时,它们是非常有效的[6,7]。如果硬件木马的大小与待检测电路相比非常小,就很难找到可以激活木马的测试向量。而且即便是木马被成功激活,也很难观察到它们的恶意效果,尤其是当硬件木马不直接影响芯片的功能时。文献[8]提出了基于随机化的方法来检测木马,把服从一定概率分布的随机输入矢量同时应用于待检测电路和没有进行综合的设计电路,如果对于某一个输入矢量待检测电路和设计电路给出了不同的输出,则该输入矢量被认为是木马感染的指纹;当 N 个输入矢量被验证完,如果待检测电路和设计电路给出了相同的特征则认为是待检测电路是没有木马的,否则会利用统计推理给出一个置信区间。另外一种比较流行的木马检测方法是旁路分析的方法[9-19],该方法通过分析电路的旁路信号(如功耗,延迟等)来判断是否包含木

① 国家自然科学基金(61173001)资助项目。

② 男,1990 年生,硕士;研究方向:计算机应用技术;联系人,E-mail:zhaoyuehui14@mails.ucas.ac.cn

(收稿日期:2016-12-26)

马。文献[14]提出了一个对收集到的功耗信号进行去噪的模型,然后对一个 IC 系列构建旁路指纹,但是由于低的信噪比,这些方法的精度会随着木马大小的减小和工艺差异的增加而显著减小。文献[9]提出了路径延迟分析的策略,通过收集不包含木马的芯片(对于不包含木马的芯片可以通过逆向工程获得的)的路径延迟来构造指纹并对收集到的延迟信息进行降维操作,然后通过同样的操作可以获取待检测芯片的路径延迟信息,与延迟指纹进行对比就可以判断是否包含木马。旁路分析方法主要的优点是可以在不完全触发木马的情况下检测到木马。同时该方法容易受到工艺差异和环境扰动带来的噪声的影响。大多数的旁路分析方法需要与没有木马的"Golden Chip"作对比,而"Golden Chip"在实际中又不易获得。例如,现在的片上系统(SoC)设计经常涉及到很多由第三方供应商提供的 IP,这些 IP 的安全性很难得到保证[20]。

硬件木马检测的问题也越来越受到国内研究者的关注,文献[21]中作者提出了基于红外光谱分析的硬件木马检测方法,该方法可以在二维空间上进行硬件木马检测,减弱了芯片工作时正常逻辑产生的信息对硬件木马产生的信息的掩盖,同时利用红外光谱仪捕获被检测点产生的红外光谱,避免了利用红外热像仪时其积分机理导致的信息损失;该方法可以对逻辑能耗量级为 10^{-3} 的硬件木马进行有效区分,但对同一木马的不同实现方式只具备一定的区分力度。文献[22]在基于功耗旁路信号分析的方法上提出了利用最大间距准则处理旁路信号,构建基准芯片与木马芯片旁路信号之间最大差异的投影子空间,通过比较投影之间的差异来判断芯片中是否包含木马。文献[23]提出了基于电路平均动态电流和最大工作频率的旁路分析的硬件木马检测方法,通过使用多个旁路信号间的关系,进一步抵御噪声的影响,提高木马检测效率。文献[24]在芯片功耗模型的基础上,提出了根据工艺偏差和木马电路功耗数据特征的不同来对功耗数据进行特征抽取,从而减弱工艺偏差对检测的影响,建立了基于重叠率检验的数理统计检测模型。文献[25]提出了一种非破坏性的、基于旁路分析的硬件木马检测方法,通过对芯片功耗瞬态变化情况的分析,用奇异值分解算法对功耗进行统计来检测芯片中的硬件木马,并在现场可编程门阵列(FPGA)上验证了该方法。文献[26]提出了一种基于主成分分析并结合马氏距离的检测方法,通过对芯片功耗进行建模分析,先通过主成分分析对旁路信息中微小差异进行放大提取,获取主特征,然后使用马氏距离进行判别区分,从而判断是否有木马。

本文提出了一种基于静态特征的精确检测硬件木马的方法,称为 HTChecker。与旁路分析方法相比,HTChecker 不需要"Golden Chip"的辅助,同时由于 HTChecker 是基于静态特征来检测木马,因此不会因为环境因素和工艺差异而影响查找精度。与基于红外光谱分析的硬件木马检测方法相比,HTChecker 对木马电路的功耗量级没有要求,而基于红外光谱分析的硬件木马检测方法[21]目前只能检测逻辑能耗量量级为 10^{-3} 的硬件木马。HTChecker 主要是运用子图同构的思想来检测木马,通过利用电路设计的知识和木马检测的规则对子图同构算法 VF2 进行了改进使之更适合于木马检测,得到了硬件木马检测算法(TDA),TDA 在木马检测方面具有很好的效率。实验结果显示,与 VF2 相比,TDA 在检测木马所用的时间上有了非常大的提升。另外,由于现代集成电路的规模都非常的庞大,为了满足对超大规模集成电路的检测,本研究引入了图数据 neo4j[27]来存储超大规模的电路,这样就可以借助图数据库存储电路的数据,当需要访问电路时,直接从 neo4j 数据库中取数据,实验结果显示使用了 neo4j 后由于数据需要从磁盘中读取,检测算法花费的时间也相应地增加,但是总体的性能还是非常高效的。

2 电路建模

2.1 电路映射为有向图

HTChecker 检测木马是基于超大规模集成电路(VLSI)设计门级网表的静态特征,通过将门级网表映射为有向图,把硬件木马检测转换为子图同构的过程。门级网表是对电路连接性关系的描述,包括电路单元、连线以及它们之间的关系。通过使用一

个点来表示网表中的一个单元,用一条边来表示网表中的一条连线,根据电路网表中每一个单元的输入输出可以获得每一条边的方向,这样电路的门级网表就可以转换为一个等价的有向图。

HTChecker 采用递归解析的技术把芯片的门级网表形式映射为有向图。映射的过程如下:

- 首先找到网表中单元的定义语句,检查该语句是否是完整的。
- 然后解析单元定义语句获取单元类型和相应的输入输出边,然后创建节点,节点的标签是该单元的类型。
- 根据上一步获取的边,创建对应有向图的边。
- 根据边的输入输出方向,设置第二步创建的节点为边的终点或者源点。

通过上面的步骤就可以把芯片的门级网表转换为有向图,然后就可以利用图论中的知识检测木马了,本文使用子图同构技术来检测木马。

2.2 使用 neo4j 存储电路

neo4j 是一个基于磁盘的事务性开源的图数据库,具有高可扩展性,可以支持上亿级别的节点。在 neo4j 数据库中,所有的对数据的修改操作都需要在一个事务中完成。它的数据存储在一个图结构中而不是表中,这就意味着数据之间的关系也可以被充分利用。在 neo4j 中节点以及节点之间的关系都可以有属性,这与我们当前的应用场景非常的一致。对于电路来说,本质上就是一个有向图结构,电路中的每一个电路单元都会有自己的属性(类型,名字等),单元之间的连线对应于图中的边。

对于超大规模的电路来说,受限于机器内存的影响,电路对应的有向图超过了内存的容量,导致木马检测方法失效。基于磁盘的图数据库的使用,可以完美地解决这个问题,大规模的图结构存放在磁盘中,当需要访问图中的数据时,向数据库进行查询操作。因此,我们的木马检测方法会在电路规模非常大时,把电路对应的图结构存储在 neo4j 数据库中,这个过程可以在对电路的网表文件进行解析的时候同时进行。

3 木马检测算法

在模式分析和模式识别领域,子图同构是一个热门的话题。很多的子图同构算法已经被提出,如 Ullmann[28], Nauty[29], VF[30], VF2[31] 等。与其他的子图算法相比 VF2 是一个比较高效的算法,所以利用 VF2 进行木马检测是一个不错的选择。子图同构是一个 NP 完全问题,对于 VF2 来说,在最好情况下的复杂度为 $O(N2)$,最坏情况下是 $O(N!\ N)$[32]。我们通过使用集成电路设计的知识和木马检测中的特殊规则来优化该算法,然后得到了木马检测算法(TDA)。

3.1 基于子图同构的木马检测算法

木马检测问题可以被看做是图 $D=(N_1, B_1)$ 和图 $T=(N_2, B_2)$ 之间的一个匹配过程,D 表示待检测电路对应的有向图,T 表示木马电路对应的有向图。N_1 和 N_2 表示图的顶点的结合,B_1 和 B_2 表示图的边的集合。

木马检测等价于在找到一个映射集 $M \subset N1 \times N2$, M 是顶点对(n, m)的集合,其中 $n \in N_1$, $m \in N_2$, M 中的每一个顶点对表示图 D 中的一个顶点 n 与图 T 中的顶点 m 的映射。

映射集 M 的构建可以通过状态空间来描述。匹配过程的每一个中间状态 s 对应于部分匹配集 $M(s)$, 其中 $M(s)$ 是 M 的一个子集。对于包含在 $M(s)$中的顶点,子图 $D(s)=(M_D(s), B_D(s))$ 表示 $M(s)$ 中待检测电路中匹配的子图, $T(s)=(M_T(s), B_T(s))$ 表示 $M(s)$中木马电路中匹配的子图。其中 $M_D(s)$ 和 $(M_T(s)$ 表示的是中间状态 s 中包含的待检测电路中匹配的顶点集和木马电路中匹配的顶点集, $B_D(s)$ 和 $B_T(s)$ 表示的中间状态 s 中包含的待检测电路中的边和木马电路中的边。

根据上面的定义,木马检测算法 TDA 如算法 1 所示。在 TDA 中有几个可行性规则被用来评估候选对集合 $P(s)$中每一个候选对 p 的可行性,可行性规则的具体内容将在下一节进行描述。

算法 1:TDA 算法

```
PROCEDURE Match(s)
输入:中间匹配状态 s;其中初始状态 s0 满足 M(s0) = Ø
输出:待检测电路图 D 与木马电路图 T 之间的匹配映射
if M(s) 覆盖了 T 中的所有节点 then
输出 M(s);
end if
else
/* 计算候选对集 P(s) */
if T_D^OUT(s) 和 T_T^OUT(s) 都不为空 then
P(s) = (n ∈ T_D^OUT(s)) ∧ (m ∈ T_T^OUT(s)) ∧ (Type(n)
== Type(m));
end if
else
if T_D^IN(s) 和 T_T^IN(s) 都不为空 then
   P(s) = (n ∈ T_D^IN(s)) ∧ (m ∈ T_T^IN(s)) ∧ (Type(n)
   == Type(m));
end if
  end else
for each p 属于 p(s) do
    if p 加入到 M(s) 后可以满足可行性规则 F(s, n, m)
then
     M(s') = M(s ∪ p);   //把 p 加入到中间转态 s 中
       call MATCH(s');
     end if
  end for each
  Restore data structures;
end else
ENDPROCEDURE Match
```

3.2 优化规则

在 VF2 算法中,所有的顶点都是等价的,这样算法就需要遍历所有的状态空间,尤其是在最坏的情况下,算法中需要把两个图中所有顶点的笛卡尔积作为候选对。TDA 算法利用了集成电路设计的知识和木马检测的特殊规则来优化了原始的子图同构算法。

VF2 中的两个可行规则 R_{Pred} 和 R_{Succ} 仍然被用在 TDA 中,除此之外,本文定义了适用于 TDA 的另外 3 个规则分别是 R_{InOut},R_{TypeOut} 和 R_{TypeIn}。只有当待检测电路中包含了与木马电路具有相同的单元类型并且这些单元的出度和入度也与木马电路相应单元的出度和入度匹配时,才认为该待检测电路中包含了木马。R_{InOut} 用来过滤那些顶点出度和入度不匹配的候选对,R_{TypeOut} 和 R_{TypeIn} 表示那些具有相同单元类型的顶点才能构建候选对。图 1 显示了剪枝的例子,剪枝后 TDA 算法会迭代地检查邻接节点判断它们是否匹配。

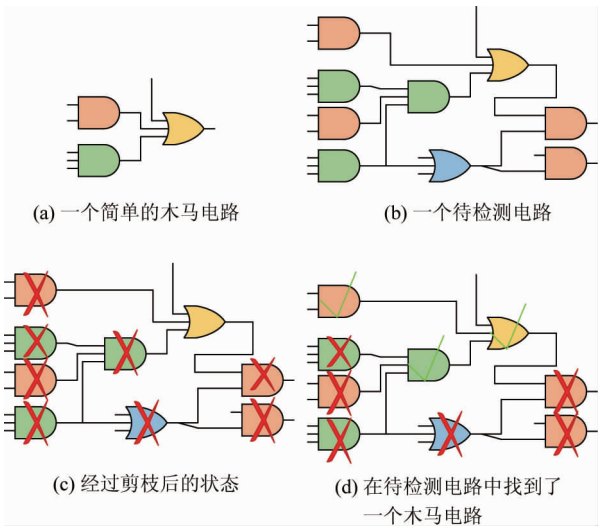

图 1　关于剪枝的一个例子

$$R_{\text{Pred}}(s, n, m) \Leftrightarrow (\forall n' \in D(s) \cap Pred(D,n) \exists m' \in Pred(T, m) \mid (n', m') \in M(s)) \wedge (\forall m' \in T(s) \cap Pred(T,m) \exists n' \in Pred(D, n) \mid (n', m') \in M(s)) \tag{1}$$

$$R_{\text{Succ}}(s, n, m) \Leftrightarrow (\forall n' \in D(s) \cap Succ(D,n) \exists m' \in Succ(T, m) \mid (n', m') \in M(s)) \wedge (\forall m' \in T(s) \cap Succ(T,m) \exists n' \in Succ(D, n) \mid (n', m') \in M(s)) \tag{2}$$

$$\begin{aligned} R_{\text{InOut}}(s, n, m) \Leftrightarrow & (Card(Succ(D, n) \cap T_D^{\text{In}}(s)) \\ & = Card(Succ(T, n) \cap T_T^{\text{In}}(s))) \wedge \\ & (Card(Pred(D, n) \cap T_D^{\text{In}}(s)) \\ & = Card(Pred(T, n) \cap T_T^{\text{In}}(s))) \wedge \\ & (Card(Succ(D, n) \cap T_D^{\text{Out}}(s)) \\ & = Card(Succ(T, n) \cap T_T^{\text{Out}}(s))) \wedge \\ & (Card(Pred(D, n) \cap T_D^{\text{Out}}(s)) \\ & = Card(Pred(T, n) \cap T_T^{\text{Out}}(s))) \end{aligned} \tag{3}$$

$$R_{TypeOut}(s, n, m) \Leftrightarrow n \in T_D^{OUT}(s) \wedge m \in T_T^{OUT}(s) \wedge Type(n) == Type(m) \quad (4)$$

$$R_{TypeIn}(s, n, m) \Leftrightarrow n \in T_D^{IN}(s) \wedge m \in T_T^{IN}(s) \wedge Type(n) == Type(m) \quad (5)$$

3.3 TDA 算法复杂度

在 TDA 中只有那些满足由式(4)和式(5)定义的 $R_{TypeOut}$ 和 R_{TypeIn} 可行性规则的点才会被用来构建候选对 p 并添加到候选对集合 $P(s)$ 中;在候选对集合 $P(s)$ 中,只有那些满足式(3)定义的 R_{InOut} 规则的候选对才会真正地被用来进行判断是否匹配。

用 K 表示木马电路对应的图中的顶点的个数,用 N 表示待检测电路对应的图中的顶点的个数,用 M 表示在待检测电路中满足可行性规则 $R_{TypeOut}$ 和 R_{TypeIn} 的顶点的数量,则在最好情况下和最坏情况下 TDA 需要遍历的候选对的数量如下式所示:

最好情况: $K \times M$ (6)

最坏情况: $K \times M \times (M-1) \times \cdots \times (M-K+1) = \frac{M!}{(K-1)!}$ (7)

通过上面的分析,TDA 的复杂度在最好情况下是 $O(K \times M)$, 最坏情况下是 $O(\frac{M!}{(K-1)!})$。除此之外,利用规则 $R_{InOut}(s, n, m)$ 可以对搜索树进行进一步的剪枝,从而进一步减少 TDA 的搜索空间。

4 实 验

本部分将会通过实验来验证 HTChecker 的有效性。实验是通过 ISCAS'89 和 OpenCores 来进行的。实验电路中的木马是被随机地插入到待检测电路中。

4.1 实验设置

本文实验平台是一台联想公司生产的微型计算机,该机器的型号是 ThinkCentre M8500t-D231,该机器的处理器为 Intel(R) Core(TM) i7-4790,主频为 3.6 GHz,运行内存为 4GB,操作系统是 64 位的 Windows 7 专业版。。由于当木马的大小与待检测电路的大小相差不大时,木马可以很容易地被发现,因此实验中用到的待检测电路是 benchmark 中规模相对比较大的电路。表 1 分别列出了 ISCAS'89[33],木马电路和 OpenCores[32]电路的大小。

表 1 实验中用到的电路的大小

ISCAS'89	#Cell	Trojans	#Cell	OpenCores	#Cell
s38417	23815	T1	8	openMSP430	4518
s38584	20679	T2	8	k68 _ soc	4658
s35932	17793	T3	9	ae18 _ core	4776
s15850	10306	T4	10	mips _ core	6738
s13207	8589	T5 ~ T11	12	M32632	265761
s9234	5808	T12	13	or1200	302093
s5378	2958	T13	14	i650	705660
s1423	731	T14	15		
s1488	659	T15	16		
s1494	653	T16	20		
s1196	547	T17	30		
s1238	526	T18	43		

4.2 TDA 性能分析

本部分会对木马检测算法进行一个详细的测试。本研究考虑了各种可能影响算法的因素包括木马大小,待检测电路的大小以及使用本文的优化规则进行剪枝对算法的影响。为了减小实验中的误差,所有的实验结果都是测量多次后取平均值的结果。

4.2.1 TDA 的综合效率分析

图 2 显示了分别使用 TDA 和 VF2 算法对 ISCAS'89 电路进行木马检测的执行时间比较。从图中可以看出经过改进后得到的 TDA 的用时明显低于VF2的执行时间。从图2中也可以看出不同

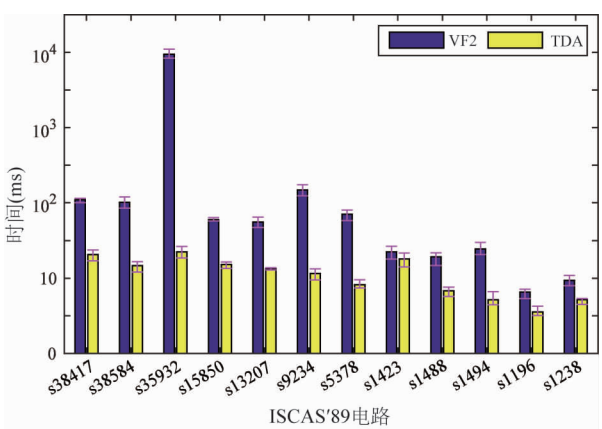

图 2 使用 TDA 和 VF2 算法对 ISCAS'89 电路进行检测木马所用时间

的待检测电路和木马电路 TDA 的执行时间有明显的差异,这是由多种因素引起的,下面会进行详细的讨论。

4.2.2 电路大小对 TDA 的影响

根据 3.3 节的分析,可以看到待检测电路和木马电路的大小会对 TDA 的效率产生影响。为了检测由于木马电路的大小对实验结果的影响,从 ISCAS'89 中选择了电路 s38417(含有 23815 个单元)作为待检测电路,然后选择不同大小的木马电路随机地插入到 s38417 电路中。实验结果如图 3(a)所示,图中横轴表示木马电路的大小,纵轴表示检测算法使用的时间,以时间的对数形式表示。

为了检测被检测电路的大小对实验结果的影响,我们选择木马 T18(包含 43 个单元),然后把 T18 随机地插入到从 ISCAS'89 中选择的待检测电路中。实验结果如图 3(b)所示。横坐标表示待检测电路的大小,纵坐标以对数形式表示检测算法使用的时间。

从图 3 可以看出,木马的大小对 TDA 和 VF2 没有非常明显的影响,但是待检测电路的大小对 TDA 和 VF2 都有明显的影响。可以看出待检测电路的大小对检测算法的影响不是线性的,这是因为还有其他的因素影响检测算法的效率,比如剪枝的效果。

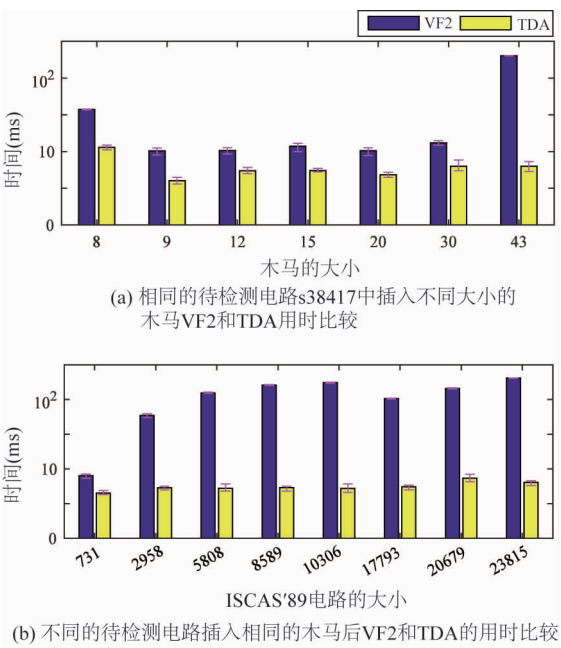

图 3 不同大小木马电路和待检测电路对 TDA 和 VF2 检测效率的影响

4.2.3 算法优化效果

根据 3.3 节的分析,可以看到除了待检测电路和木马电路的大小对检测算法效率的影响之外,剪枝的效果也会影响 TDA 的效率。为了验证剪枝的效果对 TDA 的影响,从 ISCAS'89 中选择了电路 s38417 作为待检测电路,木马电路选择了具有相同大小但是是由不同的单元类型构成的电路 T5 ~ T11,它们的大小都是包含 12 个单元。实验结果如图 4 所示。横坐标表示剪枝以后候选对的数量,纵坐标以对数的形式表示检测算法使用的时间。

从图 4 可以看出,剪枝的效果对 TDA 有着明显的影响,但是 TDA 消耗的时间并没有随着剪枝后候选对的数量的增加而增加,这是因为还有其他的因素会影响 TDA 的效率,如待检测电路和木马电路结构的相似性等。

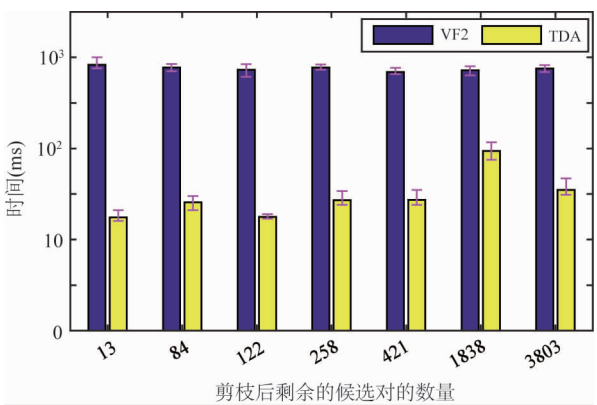

图 4 相同的待检测电路中插入大小相等结构不同的木马后的剪枝效果

4.3 HTChecker 的有效性分析

为了验证 HTChecker 对大规模电路的有效性,本文从 OpenCores 中选取了几个电路(openMSP430, ae18 _ core, k68 _ core, mips _ core, or1200, M32632, i650)来进行实验,随机选择了几个木马,然后随机地插入到这几个电路中。当电路的规模达到一定程度时,HTChecker 会受限于内存容量的影响而停止运行。为了解决这个问题,可借助于基于磁盘的 neo4j 图数据库存储待检测电路。本实验中的待检测电路都是使用 neo4j 来存储。

实验结果如图 5 所示,所有的木马都可以被 HTChecker 精确地检测到。从图 5 可以看到,HT-

Checker 检测木马花费最长的时间在 3min 左右,意味着本研究提出的方法是可以用在实际的 VLSI 的木马检测中的。

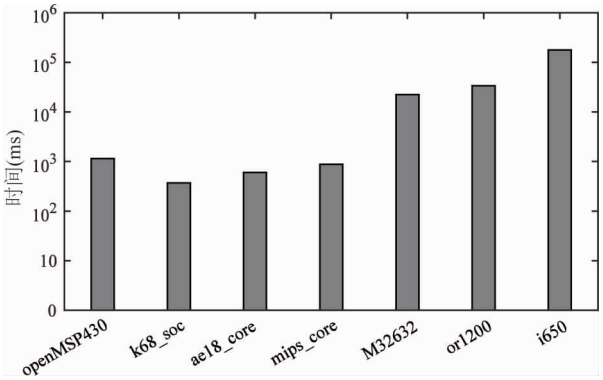

图 5 使用 neo4j 存储大规模电路 HTChecker 检测木马所用时间

通过上面的实验证明了 HTChecker 的有效性。在所有的实验中,HTChecker 都可以精确地找出木马的位置。对于大规模的电路,本文借助于图数据库进行存储,消除了由于电路规模大而导致 HTChecker 失效的问题。对于所有已知特征的木马,HTChecker 都可以快速精确地找出木马。

5 结 论

本文提出了基于静态特征的硬件木马检测方法 HTChecker,该方法利用子图同构的思想进行木马的检测。另外,基于 VF2 算法提出了木马检测算法 TDA。与 VF2 相比,TDA 大大缩短了检测木马所用时间,提高了木马检测的效率。还有,通过利用基于磁盘的图数据库 neo4j 来存储芯片电路,有效地解决了大规模的芯片受限于机器内存的影响导致 HTChecker 没法进行检测的问题,实验证明对于大规模的芯片,HTChecker 仍然可以高效地进行木马检测。

参考文献

[1] Tehranipoor M, Koushanfar F. A survey of hardware Trojan taxonomy and detection. *IEEE Design & Test*, 2010, 27(1):10-25

[2] Fournaris A P, Koufopavlou O. CRT RSA hardware architecture with fault and simple power attack countermeasures. In: Proceedings of Euromicro Conference on Digital System Design, Izmir, Turkey, 2012. 661-667

[3] Anderson R, Bond M, Clulow J, et al. Cryptographic processors: A survey. *Proceedings of the IEEE*, 2006, 94(2):357-369

[4] Bar-El H, Choukri H, Naccache D, et al. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 2004, 94(2):370-382

[5] Fern N, San I, Kaya E, et al. Hardware trojans in incompletely specified on-chip bus systems. In: Proceedings of Conference on Design, Automation & Test in Europe, Dresden, Germany, 2016. 527-530

[6] Salmani H, Tehranipoor M, Plusquellic J. New design strategy for improving hardware Trojan detection and reducing Trojan activation time. In: Proceedings of IEEE International Workshop on Hardware-Oriented Security and Trust, San Francisco, USA, 2009. 66-73

[7] Wolff F, Papachristou C, Bhunia S, et al. Towards Trojan-free trusted ICs: problem analysis and detection scheme. In: Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 2008. 1362-1365

[8] Jha S, Jha S K. Randomization based probabilistic approach to detect trojan circuits. In: Proceedings of High Assurance Systems Engineering Symposium, Nanjing, China, 2008. 117-124

[9] Jin Y, Makris Y. Hardware Trojan detection using path delay fingerprint. In: Proceedings of IEEE International Workshop on Hardware-Oriented Security and Trust, Anaheim, USA, 2008. 51-57

[10] Xiao K, Zhang X, Tehranipoor M. A clock sweeping technique for detecting hardware trojans impacting circuits delay. *IEEE Design & Test*, 2013, 30(2): 26-34

[11] Li J, Lach J. At-speed delay characterization for IC authentication and Trojan horse detection. In: Proceedings of IEEE International Workshop on Hardware-Oriented Security and Trust, Anaheim, USA, 2008. 8-14

[12] Cha B, Gupta S K. Trojan detection via delay measurements: A new approach to select paths and vectors to maximize effectiveness and minimize cost. In: Proceedings of Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France, 2013. 1265-1270

[13] Narasimhan S, Du D, Chakraborty R S, et al. Hardware Trojan detection by multiple-parameter side-channel analysis. *IEEE Transactions on Computers*, 2013, 62(11): 2183-2195

[14] Agrawal D, Baktir S, Karakoyunlu D, et al. Trojan detection using IC fingerprinting. In: Proceedings of 2007 IEEE Symposium on Security and Privacy, Berkeley, USA, 2007. 296-310

[15] Rad R M, Wang X, Tehranipoor M, et al. Power supply signal calibration techniques for improving detection resolution to hardware Trojans. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, San Jose, USA, 2008. 632-639

[16] Cao Y, Chang C H, Chen S. A cluster-based distributed

active current sensing circuit for hardware Trojan detection. *IEEE Transactions on Information Forensics and Security*, 2014, 9(12): 2220-2231
[17] Narasimhan S, Yueh W, Wang X, et al. Improving IC security against Trojan attacks through integration of security monitors. *IEEE Design & Test of Computers*, 2012, 29(5): 37-46
[18] Jin Y. Design-for-security vs. design-for-testability: A case study on dft chain in cryptographic circuits. In: Proceedings of IEEE Computer Society Annual Symposium on VLSI, Florida, USA, 2014. 19-24
[19] Rai D, Lach J. Performance of delay-based Trojan detection techniques under parameter variations. In: Proceedings of IEEE International Workshop on Hardware-Oriented Security and Trust, San Francisco, USA, 2009. 58-65
[20] Al-Anwar A, Alkabani Y, El-Kharashi M W, et al. Hardware trojan protection for third party IPs. In: Proceedings of 2013 Euromicro Conference on Digital System Design, Santander, Spain, 2013. 662-665
[21] 唐永康,王建业,李少青等. 基于红外光谱分析的硬件木马检测方法. 计算机工程与应用,2017, 53(12): 110-115 + 132
[22] 李雄伟,王晓晗,张阳等. 基于最大间距准则的硬件木马检测方法研究. 华中科技大学学报(自然科学版), 2016,44(4):23-27
[23] 李雄伟,王晓晗,张阳等. 基于多旁路综合分析的硬件木马检测方法. 计算机仿真,2015, 32(3):216-219
[24] 赵志勋,倪林,李少青. 硬件木马电路功耗的检测方法. 北京邮电大学学报,2015, 38(4):128-132
[25] 王力纬,宏伟,姚若河. 基于旁路分析的硬件木马检测方法. 华南理工大学学报(自然科学版),2012,40(6):6-10
[26] 赵毅强,杨松,何家骥等. 基于主成分分析的硬件木马检测方法. 华中科技大学学报(自然科学版),2015,43(8):66-69
[27] Neo Technology. Neo4j Graph Database. http://neo4j.org/: Neo4j, 2015
[28] Ullmann J R. An algorithm for subgraph isomorphism. *Journal of the ACM*, 1976, 23(1): 31-42
[29] McKay B D. Practical graph isomorphism. *Journal of Symbolic Computation*, 1981, 60(1): 94-112
[30] Cordella L P, Foggia P, Sansone C, et al. Performance evaluation of the VF graph matching algorithm. In: Proceedings of the 10th International Conference on Image Analysis and Processing, Venice, Italy,1999. 1172-1177
[31] Cordella L P, Foggia P, Sansone C, et al. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004, 26(10): 1367-1372
[32] OpenCores. http://opencores.org: OpenCores, 1999
[33] Brglez F, Bryan D, Kozminski K. Combinational profiles of sequential benchmark circuits. In: Proceedings of IEEE International Symposium on Circuits and Systems, Portland, USA, 1989, 3: 1929-1934

Detecting hardware Trojans based on static characteristics

Zhao Yuehui [*], Shen Haihua [**]
([*] School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing 101408)
([**] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

Abstract

The hardware security situation that the globalization of integrated-circuit (IC) design and manufacture makes IC easy to implant hardware Trojans (HT) so it has the potential risks of malicious attacks, as well as the technical characteristics of present method for hardware Trojan detection, were analyzed in detail, and based on this, a novel hardware Trojan detection scheme based on static characteristics of Trojans, named HTChecker, was proposed. Based on the static features of Trojans, the HTCheckes uses the subgraph isomorphism technique to detect hardware Trojans. Compared with other schemes, the HTChecker can quickly and accurately detect hardware Trojans. For not to be limited to the capacity of memory, the HTCheckr uses the graph database to store large-scale circuits, thus it can effectively detect very large-scale circuits. The HTChecker was evaluated with random mixtures of Trojans and the circuits from ISCAS'89 benchmarks and OpenCores. The experiments show that HTChecker can detect Trojans quickly and accurately without "Golden Chip" and it can efficiently cope with actual VLSI designs.

Key words: hardware security, static characteristics, hardware Trojan (HT), Trojan detection, subgraph isomorphism