Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.31, NO.2, Apr. 2021 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2021.31.2.187

비트스트림 역공학을 활용한 FPGA 하드웨어 악성기능 탐지 기법 연구*

조 민 기,[1†] 정 세 연,[1] 권 태 경[2‡]
[1,2]연세대학교 정보대학원 정보보호연구실 (대학원생, 교수)

A Study of FPGA Hardware Trojan Detection Using Bitstream Reverse-Engineering*

Mingi Cho,[1†] Seyeon Jeong,[1] Taekyoung Kwon[2‡]
[1,2]Information Security Lab, GSI, Yonsei University (Graduate student, Professor)

요 약

FPGA 시스템 개발 과정에서 삽입될 수 있는 하드웨어 악성기능은 전력 소모, 정보 유출 등 심각한 피해를 초래할 수 있으므로 이에 대한 탐지가 필수적으로 수행되어야 한다. 대표적인 하드웨어 악성기능 탐지 기법으로는 부채널 분석과 로직 테스팅 기법이 있으나, 비트스트림 형태로 삽입되는 하드웨어 악성기능의 경우에는 이러한 방법으로 탐지하기 어렵다. 따라서 기존 탐지 기법을 보완하기 위해서는 FPGA 비트스트림으로부터 구현된 회로의 기능을 역공학하여 하드웨어 악성기능을 탐지할 필요가 있다. 본 논문에서는 하드웨어 악성기능 탐지를 위해 FPGA 비트스트림을 역공학하는 방법을 제안하고, 실험을 통해 본 논문에서 제안하는 방법의 성능을 평가한다.

ABSTRACT

The hardware Trojan that can be loaded into an FPGA-based system during the development phase is a serious security problem. The conventional hardware Trojan detection techniques, such as side channel analysis and logic testing, are widely used, however, a hardware Trojan which is inserted as the form of the bitstream can evade those detection mechanisms. Therefore, to detect hardware Trojan, a reverse-engineering of bitstream has to be considered to analysis the functionality of the implemented circuit. In this study, we examine the method for reverse-engineering the LUT information from the FPGA bitstream to the form of Boolean equation, and evaluate the performance of the proposed method.

Keywords: FPGA, Hardware Trojan

I. 서 론

하드웨어 악성기능은 FPGA(Field programmable gate array) 기반 시스템 개발 과정 중 다양한 경로를 통해 유입될 수 있다. 이는 HDL (Hardware description language)을 이용한 시스템 설계 시 악의적인 개발자에 의해 직접 삽입되거나, 신뢰되지 않은 IP(Intellectual property) core의 사용으로 인해 전체 회로에 유입될 수 있다[1]. 또한 FPGA의 재구성 기능을 악용하여 하드웨어 악성기능이 구현된 회로가 비트스트림 형태로 FPGA에 재탑재되거나, 비트스트림의 악의적 수정을 통해 하드웨어 악성기능이 삽입될 수 있다[2, 3]. 하드웨어 악성기능은 전력 소모, 정보 유출 등 심각한 피해를 초래할 수 있어 이에 대한 탐지가 반드시 수행되

Received(10. 30. 2020), Modified(1st: 02. 23. 2021, 2nd: 03. 17. 2021), Accepted(03. 22. 2021)
* 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다(UD190016ED).
† 주저자, imgc@yonsei.ac.kr
‡ 교신저자, taekyoung@yonsei.ac.kr(Corresponding author)

어야 한다.

대표적인 하드웨어 악성기능 탐지 기법으로는 부채널 분석, 로직 테스팅 등이 있으나, 비트스트림의 형태로 삽입되는 하드웨어 악성기능은 이러한 탐지를 우회할 수 있다. 특히 비트스트림을 직접 수정하여 하드웨어 악성기능을 삽입하는 경우, 악성기능 삽입과 관련된 어떠한 로그 파일도 생성되지 않아 탐지가 어렵다[3]. 이러한 경우를 모두 고려하여 하드웨어 악성기능을 탐지하기 위해서는 비트스트림을 역공학하여 구현된 회로의 기능을 확인할 필요가 있다.

최근 FPGA 비트스트림으로부터 회로 연결 정보를 추출하거나 사용된 자원의 정보를 넷리스트로 복구하는 연구 사례가 발표되었으나[4, 5, 6], 구현된 회로의 기능을 복구하지 않아 역공학 결과로부터 하드웨어 악성기능을 탐지하기는 어렵다. 따라서 본 논문에서는 하드웨어 악성기능 탐지를 위해 FPGA 비트스트림으로부터 회로의 기능을 역공학하여 넷리스트 수준으로 복구하는 방법을 제시한다. 또한 하드웨어 역공학 결과 생성된 넷리스트를 활용하여 하드웨어 악성기능 탐지를 수행하여 비트스트림으로부터 하드웨어 악성기능 탐지가 가능한 것을 보여준다.

II. 관련 연구

2.1 하드웨어 악성기능 탐지

하드웨어 악성기능의 위협에 대응하기 위해 하드웨어 악성기능 탐지 기술이 연구되고 있다. Waksman 등[11]은 하드웨어 악성기능 탐지를 위해 하드웨어 악성기능에서 나타나는 특징을 분석하였고, 분석한 결과를 바탕으로 거의 사용되지 않는 와이어를 탐색하는 탐지 방법을 제안하였다. Zhang 등[12]은 하드웨어 악성기능 탐지를 위해 사용되지 않는 회로를 찾기 위한 기능적 검증 검사(Functional verification test)를 제안하였다. 하지만 제안된 방법들은 게이트 수준의 넷리스트를 대상으로 하므로 비트스트림을 직접 수정하거나[3], FPGA의 비트스트림 재구성 기능을 통해 삽입되는 하드웨어 악성기능[2]은 탐지하지 못하는 한계점이 있다.

2.2 비트스트림 역공학

비트스트림에 구현된 로직을 분석하기 위해 비트스트림 역공학 기술이 연구되고 있다. Note 등[4]은 상호연관 알고리즘을 사용하여 비트스트림의 대부분을 차지하는 자원 간의 연결 정보를 역공학하는 Debit을 제안하였다. Benz 등[5]은 Debit의 한계점을 개선한 BIL을 공개하였다. BIL은 FPGA의 모든 자원 정보가 있는 XDLRC를 활용하여 Debit이 역공학하지 못한 HCLK 타일의 PIP도 역공학하였다. 하지만 BIL도 여전히 CLB 타일 등의 역공학하지 못하는 타일이 존재하고 LUT에 저장되는 로직 정보를 역공학하지 않는다. Bit2NCD[6]는 FPGA 내부 자원과 비트스트림과의 관계를 저장하여 비트스트림을 역공학하였다. Zhang 등[9]은 비트스트림을 RTL 수준의 코드로 역공학하는 방법을 제안하였다.

III. FPGA 개요

3.1 FPGA 구조

FPGA는 다양한 기능을 하는 타일들의 배열 구조로 이루어져 있다. 주요 타일로는 INT(Interconnect) 타일, CLB(Configurable logic block) 타일 등이 있다. INT 타일에는 회로 연결 정보를 나타내는 자원인 PIP(Programmable interconnect point)가 존재한다. CLB 타일은 플립플롭, 메모리, LUT(Look-up table) 등 회로 구현에 관여하는 자원을 포함하는 slice로 이루어져 있다. Xilinx Virtex-5 기기를 기준으로 하나의 CLB 타일에는 2개의 slice가 존재하며, 각 slice에는 6개의 입력 핀(A1~A6)을 가지는 LUT가 4개(A~D) 존재한다.

3.2 FPGA 비트스트림 구조

FPGA 비트스트림은 합성, 맵핑, 배치 및 결선 단계를 통해 생성되는 최종 결과물로써 FPGA에 탑재되어 회로 구현에 필요한 자원들을 설정한다. 비트스트림은 크게 헤더, 구성 명령, 구성 데이터, 체크코드로 이루어져 있으며, 구성 데이터가 실제 자원을 설정한다. 구성 데이터를 이루는 기본 단위는 프레임이며, 하나의 프레임은 Fig. 1.과 같이 32-bit 길이의 워드 41개로 구성된다. 각 프레임은 블록 타입, 행 구분(상, 하), 행 번호, 열 번호, 프레임 번호로 이루어진 주소를 가진다.

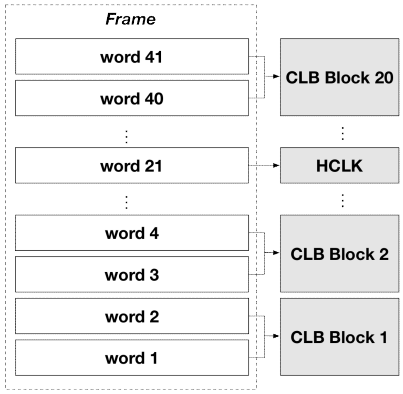

Fig. 1. Configuration data frame

구성 데이터에서 서로 인접한 INT 타일과 CLB 타일은 하나의 CLB 블록으로 구분되며 각 CLB 블록은 총 36개의 프레임으로 구성된다. 각 프레임은 Fig. 1.과 같이 20개의 CLB 블록을 구성하며 2개의 워드가 1개의 CLB 블록에 대응된다. 36개 프레임 중 0~25번 프레임은 INT 타일, 26~ 35번 프레임은 CLB 타일에 대한 데이터를 포함하고 있다.

IV. 비트스트림 역공학

본 장에서는 하드웨어 악성기능 탐지를 위한 비트스트림 역공학 기술을 설명한다. 하드웨어 악성기능은 악의적인 공격자가 비트스트림을 직접 변조하여 삽입할 수 있으므로 역공학을 통한 악성기능 탐지가 필요하다. FPGA 제조사가 비트스트림의 형식을 공개하지 않기 때문에 비트스트림에서 LUT 정보와 PIP 정보를 복원하는 것은 어려운 문제로 여겨진다. 이 문제를 해결하기 위해 우리는 비트스트림 해석 라이브러리인 BIL[5]을 활용하여 비트스트림을 역공학하였다. 비트스트림 역공학은 로직 정보가 저장되는 LUT 정보 역공학과 연결 정보가 저장되는 PIP 정보 역공학으로 이루어진다.

4.1 LUT 정보 역공학

개발자가 구현한 회로는 FPGA 상에서 여러 개의 LUT에 나누어져 구현된다. 각 LUT에는 진리표의 출력에 해당하는 INIT 속성값이 저장되며 이는 구성 데이터 영역에서 64-bit 길이로 나타난다. LUT에 구현된 회로는 문서 형식의 넷리스트인 XDL(Xilinx design language) 상에서 논리식의 형태로 표현된다. 따라서 본 논문에서는 LUT가 구현한 회로의 기능을 파악하기 위해 FPGA의 비트스트림이 주어졌을 때 해당 비트스트림으로부터 LUT 정보를 XDL 형태의 논리식으로 역공학하는 것을 목표로 한다. 또한 본 논문에서는 Xilinx 사의 Virtex-5를 대상으로 역공학 연구를 진행한다.

4.1.1 LUT 구성 데이터 오프셋 추출

LUT 정보를 역공학하기 위해서는 먼저 각 LUT에 대응되는 구성 데이터의 오프셋을 알아야 한다. 이를 위해서 베릴로그 코드를 이용하여 특정 LUT에 서로 다른 INIT 속성값이 저장되도록 구현한 후 비트스트림을 생성하여 각 비트스트림의 구성 데이터 영역을 비교하는 방법을 사용한다. 이때 해당 LUT에 대응되는 64-bit 구성 데이터의 위치를 정확히 알기 위해 Table 1.의 INIT 속성값을 사용한다.

특정 위치의 LUT에 대하여 총 66개의 비트스트림을 생성한 후 구성 데이터 영역을 모두 비교한 결과, 하나의 LUT에 대한 구성 데이터는 1개의 프레임에 16-bit 씩 총 4개의 프레임에 걸쳐 나타나는 것을 확인하였다. 또한 X 좌표가 홀수인 slice 내의 LUT는 해당 CLB 블록의 26-29번 프레임에, X

Table 1. Used INIT attribution values

Case	INIT attribution value
1	0x0000000000000000
2	0x0000000000000001
3	0x0000000000000002
4	0x0000000000000004
…	
65	0x8000000000000000
66	0xFFFFFFFFFFFFFFFF

LUT	Division	Row	Major	Minor		Offset	
				Start	End	Start	End
X00Y00_A	LOWER	1	1	32	35	0	1
X00Y00_B	LOWER	1	1	32	35	2	3
X00Y00_C	LOWER	1	1	32	35	4	5
X00Y00_D	LOWER	1	1	32	35	6	7
			…				
X75Y79_A	UPPER	1	48	26	29	156	157
X75Y79_B	UPPER	1	48	26	29	158	159
X75Y79_C	UPPER	1	48	26	29	160	161
X75Y79_D	UPPER	1	48	26	29	162	163

Fig. 2. LUT configuration data offset table

좌표가 짝수인 slice 내의 LUT는 32-35번 프레임에 나타나는 것을 확인하였다. 이러한 특성을 바탕으로 Fig. 2.와 같이 모든 LUT에 대한 구성 데이터 오프셋을 계산한다.

4.1.2 사용된 입력 핀 개수 역공학

LUT에 구현된 회로를 정확히 알기 위해서는 LUT의 구성 데이터뿐만 아니라 해당 LUT가 사용하고 있는 입력 핀 또한 알아야 한다. LUT 입력 핀가 사용됐는지는 각 핀과 연결된 PIP의 사용 여부를 통해 알 수 있으므로 입력 핀과 연결되는 PIP의 사용 여부도 함께 역공학해야 한다. LUT의 각 입력 핀은 PIP를 통해 INT 타일과 연결되어 있으므로, 입력 핀과 연결된 PIP가 사용되었다면 해당 입력 핀이 사용된 것으로 간주할 수 있다. 이러한 사실을 바탕으로 입력 핀과 연결된 PIP의 비트 오프셋과 값을 구하여 저장하고, 역공학 대상 비트스트림에서 저장된 오프셋과 값이 발견되면 해당 PIP와 연결된 입력 핀이 사용된 것을 알 수 있다. LUT 구성 데이터와 마찬가지로 PIP의 비트 오프셋과 값 또한 구성 데이터에서 나타나는 것을 찾아 저장한다. 이를 위해 FPGA 내의 사용 가능한 모든 자원이 명시된 XDLRC(Xilinx design language resource) 파일을 활용하여 역공학에 필요한 PIP를 찾은 후 각 PIP를 XDL로 구현하고 비트스트림을 생성한다. 이때 XDL 상에서 하나의 PIP만 사용하도록 구현하여 구성 데이터에 오직 해당 PIP에 대한 비트 오프셋만 나타나도록 한다.

FPGA에서 같은 종류의 타일은 모두 같은 자원으로 구성되어 있다. 따라서 특정 INT 타일에서 찾은 PIP의 비트 오프셋과 값은 다른 INT 타일에도 동일하게 적용이 가능하므로 모든 CLB 블록에 대하여 같은 과정을 반복하지 않아도 역공학이 가능하다.

4.1.3 INIT 속성값 복구

FPGA 비트스트림 생성 과정 중 배치 및 결선 단계에서는 최적화된 신호 전달 경로를 설정한다. 이로 인해 합성 단계에서 결정된 LUT 입력의 결선 상태가 변경되어 FPGA에 배치되는 경우가 발생한다. 이 경우 추출한 LUT의 구성 데이터가 실제 INIT 속성값과 달라질 수 있다. 또 사용되는 입력 개수에 따라 INIT 속성값이 64-bit보다 짧은 경우에도 구성 데이터에서는 64-bit로 변환되어 나타나기 때문에 추출한 LUT 구성 데이터에서 INIT 속성값을 계산할 필요가 있다.

이를 해결하기 위해 INIT 속성값과 비트스트림의 맵핑 관계를 활용한다. INIT 속성값의 각 비트가 구성 데이터에 맵핑되는 위치를 알기 위해 하나의 비트만 1로 설정된 INIT 속성값을 저장하는 LUT를 구현하여 사용한다. 예를 들어, A5와 A6 입력 핀이 사용되는 경우 INIT 속성값은 4-bit 길이를 가지므로 각 인덱스의 비트를 1로 설정한 INIT 속성값을 저장하는 LUT를 구현한다. 각 비트스트림에서 LUT의 구성 데이터를 추출하면 Table 2.와 같이 각 비트와 구성 데이터의 맵핑 관계를 알 수 있다. 이러한 맵핑 관계를 파악하여 추출한 구성 데이터에서 INIT 속성값을 계산한다.

Table 2. Example of INIT attribution -configuration data mapping table

Byte / INIT	1	2	3	4	5	6	7	8
0001	0xF0	0x00	0xF0	0x00	0xF0	0x00	0xF0	0xF0
0010	0x0F	0x00	0x0F	0x00	0x0F	0x00	0x0F	0x0F
0100	0x00	0xF0	0x00	0xF0	0x00	0xF0	0x00	0x00
1000	0x00	0x0F	0x00	0x0F	0x00	0x0F	0x00	0x00

4.1.4 논리식 변환

최종적으로 해당 LUT의 논리식을 도출하기 위해 추출한 구성 데이터로부터 계산한 INIT 속성값과 역공학한 입력 핀 정보를 활용하여 진리표를 생성한 후 이를 논리식으로 변환한다. XDL 상에서는 논리 연산 기호를 Table 3.과 같이 사용하므로 역공학 결과로 얻은 논리식을 XDL 형태로 변환한다.

Table 3. Boolean operator in XDL

Operation	General	XDL
AND	∧	*
OR	∨	+
NOT	¬	~

4.2 PIP 정보 역공학

하드웨어 악성기능을 효과적으로 탐지하기 위해서는 LUT에 저장되는 로직 정보뿐만 아니라 PIP 정

보 또한 필요하다. PIP 역공학을 위해서는 BIL에서 제공하는 비트스트림 역공학 도구를 개선한 도구인 [7]를 별도의 수정 없이 사용하였다. [7]은 다음과 같이 세 부분에 대해 BIL 개선하였다. 첫 번째로 구성 데이터베이스 구축 방법이 개선되었다. 기존 BIL은 한 쌍의 비트스트림과 XDL을 사용하여 구성 데이터베이스를 구축하기 때문에 데이터베이스 구축에 사용되는 데이터에 따라 역공학 결과가 달라진다. 이를 해결하기 위해 [7]에서는 여러 쌍의 비트스트림 파일과 XDL 파일을 사용한다.

두 번째로 *not isolated PIP*를 복구한다. Not isolated PIP는 BIL의 상호상관 알고리즘 수행 결과 하나의 비트스트림 구성 비트에 대해 2개 이상의 PIP가 남아있는 PIP이다. 항상 같이 사용되는 PIP의 경우 하나의 구성 비트를 공유할 수 있으므로 일대일 관계가 나타나지 않는 not isolated PIP는 별도의 데이터베이스를 구축한다.

세 번째로 INT 타일 내의 *zeroed PIP*를 복구한다. Zeroed PIP는 비트스트림에 저장되지 않는 PIP를 의미하며, 비트스트림에 저장되지 않으므로 비트스트림에 나타난 구성 비트를 활용하는 BIL은 zeroed PIP를 역공학하지 못한다. Zeroed PIP는 같은 타일 내의 다른 PIP와 연결되어 사용되기 때문에 zeroed PIP와 연결되는 PIP 정보를 별도의

Table 4. List of features by category.

Category	Feature	Type	Description
A	**Boolean Equation**		
A.1	Equation_hash	Numeric	Hash of Boolean equation
A.2	Equation_length	Numeric	Length of Boolean equation
A.3	#Equation_operator	Numeric	Number of operators
A.4	#Equation_operand	Numeric	Number of operands
B	**Truth Table**		
B.1	#LUT_output_1	Numeric	Number of output 1s
B.2	Rare_condition	Numeric	RC or non-RC
B.3	LUT_output	Numeric	Output of truth table
C	**Input**		
C.1	#Used_inpin	Numeric	Number of used inpins
C.2	Inpin_1_conn_resource	Property	Resource connected to the inpin1
C.3	Inpin_2_conn_resource	Property	Resource connected to the inpin2
C.4	Inpin_3_conn_resource	Property	Resource connected to the inpin3
C.5	Inpin_4_conn_resource	Property	Resource connected to the inpin4
C.6	Inpin_5_conn_resource	Property	Resource connected to the inpin5
C.7	Inpin_6_conn_resource	Property	Resource connected to the inpin6
C.8	#Inpin_unused	Numeric	Number of unused inpins
C.9	#Inpin_conn_RC_LUT	Numeric	Number of RC LUTs connected to the inpins
C.10	#Inpin_conn_non_RC_LUT	Numeric	Number of non-RC LUTs connected to the inpins
C.11	#Inpin_conn_FF	Numeric	Number of flip-flops LUTs connected to the inpins
C.12	#Inpin_conn_IOB	Numeric	Number of IOBs connected to the inpins
C.13	#Inpin_conn_etc	Numeric	Number of other resources connected to the inpins
D	**Output**		
D.1	Outpin_conn_LUT	Property	Resource connected to the LUT outpin
D.2	Outpin_conn_MUX	Property	Resource connected to the LUT outpin
D.3	Outpin_conn_FF	Property	Resource connected to the LUT outpin
D.4	#Outpin_conn_LUT	Numeric	Number of LUT connected to the outpin
D.5	#Outpin_conn_MUX	Numeric	Number of LUT connected to the outpin
D.6	#Outpin_conn_FF	Numeric	Number of LUT connected to the outpin
D.7	#Outpin_conn_RC_LUT	Numeric	Number of LUT connected to the outpin
E	**Flip-flop**		
E.1	Conn_FF_location	Property	Location information of connected flip-flop
E.2	#Conn_FF_influence	Numeric	Number of resources affected by the connected flip-flop

데이터베이스에 저장하여 복구를 진행한다.

V. 하드웨어 악성기능 탐지

하드웨어 악성기능 탐지 방법은 기계학습 기반 하드웨어 악성기능 탐지 기술[8]을 기반으로 설계된다. 하드웨어 악성기능 탐지에 사용되는 특징은 논리식 특징(A), 진리표 특징(B), LUT 입력 특징(C), LUT 출력 특징(D), flop-flop 특징(E)으로 나누어진다. 논리식 특징(A), 진리표 특징(B), 사용된 입력 핀의 수 특징(C.1)은 LUT 역공학을 통해 얻으며, 나머지 특징들은 PIP 역공학을 통해 얻는다.

본 연구에서는 [8]에서 사용된 31개의 하드웨어 악성기능 특징들 중 Table 4.와 같이 29개의 특징을 선정하여 사용한다. [8]에서 사용되는 특징 중 LUT_output_0와 Inpin_conn_resource_ratio는 본 연구에서 사용하지 않았으며 그 이유는 다음과 같다. [8]에서 LUT_output_0와 LUT_output_1을 모두 사용하고 있지만 LUT_output_0이 결정되면 LUT_output_1 또한 결정되므로 이 두 개의 특징은 하나의 특징으로 볼 수 있다. 따라서 본 연구에서는 두 개의 특징 중 LUT_output_1만을 사용한다. Inpin_conn_resource_ratio 특징은 역공학 실험 결과 약 1%의 매우 낮은 수준의 역공학 정확도가 나타났기 때문에 본 연구에서는 사용하지 못하였다.

논리식 특징(A)은 LUT 역공학 결과 변환된 논리식에서 추출되며, 논리식의 해시값(A.1), 길이(A.2), 연산자 수(A.3), 피연산자 수(A.4)를 특징으로 사용한다. 진리표 특징(B)은 LUT 구성 데이터에서 얻은 INIT 속성값을 사용하며, 진리표 출력에 포함된 1의 수(B.1), RC LUT 여부(B.2), 진리표 출력(B.3)이 특징으로 사용된다.

LUT 입력 특징(C)은 총 13개가 있으며 사용된 입력 핀의 수(C.1), 각 입력 핀과 연결된 자원 종류(C.2-C.7), 사용되지 않는 입력 핀의 수(C.8), 입력 핀과 연결된 자원의 유형별 개수(C.9 -C.13)가 있다. LUT 출력 특징(D)은 총 7개가 있으며 출력 핀과 연결된 자원의 종류(D.1-D.3), 출력 핀과 연결된 자원별 개수(D.4- D.7)가 있다. 마지막으로 flip-flop 특징(E)은 총 2개가 사용되며 flip-flop의 위치(E.1), flop-flop에 영향을 받는 자원의 수(E.2)가 있다.

VI. 실험 결과

6.1 데이터 셋 및 실험 환경

본 연구는 Xilinx 사의 Virtex-5 기기를 대상으로 하였으며 Xilinx ISE를 사용하여 비트스트림을 생성하였다. 제안 기술의 성능을 평가하기 위해

Table 5. Result of HT LUT detection on TrustHub benchmarks.

Benchmarks	#Utilized LUTs	#HT LUTs	Forward Detection		Backward Detection	
			TPR	FPR	TPR	FPR
AES_T400	11,820	21	1.0000	0.0000	1.0000	0.0000
AES_T700	11,519	21	1.0000	0.0000	1.0000	0.0000
AES_T800	11,593	84	1.0000	0.0000	1.0000	0.0000
AES_T900	11,775	21	1.0000	0.0000	1.0000	0.0000
AES_T1000	11,539	21	1.0000	0.0000	1.0000	0.0000
AES_T1100	11,610	84	1.0000	0.0000	1.0000	0.0000
AES_T1200	11,792	21	1.0000	0.0000	1.0000	0.0000
AES_T1600	11,451	84	1.0000	0.0000	1.0000	0.0000
AES_T1700	12,064	21	1.0000	0.0000	1.0000	0.0000
B19_T300	26,297	144	1.0000	0.0025	0.9861	0.0025
B19_T400	26,365	143	1.0000	0.0026	1.0000	0.0024
B19_T500	25,594	46	1.0000	0.0029	0.9783	0.0025
BasicRSA_T100	1,011	6	0.8333	0.0139	0.8333	0.0139
BasicRSA_T300	1,043	6	0.8333	0.0135	0.8333	0.0116
BasicRSA_T400	960	12	1.0000	0.0179	1.0000	0.0137
Total	**186,433**	**735**	**0.9973**	**0.0014**	**0.9932**	**0.0012**

TrustHub[10]에서 제공하는 15개의 샘플을 사용하였다. 15개의 샘플은 AES, BasicRSA, B19 그룹으로 나뉘며 각 샘플에서 사용된 LUT 수와 하드웨어 악성기능(HT) LUT 수는 Table 5.에서 볼 수 있다.

6.2 하드웨어 악성기능 탐지

Table 5.는 TrustHub 벤치마크를 대상으로 하드웨어 악성기능을 탐지한 결과를 보여준다. 정방향 탐지(Forward detection)는 소스코드로부터 추출된 특징을 사용하고, 역방향 탐지(Backward detection)는 비트스트림으로부터 역공학된 특징을 사용한다. 양방향 모두 Table 4.에 선정된 29개의 특징을 사용하였다. 실험 결과 정방향 및 역방향 모두 높은 TPR(True positive rate)이 나타났다. 정방향 탐지는 [8]에서 사용된 31개의 특징 중 29개의 특징을 사용하였으며 99.7%의 TPR과 0.14%의 FPR(False positive rate)이 나타났다. [8]에서는 Xilinx 사의 Artix-7을 대상으로 실험하였기 때문에 [8]의 실험 결과와 직접적인 성능 비교를 하기는 어렵다. Virtex-5를 대상으로 [8]에서 사용한 31개의 특징을 사용하여 정방향 탐지를 수행한 결과 99.6%의 TPR과 0.13%의 FPR이 나타났다. 이를 통해 29개의 특징을 사용하여도 31개의 특징을 모두 사용한 것과 유사한 성능으로 하드웨어 악성기능을 탐지할 수 있는 것을 알 수 있다.

역방향 탐지에서 735개의 하드웨어 악성기능 LUT 중 725개를 탐지하여 99.32%의 TPR을 달성하였고, 동시에 0.12%의 낮은 FPR을 달성하였다. 이는 99.73%의 TPR과 0.14%의 FPR이 나타난 정방향 탐지와 거의 비슷한 수준으로 역공학 기술이 하드웨어 악성기능 탐지에 효과적으로 활용될 수 있는 것을 알 수 있다.

6.3 비트스트림 역공학

비트스트림 역공학 성능을 평가하기 위해 비트스트림에서 추출된 특징과 소스코드에서 추출된 특징을 비교하였다. Table 6.은 비트스트림 역공학을 통해 얻은 특징과 소스코드에서 추출된 특징을 비교한 정확도(Accuracy) 결과를 보여준다. 역공학 정확도는 전체 역공학 대상 자원 개수 중 정확하게 역공학된 자원 개수의 비율을 계산하여 구하였다. 역공학을 통해 얻은 특징은 평균적으로 82.32%의 정확도를 보였다. 전체 특징 카테고리 중 진리표 특징(B)이 96.91%의 가장 높은 정확도를 달성했다. 이는 본 논문에서 제안한 LUT 역공학 기술이 효과적으로

Table 6. Recovery rate of reverse-engineered features.

Category	Feature	Accur.
A	**Boolean Equation**	**76.25%**
A.1	Equation_hash	63.75%
A.2	Equation_length	78.24%
A.3	#Equation_operator	78.52%
A.4	#Equation_operand	84.49%
B	**Truth Table**	**96.91%**
B.1	#LUT_output_1	98.29%
B.2	Rare_condition	98.99%
B.3	LUT_output	93.46%
C	**Input**	**75.99%**
C.1	#Used_inpin	96.24%
C.2	Inpin_1_conn_resource	83.79%
C.3	Inpin_2_conn_resource	83.33%
C.4	Inpin_3_conn_resource	90.59%
C.5	Inpin_4_conn_resource	84.13%
C.6	Inpin_5_conn_resource	91.22%
C.7	Inpin_6_conn_resource	75.09%
C.8	#Inpin_unused	89.77%
C.9	#Inpin_conn_RC_LUT	91.90%
C.10	#Inpin_conn_non_RC_LUT	86.93%
C.11	#Inpin_conn_FF	55.81%
C.12	#Inpin_conn_IOB	97.40%
C.13	#Inpin_conn_etc	42.67%
D	**Output**	**90.48%**
D.1	Outpin_conn_LUT	95.54%
D.2	Outpin_conn_MUX	93.23%
D.3	Outpin_conn_FF	88.55%
D.4	#Outpin_conn_LUT	91.80%
D.5	#Outpin_conn_MUX	92.86%
D.6	#Outpin_conn_FF	77.16%
D.7	#Outpin_conn_RC_LUT	94.19%
E	**Flip-Flop**	**85.17%**
E.1	Conn_FF_location	93.33%
E.2	#Conn_FF_influence	77.02%
Average		**82.32%**

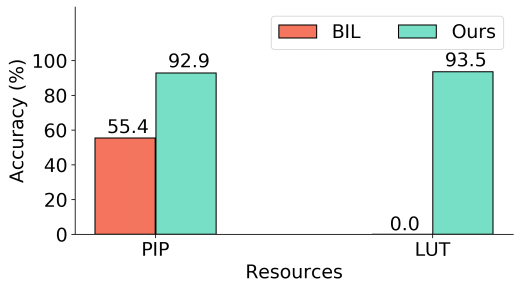

Fig. 3. Result of bitstream reverse-engineering by BIL and our tool.

LUT 구성 데이터를 역공학 할 수 있음을 보여준다. 이에 반해 논리식 해시는 63.75%의 낮은 역공학 정확도를 달성했다. 분석 결과 이는 역공학 대상인 TrustHub 샘플의 논리식에 (A1 and ~A1)과 같이 논리적으로 의미 없는 부분(항상 참)이 포함되어 있기 때문이었다. LUT 구성 데이터에서 추출된 진리표에서 논리식을 생성할 때 항상 참인 논리식 부분을 생성하는 것은 불가능하므로 해당 특징을 정확하게 역공학하지 못하였다. 또한 #Inpin_conn_FF, #Inpin_conn_etc에서 상대적으로 낮은 정확도를 달성하였는데 이는 PIP 역공학에서 해당 자원을 연결하는 모든 PIP를 역공학 하지 못했기 때문이다. 이는 PIP 역공학의 정확도를 높임으로써 해결할 수 있다.

Fig. 3.은 제안된 방법과 BIL의 역공학 결과를 보여준다. PIP 역공학 결과, BIL을 개선한 [7]을 사용한 결과 BIL에 비해 약 68% 더 높은 PIP 역공학 정확도가 나타났다. LUT 역공학 결과, 제안된 방법을 사용하였을 때 약 93.5%의 LUT 자원을 역공학할 수 있었다. 반면 BIL은 LUT 자원을 역공학하지 못하였다. 이를 통해 본 논문에서 제안된 방법이 기존 연구인 BIL과 [7]에서 역공학하지 못한 LUT 자원을 성공적으로 역공학한 것을 알 수 있다.

VII. 결 론

본 논문에서는 Xilinx Virtex-5 비트스트림으로부터 비트스트림 정보를 역공학하는 방법을 제시하였다. 기존 연구인 BIL과는 다르게 PIP 정보뿐만 아니라 LUT 정보 또한 역공학을 수행하였다. 역공학을 통해 추출한 특징은 평균 82.32%의 정확도를 달성하였으며, 진리표와 관련된 특징은 96.91%의 높은 복구 정확도를 보였다. 이를 통해 본 논문에서 제안하는 LUT 정보 역공학 방법이 효과적인 것을 보였다. 또한 역공학 기술을 활용하여 TrustHub 데이터 셋을 대상으로 하드웨어 악성기능 탐지를 수행하였고 그 결과 99.32%의 하드웨어 악성기능 LUT를 탐지할 수 있었다. 이를 통해 비트스트림 역공학 기술이 하드웨어 악성기능 탐지에 효과적으로 활용될 수 있음을 보였다.

References

[1] S. Bhunia, M.S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: Threat analysis and countermeasures," Proceedings of the IEEE, vol. 102, no. 8, pp. 1229 - 1247, Jul. 2014.

[2] A.P. Johnson, S. Patranabis, R.S. Chakraborty, and D. Mukhopadhyay, "Remote dynamic clock reconfiguration based attacks on Internet of Things applications," Proceedings of the Euromicro Conference on Digital System Design, pp. 431 - 438, Aug. 2016.

[3] R.S. Chakraborty, I. Saha, A. Palchaudhuri, and G.K. Naik, "Hardware Trojan insertion by direct modification of FPGA configuration bitstream," IEEE Design & Test, vol. 30, no. 2, pp. 45 - 54, Feb. 2013

[4] J.-B. Note and E. Rannaud, "From the bitstream to the netlist." Proceedings of the International Conference on Field Programmable Gate Array, pp. 264 - 264, Feb. 2008.

[5] F. Benz, A. Seffrin, and S.A. Huss, "BIL: A toolchain for bitstream reverse-engineering," Proceedings of the International Conference on Field Programmable Logic and Applications, pp. 735 - 738, IEEE, Aug. 2012.

[6] Z. Ding, Q. Wu, Y. Zhang, and L. Zhu, "Deriving an NCD file from an FPGA bitstream: Methodology, architecture and evaluation," Microprocessors and

Microsystems, vol. 37, no. 3, pp. 299 - 312, May. 2013.
〔7〕 J. Yoon, Y. Seo, J. Jang, and T. Kwon. "A Study on the BIL bitstream reverse-engineering tool-chain improvement," Journal of the Korea Institute of Information Security & Cryptology, 28(5), pp. 1225-1231, Oct. 2018.
〔8〕 J. Jang, M. Cho, Y. Seo, S. Jeong, and T. Kwon. "A study of machine learning based hardware Trojans detection mechanisms for FPGAs," Journal of Internet Computing and Services, 21(2), pp. 109-119, Apr. 2020.
〔9〕 T. Zhang, J. Wang, S. Guo, and Z. Chen. "A comprehensive FPGA reverse engineering tool-chain: From bitstream to RTL code," IEEE Access, vol. 7, 38379-38389, Feb. 2019.
〔10〕 TrustHub, "TrustHub," https://trust-hub.org/, Feb. 2021.
〔11〕 A. Waksman, M. Suozzo, and S. Sethumadhavan. "FANCI: Identification of stealthy malicious logic using Boolean functional analysis," Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 697-708, Nov. 2013.
〔12〕 J. Zhang, F. Yuan, and Q. Xu. "Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware Trojans," Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, pp. 153-166, Nov. 2014.

〈저자소개〉

조 민 기 (Mingi Cho) 학생회원
2017년: 부산대학교 정보컴퓨터공학과 학사
2017년~현재: 연세대학교 정보대학원 정보보호 석박통합과정
〈관심분야〉 소프트웨어 보안, 시스템 보안, 정보 보안

정 세 연 (Seyeon Jeong) 학생회원
2018년: 대구가톨릭대학교 정보보호학과 학사
2019~현재: 연세대학교 정보대학원 정보보호 석사과정
〈관심분야〉 소프트웨어 보안, 시스템 보안, IoT 보안

권 태 경 (Taekyoung Kwon) 종신회원
1992년: 연세대학교 컴퓨터과학과 학사
1995년: 연세대학교 컴퓨터과학과 석사
1999년: 연세대학교 컴퓨터과학과 박사
1999년~2000년: U.C. Berkeley EECS Post-Doc.
2001년~2013년: 세종대학교 컴퓨터공학과 교수
2007년~2008년: Univ. of Maryland, College Park 교환교수
2013년~현재: 연세대학교 정보대학원 교수
〈관심분야〉 암호 프로토콜, Usable Security, 소프트웨어/시스템 보안, 기계학습과 보안