J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr
Copyright ⓒ 2020 KSII

FPGA의 Hardware Trojan 대응을 위한 기계학습 기반 탐지 기술 연구☆

A Study of Machine Learning based Hardware Trojans Detection Mechanisms for FPGAs

장 재 동[1] 조 민 기[1] 서 예 지[1] 정 세 연[1] 권 태 경[1*]
Jaedong Jang Mingi Cho Yezee Seo Seyeon Jeong Taekyoung Kwon

요 약

FPGA는 초기 제작 후 다시 설계 할 수 있는 반도체로 신호 처리, 자동차 산업, 국방 및 군사 시스템 등과 같은 다양한 임베디드 시스템 분야에서 사용된다. 하지만 하드웨어 설계의 복잡성이 증가하고 설계 및 제조 과정이 세계화됨에 따라 하드웨어에 삽입되는 하드웨어 악성기능에 대한 우려가 커져가고 있다. 이러한 위협에 대응하기 위해 많은 탐지 방법들이 제시되었지만, 기존 방법 대부분은 IC칩을 대상으로 하고 있어 IC칩과 구성요소가 다른 FPGA에 적용하기 어렵다. 또한 FPGA 칩을 대상으로 하는 하드웨어 악성기능 탐지 연구는 거의 이루어지지 않고 있다. 본 논문에서는 이러한 문제점을 해결하기 위해 FPGA의 LUT-level netlist에서 나타나는 하드웨어 악성기능의 정적인 특징을 기계학습을 통해 학습하여 하드웨어 악성기능을 탐지하는 방법을 제시한다.

☞ 주제어 : FPGA, 하드웨어 보안, 하드웨어 트로이 목마

ABSTRACT

The FPGAs are semiconductors that can be redesigned after initial fabrication. It is used in various embedded systems such as signal processing, automotive industry, defense and military systems. However, as the complexity of hardware design increases and the design and manufacturing process globalizes, there is a growing concern about hardware trojan inserted into hardware. Many detection methods have been proposed to mitigate this threat. However, existing methods are mostly targeted at IC chips, therefore it is difficult to apply to FPGAs that have different components from IC chips, and there are few detection studies targeting FPGA chips. In this paper, we propose a method to detect hardware trojan by learning the static features of hardware trojan in LUT-level netlist of FPGA using machine learning.

☞ keyword : FPGA, Hardware Security, Hardware Trojan

1. 서 론

FPGA (Field-Programmable Gate Array)는 초기 제작 후에 회로의 변경이 불가능한 일반적인 반도체와 다르게 제작 후 회로 수정이 가능한 반도체이다. 이러한 특징으로 인해 FPGA는 회로 설계에 사용되는 핵심적인 매체가 되었으며, 임베디드 시스템에서 요구되는 모든 자원 공급이 가능하다 [1][2]. FPGA는 자동화, 컴퓨터 네트워크, 신호 처리, 자동차 및 군사 시스템 등과 같이 다양한 임베디드 시스템 분야에서 사용되고 있다 [3]. 이와 같이 FPGA가 다양한 분야에 사용됨에 따라 보안에 대한 우려도 함께 커져가고 있다. 특히 나라의 안전과 직결된 국방 및 군사 시스템이 공격을 받을 경우 큰 위험을 초래할 것이다.

FPGA를 비롯한 하드웨어에 삽입 가능한 HT (Hardware Trojan)는 임베디드 시스템 안전에 위협이 된다. 하드웨어 설계의 복잡성이 증가하고 설계와 제조 과정이 세계화됨에 따라서 신뢰되지 않은 공급자, IP (Intellectual Property), 그리고 외부의 공장으로부터 HT가 유입 될 가능성이 존재하여 이것에 대한 우려가 크게 증가하고 있다 [4]. 이러한 위협에 대응하기 위하여 HT에 나타나는 물리적인 특징, 트리거 및 페이로드의 특징을 기반으로 하여 HT를 탐지하는 방법들이 제안되었다 [5][6][7][8].

1 Information Security Lab., GSI, Yonsei University, Seoul, 03722, Korea.
* Corresponding author (taekyoung@yonsei.ac.kr)
[Received 18 November 2019, Reviewed 21 November 2019(R2 2 January 2020), Accepted 22 January 2020]
☆ 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다 (UD190016ED).
☆ 본 논문은 장재동의 석사학위논문을 바탕으로 작성되었음

http://dx.doi.org/10.7472/jksii.2020.21.2.109

하지만 기존의 탐지 방법들은 대부분 대상이 IC칩으로 한정되어있기 때문에 내부 구성요소가 다른 FPGA에는 기존의 방법을 적용하기가 어렵다. 따라서 기존 연구의 한계점을 해결하면서 FPGA에 삽입되는 HT에 대응 가능한 탐지 방법이 필요하다. 본 논문에서는 이러한 문제를 해결하기 위해 FPGA 아키텍처에 삽입될 수 있는 HT를 탐지하는 방법을 제안한다.

2. 배경 기술

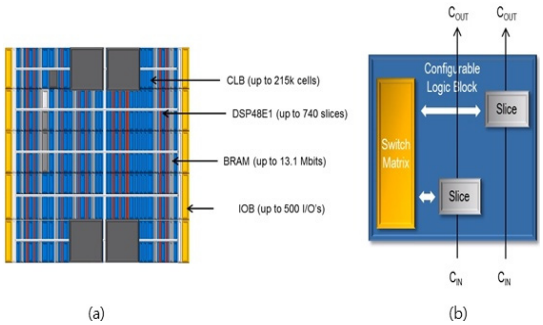
2.1 FPGA

본 장에서는 Xilinx 사의 Artix-7을 예시로 하여 전체적인 FPGA의 구조와 design flow에 대하여 설명한다.

FPGA 구조. Xilinx FPGA는 Figure 1의 (a)와 같이 IOB (Input/Output Block), DSP (Digital Signal Processing), CLB (Configurable Logic Block), BRAM (Block RAM) 등의 FPGA 구성요소들이 나열되어 있는 column 구조를 가진다. Column 구조는 clock region이 포함되어 있는 행으로 나누어지며, 행에 포함된 각 column을 tile이라고 한다. 하나의 IOB, DSP, CLB 그리고 BRAM tile에는 각각 40개의 IOB, 8개의 DSP slice, 20개의 CLB, 그리고 4개의 Block RAM이 존재한다. CLB는 회로 로직 (logic) 정보와 함께 대부분의 연결 정보를 가지며, BRAM은 작은 internal memory를 제공하고 IOB는 I/O interface 옵션을 제공한다. 각각의 기기에 포함되어 있는 자원의 수는 디바이스별로 상이하다 [9].

LUT. LUT (Lookup Table)는 회로의 로직이 FPGA에 저장되는 최소 단위이다. 구현된 로직에 해당하는 boolean equation은 진리표로 변환되고, 변환된 진리표의 출력 값이 LUT의 configuration bit로 저장된다. Artix-7의 LUT는 최대 6개의 입력을 사용 가능하며, 입력 수에 따라서 configuration bit의 길이가 달라진다. LUT에는 FPGA 상에서 구현된 회로 로직 정보가 저장되기 때문에, HT를 탐지하기 위하여 LUT는 가장 중요한 구성요소로 사용된다.

PIP. PIP (Programmable Interconnect Point)는 input pin과 output pin 간 연결 정보를 갖는 자원으로, 해당 PIP가 존재하는 tile 위치, 그리고 start wire와 end wire로 구성된다. FPGA 안의 대부분의 PIP는 CLB tile에 위치하고 있으며, 이것은 전체 로직의 연결을 담당한다. CLB tile에 존재하는 PIP는 Figure 1의 (b)에서 볼 수 있듯 switch matrix와 CLB를 연결하는 PIP는 INT (Interconnect) type



(Figure 1) Xilinx Artix-7 FPGA Structure, (a) Tile Structure, (b) CLB Structure [9]

PIP, CLB 내부에 존재하는 PIP는 CLB type PIP라고 한다. INT type PIP는 switch matrix와 CLB의 연결에 사용되며, CLB type PIP는 CLB 내부 자원들 간 연결에 사용된다. PIP는 CLB tile 외에도 BRAM, DSP, IOB 등의 tile에도 존재하며 각 자원들 간 연결을 위하여 사용된다. 또 PIP는 로직을 연결하는 것뿐만 아니라, 외부와의 데이터 입력 및 출력을 담당하는 IOB tile과 로직을 연결한다.

Design flow. FPGA 개발자는 먼저 RTL (Register-Transfer Level)을 이용하여 회로를 설계한다. 그 후 FPGA 설계 프로그램 (Xilinx Vivado 또는 Intel Quartus 등)을 이용하여 최종적으로 구성 정보를 저장하고 있는 비트스트림 (bitstream) 파일을 생성한다.

예를 들어, Xilinx 사의 Vivado에서 RTL은 먼저 합성 (Synthesis) 과정을 거쳐 Xilinx-specific netlist 파일인 NGC 파일로 변환된다. 그리고 translate (또는 NGDBuild) 과정을 거쳐 전체 회로의 구성 정보를 가지고 있는 NGD (Native Generic Database) 파일이 생성된다. 이 NGD 파일은 map 과정을 거쳐서 물리적 설계를 가지고 있는 NCD (Native Circuit Description) 파일로 변환된다. NCD 파일은 배치 및 결선 (Place and Route) 과정을 거친 후 실제 기기에 물리적으로 배치 및 결선된 NCD 파일로 변환이 이루어지고, 이것은 최종적으로 비트스트림으로 변환된다. 비트스트림은 비휘발성 메모리에 저장되어 있다가 전원이 인가될 경우 FPGA로 로드된다.

XDL. XDL (Xilinx Design Language)은 netlist의 텍스트 형식을 말하며, NCD 파일을 변환하여 생성 가능하다{명령어: xdl - ncd2xdl}. XDL 파일은 design statement, module statement, net statement, 그리고 instance statement로 이루어진다. Design statement는 각 XDL 파일마다 하나씩 존재하며, 설계 명, 그리고 해당 FPGA 기기 명과 같은 정보를 기술한다. Module statement는 모듈 (module)

의 인터페이스를 알려주는 포트 정보를 갖고 있으며, 다수의 instances, nets를 포함하고 있다. Net statement는 net들의 input pin, output pin, 그리고 pin들 간의 연결 정보인 PIP를 포함하고 있다. Instance statement는 사용된 FPGA의 자원 (DSP, SLICEM, SLICEL 등)에 대한 정보를 가지고 있다. 예를 들어, 하나의 SLICEL instance에는 해당 slice의 좌표, 해당 slice가 있는 CLB의 좌표, 4개의 LUT 중 사용 중인 LUT, 각각의 사용 중인 LUT 내부에 저장된 boolean equation 정보 등이 저장되어 있다.

2.2 HT (Hardware Trojan)

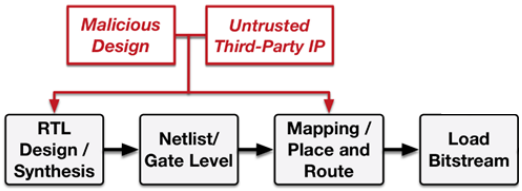
일반적으로 HT는 트리거와 페이로드로 구성된다 [10]. 트리거는 특정 조건이 만족되었을 경우 페이로드를 활성화시키며, 페이로드는 공격자가 의도한 악의적인 동작으로 회로에 삽입되어 서비스 거부, 기능 변경, 기능 저하, 민감 정보 유출 등의 악성 기능을 수행한다. 여러 모듈에 분산되어 구현되는 페이로드와는 다르게 트리거는 하나의 모듈에 구현되며, 테스트 단계에서 페이로드가 활성화되지 않도록 복잡한 조건을 사용한다 [7]. 이와 같은 트리거의 종류 및 페이로드의 특성에 따라서 HT를 분류하는 많은 연구가 진행되었다 [11][12][13][14]. TrustHub는 HT 탐지 위한 벤치마크 샘플을 제공하고 있으며, 본 논문에서는 해당 샘플을 사용하여 FPGA에 나타나는 트리거의 특징을 분석하였다.

3. 위협 모델

FPGA는 개발 단계에서 여러 하드웨어 설계자, 3PIP (Third-party Intellectual Property) 제공자, 아웃소싱 회사를 거치며 HT가 유입 될 수 있다.

이러한 위협 모델을 설명하기 위하여, 아래와 같은 세 가지 가정을 한다.

- 공격자는 FPGA의 전체 개발 프로세스, 즉 설계와 합성, 배치 단계에서 HT를 삽입 가능하다.
- RTL과 3PIP는 배치 및 결선이 완료된 netlist 형태로 변환 가능하고, 아웃소싱 회사는 하드웨어 설계를 netlist 형태로 제공한다.
- HT가 삽입된 회로와 삽입되지 않은 회로는 악의적인 기능이 트리거되지 않으면 정상적인 회로와 구분이 불가능하여 배포하기 전 테스트 단계에서 수행하는 기능의 검증으로는 HT를 탐지할 수 없다.



(Figure 2) Threat Model in FPGA Development

Figure 2는 FPGA의 전반적인 개발 과정에서 나타나는 위협 모델을 나타낸다. 그림에서 볼 수 있듯이 HT는 모든 개발 과정에서 삽입될 수 있다. 회로의 설계 단계에서 악의적인 디자이너가 회로를 설계하거나, 신뢰되지 않은 CAD tool을 활용하여 HT가 회로에 삽입될 수 있으며, 악의적인 3PIP 디자이너가 HT를 IP에 삽입하거나, 3PIP가 배포되는 과정에서 HT를 삽입할 수 있다.

본 논문의 목표는 netlist, HDL로 제공되는 FPGA 설계에서의 HT 삽입 여부를 알아내는 것이다. 이를 위해 FPGA의 HT에서 나타나는 특징을 기계학습 모델에 학습한 후, 학습된 모델을 사용하여 HT를 탐지한다.

4. 시스템 설계

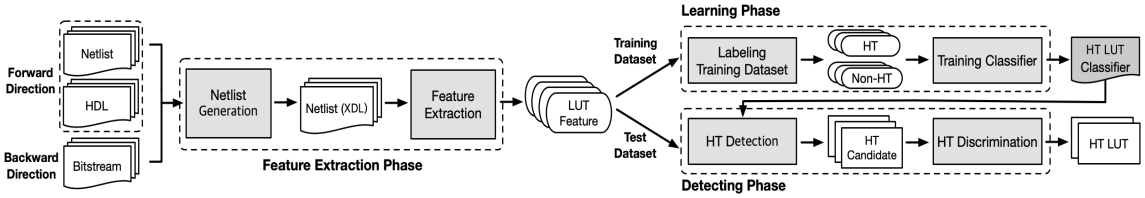
본 장에서는, HT가 FPGA의 LUT-level에서 나타내는 특징을 분석한 후 위협 모델에서 언급한 HT의 위협에 대응 가능한 시스템을 설계하는 방법에 대해 서술한다.

Figure 3은 본 논문에서 제시한 HT 탐지 시스템을 나타낸다. 본 탐지 시스템은 3가지 단계로 구성된다.

(1) 특징 추출 단계 (Feature Extraction Phase) : 주어진 회로에 대한 netlist를 생성한 후 해당 특징 정보를 추출하는 단계
(2) 학습 단계 (Learning Phase) : 학습 샘플에서 추출되어진 특징 정보를 Non-HT와 HT로 구분하여 머신러닝을 통해 HT LUT Classifier를 학습하는 단계
(3) 탐지 단계 (Detection Phase) : HT LUT Classifier를 통해 HT Candidate (HT 후보)를 탐지하고, 탐지된 HT Candidate에 대하여 거리를 기반으로 구별하여 HT LUT를 탐지하는 단계

4.1 특징 정보 추출 단계

특징 정보 추출 단계에서는 netlist, HDL 파일이 입력되어 XDL 파일로 변환되며, 변환된 XDL 파일에서 기계



(Figure 3) Hardware Trojans Detection System Design

학습을 위한 특징 정보를 추출한다. 앞서 강조한 것과 같이 설계되어진 회로의 로직은 LUT-level로 구현이 되기 때문에, 특징 정보를 LUT 단위로 추출하였다.

Figure 3에서 'Netlist Generation'은 Xilinx 사의 ISE 툴을 사용하여 netlist 또는 HDL을 XDL 파일로 변환하는 과정이다. 'Feature Extraction'은 XDL 파일을 파싱하여 각 LUT의 특징을 추출하는 과정이다. XDL의 net statement와 instance statement에서 LUT의 연결 정보 (예를 들어, inpin 과 outpin)와 로직 정보 (예를 들어, boolean equation)를 파싱하여 특징으로 추출한다.

Table 1은 HT LUT 분류에 사용되는 17개의 feature를 5개 category로 나누어 보여준다. Table 1에서 Equation 및 truth-table은 instance statement의 논리식을 이용하여 특징으로 사용한 것이고, LUT Input과 LUT Output, 그리고 Flip-Flop은 net instance의 inpin과 outpin을 이용하여 특징으로 사용한 것이다. 각각의 특징 정보에 대한 자세한 설명은 아래와 같다.

논리식 (Boolean equation). HT LUT의 논리적 특징과 관련해 LUT의 논리식과 관련된 4개의 특징을 선정하였다. LUT의 논리식은 해당 LUT가 수행하게 되는 로직을 식으로 나타낸 것으로써 수행하는 로직에 따라서 다양한 식을 나타낸다. 앞에서 확인한 바와 같이, HT LUT는 RC (Rare Condition) 특성을 가지고 있으며 이러한 LUT들은 서로 비슷한 패턴을 가지고 있었다.

예를 들어, aes_t400, basicrsa_t100 샘플에서와 같이 모든 연산이 AND로 연결된 비슷한 패턴이 나타난다 (FPGA에서 사용하는 식은 * 은 and, + 는 or, @는 xor을 나타낸다). 논리식은 매우 긴 식이 존재하기 때문에 식을 SHA-1으로 해시화한 값 (A.1)을 특징으로 사용하였다. 또한 논리식의 문자열 길이 (A.2), 연산자의 개수 (A.3), 피연산자의 개수 (A.4)를 특징 정보로 사용하였다.

진리표 (Truth table). HT LUT의 논리적 특징과 관련해 LUT의 진리표와 관련된 4가지의 특징을 선정하였다. LUT의 진리표는 모든 입력에 대한 출력을 나타내는 것으로써 논리식의 변환을 통해 생성 가능하며, 사용된 개수 (n)에 따라 2^n개의 output 크기를 갖는 진리표가 생성된다. 이전에 확인한 것과 같이, HT LUT는 특정한 한 가지 경우에만 다른 출력을 내보내는 RC 특성을 갖는다. 따라서 LUT의 Truth table의 출력을 특징 정보로 사용하였다. 결과적으로 LUT의 진리표 출력에서 1과 0의 개수 (B.2),(B.3), RC LUT를 구분하는 특징 (B.4), 진리표의 전체 출력 값 (B.1)을 특징으로 사용하였다.

입력 (Input). 트리거 LUT는 서로 연결이 되어 있다는 관찰을 기반으로 하여 LUT의 입력과 관련된 4가지 특징을 선정하였다. LUT는 최대 6개 입력 핀을 사용하여 6비트의 입력을 받을 수 있다. 또한 복잡한 활성화 조건 (예를 들어, 128 비트의 활성화 조건)으로 설계되는 트리거는 여러 LUT가 결합되어서 만들어진다. 앞서 확인한 바와 같이 활성화를 위하여 외부로부터 입력을 받거나, 다른 HT LUT와 연결되어 있다. 따라서 입력과 연결되는 자원을 RC LUT, Non-RC LUT, FF, IOB, 기타 자원으로 구분하여 각각의 입력 핀이 어떤 자원과 연결되어 있는지를 특징으로 사용하였다. 결과적으로 LUT에 사용된 inpin의 개수 (C.1)와, 각각의 inpin에 연결된 자원 종류 (C.2), 연결된 자원 개수 (C.3), RC LUT, IOB와 연결된 pin 비율 (C.4)을 특징으로 사용하였다.

출력 (Output). 입력과 마찬가지로 트리거 LUT가 서로 연결되어 있다는 관찰에 기반으로 하여, LUT의 출력과 관련된 3가지 특징을 선정하였다. LUT의 출력은 LUT와 직접 연결된 출력 핀을 통하여 다른 자원으로 전달이 되거나, FF, MUX를 거쳐 다른 자원과 연결될 수 있다. 따라서 이러한 출력이 어떤 자원과 연결되어 있는지를 특징으로 추출하였다. 앞서 확인한 HT LUT 물리적 특성과 같이 HT LUT의 출력은 다음 HT LUT로 연결이 되어 있다. 따라서 outpin이 영향을 미치는 자원을 IOB, single LUT, Non-single LUT 등으로 구분하여 출력이 어떤 자원

(Table 1) Feature Information Extracted From Each LUT

Category	Feature	Type	Description
A	Boolean Equation		
A.1	Equation_hash	Numeric	논리식 해시 값
A.2	Equation_length	Numeric	논리식 길이
A.3	Equation_op_count	Numeric	논리식 연산자 개수
A.4	Equation_pin_count	Numeric	논리식 피연산자 개수
B	Truth-table		
B.1	Lut_output	Numeric	전체 출력 비트
B.2	Lut_output_1	Numeric	출력에서 1의 개수
B.3	Lut_output_0	Property	출력에서 0의 개수
B.4	Rare_condition	Numeric	RC 여부
C	Input		
C.1	Used_pin_count	Numeric	사용된 핀의 개수
C.2	Inpin_conn_resource	Property	입력받은 자원 종류
C.3	Inpin_conn_resource_count	Numeric	입력받은 자원별 개수
C.4	Inpin_conn_resource_ratio	Numeric	입력받은 자원의 RC 비율
D	Output		
D.1	Out_influence	Property	출력이 영향을 미치는 자원 종류
D.2	Out_influence_count	Numeric	출력이 영향을 미치는 자원 개수
D.3	Out_conn_rc_count	Numeric	출력이 영향을 미치는 RC 자원 개수
E	Flip-flop		
E.1	Conn_FF	Property	연결된 Flip-flop 위치
E.2	Conn_FF_influence_count	Numeric	연결된 Flip-flop의 출력이 영향을 주는 자원 개수

과 연결되어 있는지를 특징으로 사용하였다. 결과적으로 outpin이 영향을 미치는 자원 종류 (D.1)와 outpin이 영향을 미치는 자원 개수 (D.2), 출력과 연결된 RC LUT 개수 (D.3)를 특징으로 사용하였다.

Flip-Flop. LUT의 출력은 Flip-Flop에 저장되어 있다가 다음 입력으로 사용될 수 있다. 만약 LUT와 연결된 Flip-Flop의 출력을 고려하지 않는다면, 출력과 연결된 자원을 완벽하게 찾아낼 수 없다. 따라서 해당 LUT와 연결된 Flip-Flop의 정보를 함께 특징으로 사용한다. 위치 정보를 나타내는 Conn_FF는 LUT와 연결된 Flip-Flop이 LUT와 같은 slice에 있으면 1, 다른 slice에 있으면 2로 표현하고, LUT와 연결된 Flip-Flop이 없을 경우 0으로 표현한다 (E.1). 그리고 LUT와 연결된 Flip-Flop이 있을 경우, 출력과 연결된 자원 수 (E.2)를 특징 정보로 사용하였다.

4.2 학습 단계

학습 단계에서는 주어진 LUT가 HT LUT인지 아닌지를 분류하기 위하여 Classifier를 생성하는 단계이다. 이를 위하여 학습 데이터 셋을 HT 또는 Non-HT로 구분한 뒤에 나눠진 특징을 모델에 학습하여 Classifier를 생성한다.

학습 데이터 분류. 지도 학습으로 Classifier를 학습하기 위해 추출된 LUT feature를 HT 또는 Non-HT로 아래 과정을 통하여 분류하였다.

(1) Netlist 파일에서 소스코드의 작성 시에 사용한 트리거 변수 이름이 포함된 LUT 위치 식별
(2) Netlist 파일에서 트리거 이름이 포함되어 있지 않은 LUT 위치를 분석하여 위치 식별
(3) 발견한 트리거 LUT에서 구별되는 트리거 특징을 나타내는 HT LUT만 남기는 가지치기 (pin 및 RC 개수)

(Table 2) Experimental Performance of Each Model

실험 (Train/Test)	전체 LUT	HT LUT	Extra Trees TPR	Extra Trees FPR	Random forest TPR	Random forest FPR	SVM TPR	SVM FPR
실험 1 (Benchmarks/Benchmarks)	107,356	601	93.67%	0.19%	93.34%	0.12%	75.38%	0.07%
실험 2 (Benchmarks/HT-Free)	57,146	-	-	0.52%	-	0.40%	-	0.09%
Average			93.67%	0.35%	93.34%	0.26%	75.38%	0.08%

(4) HT LUT와 Non-HT LUT를 두 개의 클래스 (class)로 학습

Training Classifier. 분류 과정을 통해 나눠진 특징 정보를 머신러닝 학습 모델 중 Extra Trees 모델을 사용하여 학습한다. 앙상블 기법을 적용한 Extra Trees 모델은 단일 트리 모델에 비하여 훈련 데이터에 대한 과대 적합을 막아주고, 일반화 성능이 높다는 장점이 존재한다. 또한 앙상블 기법을 사용한 Random forest 모델보다 무작위성을 증가시킨 모델로 Random forest에 비해 더 넓은 시각으로 특징 정보를 평가할 수 있다. 모델 학습 시 HT 특징 정보와 Non HT 클래스를 balanced weight 옵션을 주어 학습을 진행하였다. 그 이유는 전체 LUT 중에서 HT LUT의 비율이 매우 낮아서 만약 이대로 모델을 학습한다면, HT LUT 식별보다는 Non-HT LUT 식별에 특화된 모델이 생성되기 때문이다. Balanced weight 옵션을 적용한 각 클래스의 가중치는 아래 식과 같이 계산된다.

$$w_j = \frac{n}{kn_j} \quad (1)$$

여기서 w_j는 클래스 j 의 가중치를 나타낸다. n은 전체 특징 정보의 개수, n_j는 클래스 j의 특징 정보 개수를 나타내며, k는 클래스의 총 개수이다.

이러한 학습 과정을 통해 생성된 Classifier를 통해 탐지 과정에서 주어진 netlist의 모든 LUT를 Non-HT LUT와 HT Candidate로 분류하게 된다.

4.3 탐지 단계

탐지 단계에서는 생성된 HT LUT Classifier를 사용해 주어진 평가 데이터 셋의 모든 LUT에서 HT Candidate를 분류하게 된다. 이후 탐지된 HT Candidate를 거리 기반으로 검사하는 HT discrimination 과정을 통해 HT LUT를 최종적으로 탐지하게 된다.

HT Detection 과정에서는 학습하여 생성된 HT LUT Classifier를 사용해 테스트 데이터 셋으로부터 추출된 LUT들 중 HT LUT로 의심되는 LUT를 분류한다. 이 단계에서는 HT의 트리거로 사용된 LUT를 발견하지 못하는 false negative error를 줄인다. 이 과정에서 HT가 삽입되지 않은 정상 회로에서 HT LUT로 잘못 탐지하는 false positive error가 발생할 수 있다. 이 문제를 해결하기 위하여 HT Discrimination 과정에서는 HT LUT의 집약적인 특징을 이용하여 false positive error의 발생을 최소화시킨다. 본 연구에서 관찰한 HT LUT의 분포에서 1개의 HT 회로는 둘 이상의 HT LUT를 가지며, 해당 LUT는 FPGA 상에서 물리적으로 일정한 거리 이내에서 구현되어 있다는 가정을 도출하였다. 이 가정을 기반으로 하여 학습된 Classifier의 입력으로 주어진 LUT가 HT Candidate로 분류되었을 때, 이 LUT P와의 거리 D(P, Q)가 임계치 T 이내인 LUT Q가 하나 이상 존재하는지를 검사한다. 만약 하나 이상의 LUT Q가 존재할 경우, LUT P를 HT LUT로 판단한다. 이때 사용하는 distance dist(P, Q)는 아래의 식과 같다.

$$dist(P,Q) = |P_x - Q_x| + |P_y - Q_y| \quad (2)$$

여기서 x와 y는 각각 LUT의 인스턴스의 슬라이스 x축과 y축 좌표를 나타낸다.

5. 실험 방법 및 결과

5.1 실험 환경 및 데이터 수집

실험 환경. 본 연구는 Xilinx 사의 Artix-7 XC7A200T FFG1156 기기를 대상으로 하였으며, Xilinx 사의 ISE 툴을 사용해 각 샘플을 구현하여 XDL netlist를 생성하였다. 특징 추출 방식은 XDL을 파싱해 추출하는 방법으로 python을 이용해 구현하였다. 기계학습을 이용한 탐지 부분은 python의 scikit-learn을 사용하여 모델을 학습하였고,

(Table 3) Experimental Results by Benchmark Sample (Extra Trees)

Test Sampels	Total LUT	HT LUT	TP	FN	FP	TN	TPR	FPR	Detect
basicrsa_t100	795	6	2	787	2	4	33.33%	0.25%	✓
basicrsa_t200	829	6	6	823	0	0	100.00%	0.00%	✓
basicrsa_t300	787	6	6	777	4	0	100.00%	0.51%	✓
aes_t400	3,458	21	21	3,437	0	0	100.00%	0.00%	✓
aes_t700	3,190	21	21	3,169	0	0	100.00%	0.00%	✓
aes_t800	3,214	38	30	3,176	0	8	78.95%	0.00%	✓
aes_t900	3,319	25	24	3,294	0	1	96.00%	0.00%	✓
aes_t1000	3,190	21	21	3,169	0	0	100.00%	0.00%	✓
aes_t1100	3,214	38	28	3,176	0	10	73.68%	0.00%	✓
aes_t1200	3,319	25	24	3,294	0	1	96.00%	0.00%	✓
aes_t1600	3,485	38	26	3,447	0	12	68.42%	0.00%	✓
aes_t1700	3,585	20	20	3,565	0	0	100.00%	0.00%	✓
b19_t300	25,090	144	144	24,841	105	0	100.00%	0.42%	✓
b19_t400	25,179	142	142	24,970	67	0	100.00%	0.27%	✓
b19_t500	24,702	50	48	24,621	31	2	96.00%	0.13%	✓
Total	107,356	601	563	106,546	209	38	93.67%	0.19%	15/15

이후 탐지 단계 또한 python을 이용하여 구현하였다.

Benchmark. 탐지 실험에 사용된 벤치마크 샘플은 TrustHub에서 15개의 샘플을 다운로드하여 사용하였다. 해당 샘플들은 3개의 서로 다른 그룹으로 구성되며, 원본 소스코드에 다양한 트리거 구조와 페이로드가 삽입되었다. 기기에서 가용 가능한 LUT 자원의 50%를 사용한 AES 벤치마크 샘플은 AES 암호화 알고리즘의 기능을 하는 Verilog 소스코드로, 암호화 기능을 방해하는 서로 다른 9개의 HT 샘플이 포함된다. 실험에서 약 90%로 가장 많은 LUT를 사용한 B19 벤치마크 샘플은 인텔 (Intel) 80386 프로세서에 들어가는 VHDL 소스코드로 프로세서의 주소 버스 (Address bus)나 데이터 버스 (Data bus) 등을 조작하는 3가지의 서로 다른 HT 샘플이 포함된다. 실험 샘플 중 3%로 가장 적은 LUT를 사용한 Basicrsa 벤치마크 샘플은 RSA 암호화 알고리즘을 수행하는 VHDL 샘플로 암호화 기능을 방해하는 3개의 서로 다른 HT 샘플이 포함된다.

HT-Free Samples. 실험에서 사용한 HT가 삽입되지 않은 샘플 (HT-Free)은 OpenCores에서 정상적인 샘플 48개를 다운로드하였다. 수집된 HT-Free 샘플은 통신 컨트롤러, SoC (System on chip), 프로세서, 산술 연산 코어 등 다양한 기능을 하는 오픈소스들로 구성된다. HT-Free 샘플 48개는 본 연구의 탐지 기법이 HT가 삽입되지 않은 샘플에서도 낮은 오탐율을 갖는지 확인하기 위하여 실험에 사용한다.

5.2 실험 방법

본 장에서는 HT 탐지 시스템이 HT를 탐지하는데 얼마나 효과적인지 평가한다. Table 2는 각 모델 별 실험 성능을 보여준다. 실험에서 사용한 데이터는 TrustHub에서 수집한 벤치마크 샘플에 대한 실험 1, 그리고 HT가 삽입되지 않은 샘플에 대한 실험 2이다. 실험 1은 Leave-One-Out 방식의 교차검증을 통해 실험 결과를 도출하였다. 이때 HT LUT로 탐지한 개수를 true, 그렇지 않은 것을 false로 하여 측정하였다.

여기서 Leave-One-Out 교차검증 방법이란 모델 성능을 평가할 때 쓰이는 검증 방식으로, 실험의 샘플이 적을 때 효과적이다. 이 방식은 검증 데이터 셋 중 하나를 테스트 데이터 셋으로 지정하고 나머지를 학습에 사용한다. 예를 들어, 검증에 사용하는 15개 데이터 셋에서 14개를 학습하고 나머지 1개를 테스트에 사용하는 방식으로, 모든 데이터 셋을 반복적으로 테스트하여 모델 성능을 검증한다.

HT가 삽입되지 않은 실험 2는 TrustHub에서 수집한 벤치마크 샘플을 학습하여 모델을 생성한 후, 생성된 모델을 HT가 삽입되지 않은 샘플 48개에 대해 테스트하여 성능을 평가하였다.

특징 정보 추출. 먼저 모든 샘플에서 특징 정보를 추출하였다. TrustHub 벤치마크 샘플로부터 107,356개, HT가 삽입되지 않은 샘플로 LUT 57,146개를 추출하였고,

각 LUT에 대하여 17개의 특징을 추출하였다. 추출한 특징 정보로부터 HT의 특징을 기계학습 모델에 학습시키기 위해 학습 데이터 셋으로 사용된 TrustHub 벤치마크 샘플에서 HT LUT 604개와 Non HT LUT 106,752개로 분류하였다.

모델 학습. HT를 탐지하는데 효과적인 학습 모델을 선택하기 위해 Random forest, Extra Trees, SVM (Support Vector Machine) 알고리즘을 사용하여 비교하는 실험을 진행하였다. 앞서 설명한 것과 같이 각 모델의 학습 시 HT 클래스와 Non HT 클래스를 balanced 옵션으로 가중치를 주어 학습하였다.

5.3 실험 결과

실험은 거리 임계치를 1로 적용하여 진행하였다. 앞서 언급하였듯, Table 2는 각 모델 별 실험 성능을 보여준다. 여기서 TPR (True Positive Rate)은 전체 HT LUT (Total True) 중 탐지한 HT LUT (TP)의 비율을 나타내고, FPR (False Positive Rate)은 전체 Non HT LUT 중 HT라고 잘못 탐지한 LUT (FP)의 비율을 나타낸다.

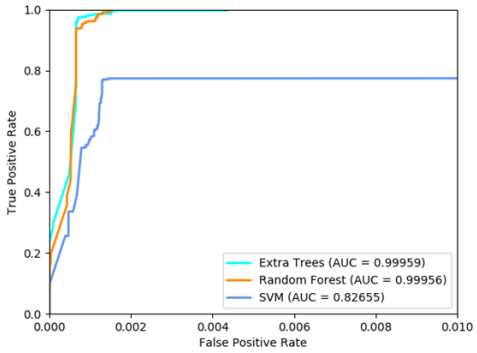
실험 1의 결과 Extra Trees와 Random forest는 각각 93.67%와 93.34%의 TPR로 HT LUT를 탐지하였고, 약 0.19%와 0.12%의 낮은 FPR을 보여주었다. SVM의 경우, FPR이 0.07%로 다른 모델에 비해 더 좋은 성능을 보였으나 HT LUT 178개를 탐지하지 못하여 TPR은 75.38%에 불과했다. HT-free 샘플에 대한 실험 2에서는 Extra Trees, Random forest에서 각각 0.52%와 0.40%의 FPR을 나타내었고, SVM은 0.09%로 다른 모델에 비하여 좀 더 좋은 성능을 보였다.

Extra Trees 모델은 Decision Tree 모델에 비하여 전체 데이터 셋에서 모든 HT LUT를 탐지하며 HT 탐지 시스템에 가장 이상적인 결과를 나타내었다.

SVM 모델의 경우, 다른 모델에 비하여 FPR 성능은 좋았으나, HT LUT를 탐지하는 것보다는 Non-HT LUT를 탐지하는데 특화된 성능을 보였다. 하지만 낮은 TPR 결과가 나타났기 때문에 HT LUT 탐지가 목적인 본 시스템에서는 적합하지 않다. 따라서 HT 탐지에 있어서 가장 좋은 성능을 가진 Extra Trees를 실험에 사용하였다.

Table 3은 각 데이터 셋에 대한 실험 결과를 나타낸 표이다. 실험 결과에서 볼 수 있듯이 본 탐지 시스템을 통해 모든 벤치마크 샘플에서 적어도 2개 이상의 HT LUT를 탐지하여 HT 샘플 15개를 모두 HT 샘플로 분류해내었고, 약 0.35%의 낮은 FPR로 유의미한 결과를 확인 할 수 있다.

Figure 4는 실험 1에 대한 각 모델의 ROC (Receiver Operating Characteristics) 곡선을 나타낸다. ROC 곡선이란 TPR과 FPR의 곡선을 나타낸 것으로, 왼쪽 위에 가까울수록 이상적인 결과로 볼 수 있으며, ROC 곡선이 차지하는 면적을 AUC (Area Under the Curve)라고 한다. 해당 그림에서 볼 수 있듯, Extra Trees 모델이 0.99959의 AUC로 다른 모델에 비해 높은 AUC를 나타내는 것을 확인할 수 있다. 우리는 각 데이터 셋에서 나타나는 모델의 성능과 가중치를 적용한 실험 결과를 토대로 하여 가중치를 준 Extra Trees를 HT 탐지를 위해 적합한 모델로 선택하였다.



(Figure 4) ROC Curve of Each Model

6. 관련 연구

Hicks 등은 HDL 소스코드 분석을 통해 생성한 그래프를 이용하여 기능 검증 테스트 중 사용되지 않는 노드를 식별하는 UCI (Unused Circuit Identification) 알고리즘을 제시하였다 [5].

Zhang 등은 기능 검증 테스트 중 활성화되지 않는 회로를 식별하여, 식별된 회로에서 HT 트리거의 입력신호를 탐지하는 방법을 제시하였다 [6].

Waksman 등은 gate-level netlist에서 진리표를 구성하여 Boolean function 분석을 통하여 출력에 거의 영향을 미치지 않는 입력 핀을 식별해 HT 트리거 wire를 탐지하는 FANCI (Functional Analysis for Nearly Unused Circuit Identification)알고리즘을 제시하였다 [7].

Fyrbiak 등은 gate-level netlist를 분석 및 역공학 하는 HAL툴을 개발하여, AES 암호화 알고리즘을 수행하는

회로에 AES key를 유출하는 HT를 삽입하는 방법을 보였다 [8].

Sturton 등은 기능 검증 중에도 활성화 되는 HT를 자동으로 구축하는 방법을 제안하며, UCI가 unused circuit 이라고 정의한 HT 탐지 방법이 매우 제한된 HT만 탐지할 수 있다는 한계점을 보여주었다 [15].

Zhang 등은 FANCI의 탐지 방법을 회피하는 HT를 설계하는 방법을 제시하였다 [16].

7. 결 론

Hardware Trojans는 학계, 산업, 정부 기관 등에서 지속적으로 주목받아 왔다. HT 탐지를 위하여 많은 노력이 이루어지고 있으나 FPGA에 유입 가능한 HT의 위협을 해결하기 위한 방안은 근본적으로 아직 부족하다고 할 수 있다. 본 연구에서는 이러한 문제를 해결하기 위하여 FPGA의 회로 설계 과정에서 삽입 가능한 HT를 탐지하는 방법을 제시하였다. 이를 위하여 HT가 FPGA의 구성요소에서 나타나는 특징을 분석한 후 기계학습을 통해 그 특징을 학습하여 HT를 탐지하였다.

본 연구는 FPGA의 Artix 7세대를 대상으로 실험을 진행하였다. FPGA는 개발이 완료된 이후에도 재 프로그래밍 기능을 사용하여 HT가 유입될 가능성이 존재한다. 따라서 개발이 완료된 이후에도 FPGA 칩의 내부 로직을 역공학하여 HT를 탐지하는 방법에 대한 연구가 향후 이루어져야 할 것이다.

참고문헌(Reference)

[1] Kuon, Ian, Russell Tessier, and Jonathan Rose, "FPGA Architecture: Survey and Challenges," Foundations and Trends in Electronic Design Automation, vol. 2, no. 2, pp. 135-253, 2008. http://dx.doi.org/10.1561/1000000005

[2] Sass, Ronald, and Andrew G. Schmidt, "Embedded Systems Design with Platform FPGAs," Principles and Practices, 2010. https://dl.acm.org/doi/pdf/10.5555/1895071

[3] Rostami, Masoud, Farinaz Koushanfar, and Ramesh Karri, "A primer on Hardware Security: Models, Methods, and Metrics," Proceedings of the IEEE, vol. 102, iss. 8, pp. 1283-1295, 2014. http://dx.doi.org/10.1109/JPROC.2014.2335155

[4] Alkabani, Yousra, and Farinaz Koushanfar, "Designer's Hardware Trojan Horse," In IEEE International Workshop on Hardware-Oriented Security and Trust (HOST), pp. 82-83, 2008. http://dx.doi.org/10.1109/HST.2008.4559059

[5] Hicks, Matthew, Murph Finnicum, Samuel T. King, Milo MK Martin, and Jonathan M. Smith, "Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically," IEEE Symposium on Security and Privacy (S&P), pp. 159-172, 2010. http://dx.doi.org/10.1109/SP.2010.18

[6] Zhang, Jie, Feng Yuan, Linxiao Wei, Yannan Liu, and Qiang Xu, "VeriTrust: Verification for hardware trust," Design Automation Conference (DAC), pp. 1-8, 2013. http://dx.doi.org/10.1109/TCAD.2015.2422836

[7] Waksman, Adam, Matthew Suozzo, and Simha Sethumadhavan, "FANCI: Identification of Stealthy Malicious Logic using Boolean Functional Analysis," In Proc. the Conference on Computer and Communications Security (CCS), pp. 697-708, 2013. http://dx.doi.org/10.1145/2508859.2516654

[8] Fyrbiak, Marc, Sebastian Wallat, Pawel Swierczynski, Max Hoffmann, Sebastian Hoppach, Matthias Wilhelm, Tobias Weidlich, Russell Tessier, and Christof Paar, "HAL－The missing piece of the puzzle for hardware reverse engineering, Trojan detection and insertion," IEEE Transactions on Dependable and Secure Computing, vol. 16, iss. 3, pp. 498-510, 2018. http://dx.doi.org/10.1109/TDSC.2018.2812183

[9] Xilinx, https://xilinx.com/

[10] Alkabani, Yousra, and Farinaz Koushanfar, "Designer's Hardware Trojan Horse," In IEEE International Workshop on Hardware-Oriented Security and Trust (HOST), pp. 82-83, 2008. http://dx.doi.org/10.1109/HST.2008.4559059

[11] Wang, Xiaoxiao, Mohammad Tehranipoor, and Jim Plusquellic, "Detecting Malicious Inclusions in Secure Hardware: Challenges and Solutions," In IEEE International Workshop on Hardware-Oriented

Security and Trust (HOST), pp. 15-19, 2008. http://dx.doi.org/10.1109/HST.2008.4559039

[12] Karri, Ramesh, Jeyavijayan Rajendran, Kurt Rosenfeld, and Mohammad Tehranipoor, “Trustworthy Hardware: Identifying and Classifying Hardware Trojans,” Computer, vol. 43, no. 10, pp. 39-46, 2010. http://dx.doi.org/10.1109/MC.2010.299

[13] Tehranipoor, Mohammad, and Farinaz Koushanfar, “A Survey of Hardware Trojan Taxonomy and Detection,” IEEE Design & Test of Computers, vol. 27, no. 1, pp 10-25, 2010. http://dx.doi.org/10.1109/MDT.2010.7

[14] Shakya, Bicky, Tony He, Hassan Salmani, Domenic Forte, Swarup Bhunia, and Mark Tehranipoor, “Benchmarking of hardware Trojans and maliciously affected circuits,” Journal of Hardware and Systems Security, vol. 1, no. 1, pp. 85-12, 2017. http://dx.doi.org/10.1007/s41635-017-0001-6

[15] Sturton, Cynthia, Matthew Hicks, David Wagner, and Samuel T. King, “Defeating UCI: Building Stealthy and Malicious Hardware,” IEEE Symposium on Security and Privacy (S&P), pp. 64-77. 2011. http://dx.doi.org/10.1109/SP.2011.32

[16] Zhang, Jie, Feng Yuan, and Qiang Xu. “Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans,” In Proc. the Conference on Computer and Communications Security (CCS), pp. 153-166. 2014. http://dx.doi.org/10.1145/2660267.2660289

◐ 저 자 소 개 ◑

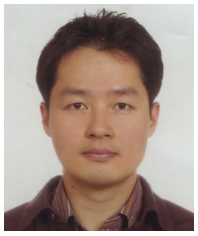
장 재 동(Jae-dong Jang)
2017년 성결대학교 컴퓨터공학과(학사)
2019년 연세대학교 정보대학원 정보보호(석사)
관심분야 : 정보 보안, IoT 보안, 웹 보안, etc.
E-mail : woehd91@yonsei.ac.kr

조 민 기(Min-gi Cho)
2017년 부산대학교 정보컴퓨터공학과(학사)
2017년~현재 연세대학교 정보대학원 정보보호(석박통합과정)
관심분야 : 소프트웨어 보안, 시스템 보안, etc.
E-mail : imgc@yonsei.ac.kr

◐ 저 자 소 개 ◑

서 예 지(Ye-zee Seo)
2017년 덕성여자대학교 디지털미디어학과(학사)
2019년 연세대학교 정보대학원 정보보호(석사)
관심분야 : 정보 보안, IoT 보안, 소프트웨어 보안, etc.
E-mail : seoyz0716@yonsei.ac.kr

정 세 연(Se-yeon Jeong)
2018년 대구가톨릭대학교 정보보호학과(학사)
2019년~현재 연세대학교 정보대학원 정보보호(석사과정)
관심분야 : 소프트웨어 보안, 시스템 보안, IoT 보안, etc.
E-mail : yeonny@yonsei.ac.kr



권 태 경(Taekyoung Kwon)
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, etc.
E-mail : taekyoung@yonsei.ac.kr