Detecting RTL Trojans Using Artificial Immune Systems and High Level Behavior Classification

Farhath Zareen
University of South Florida, fzareen@mail.usf.edu

Detecting RTL Trojans Using Artificial Immune Systems and High Level Behavior Classification

by

Farhath Zareen

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Date of Approval:
February 18, 2019

Keywords: Hardware Trojans, Hardware Security, Negative Selection Algorithm, Clonal Selection Algorithm, Control/Data-Flow Graphs

DEDICATION

Dedicated to my mom, dad and sister. I wouldn't be where I am without your unconditional love and guidance.

ACKNOWLEDGMENTS

I would like to give special thanks to my advisor, Dr. Robert Karam for his wisdom, help and support throughout this process. Thank you for having faith in me, I could not have asked for a better mentor. I would also like to thank my family and friends for always having my back, especially David, thank you for your constant support and patience. I would also like to thank the staff of the CSE main office for their help and always treating me with warmth and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Security assurance in a computer system can be viewed as distinguishing between *self* and *non-self*. Artificial Immune Systems (AIS) are a class of machine learning (ML) techniques inspired by the behavior of innate biological immune systems, which have evolved to accurately classify self-behavior from non-self-behavior. This work aims to leverage AIS-based ML techniques for identifying certain behavioral traits in high level hardware descriptions, including unsafe or undesirable behaviors, whether such behavior exists due to human error during development or due to intentional, malicious circuit modifications, known as hardware Trojans, without the need for a golden reference model. We explore the use of Negative Selection and Clonal Selection Algorithms, which have historically been applied to malware detection on software binaries, to detect potentially unsafe or malicious behavior in hardware. We present a software tool which analyzes Trojan-inserted benchmarks, extracts their control and data-flow graphs (CDFGs), and uses this to train an AIS behavior model, against which new hardware descriptions may be tested.

CHAPTER 1

INTRODUCTION

One of the major challenges in hardware security is the detection of hardware Trojans. Hardware Trojans are malicious modifications to an Integrated Circuit (IC) that change its original, desired functionality and can perform a wide range of undesirable or unsafe actions, such as leaking information to an attacker or causing system failure under specific conditions determined by the Trojan designer. Trojans may be inserted at any phase of the IC design cycle, including in high level circuit descriptions, or during fabrication in a foundry. Broadly, Trojans may be viewed as intentional faults, where the faulty behavior is difficult to model, since it is the product of a human designer rather than a naturally arising error caused by physical processes in fabrication. For Trojans in logic or memory circuits, the *payload*, or faulty behavior, may be activated by certain input combinations or sequences of input combinations, triggering the undesirable or faulty behavior [2].

These rare triggering events are designed to be stealthy and undetectable during simulation and testing. Detecting these malicious events usually requires a golden reference model [3] that is guaranteed to be Trojan-free. However, this may not be available, especially as Intellectual Property (IP) cores in recent times are mostly licensed from typically untrusted third-party vendors. Another disadvantage is that using a golden reference as the basis for detection may be inconclusive or too complex for exhaustive verification, especially for large designs [3]. Many techniques also require direct measurement of parameters such as power consumption or electromagnetic (EM) emissions, and can only be faithfully applied to fabricated ICs, but not at higher levels of abstraction.

More recently, researchers have proposed detection methodologies that do not require a golden reference model. Narasimhan et al. used a combination of functional testing and side-channel analysis to measure the transient current signature for multiple time frames under invariable state transitions [4]. This technique is effective when the knowledge of the activation time of the Trojan is known. The work by Chakraborty et al. is based on statistical vector generation wherein rare circuit nodes are triggered numerous times by the test vectors, uncovering Trojans hidden in these rare nodes [5]. This can provide valuable insight as to where in the circuit a Trojan may be hidden, and it can be a beneficial tool for finding Trojans at lower levels of abstraction. For designs still at high levels of abstraction, however, alternative techniques are required to determine whether or not unsafe behavior exists. Zhang and Tehranipoor proposed combining multiple stages of verification, including code-coverage analysis, assertion checking, equivalence analysis, removal of redundant circuits, and sequential Automatic Test Pattern Generation (ATPG) to recognize suspicious signals [6]. Fern et al. utilize Mutation Testing to detect Trojans inserted in unspecified functionality, i.e. "don't care" clauses, which may escape detection through traditional verification techniques relying on a formal specification [7].

Instead of looking for Trojans only in rare nodes, or in unspecified functionality, the goal of this work is to develop a system which is capable of recognizing certain *behavioral traits* in a circuit and classifying these traits either as benign or as containing potentially unsafe operations. A design that contains potentially unsafe operations may be compromised, either intentionally with malicious modifications or Trojan insertions or unintentionally through human error. Regardless, the ability to accurately classify such behavior for any given design would enable IP designers to quickly verify the security of their own work, or system integrators to check third party IP (3PiP) cores used in larger designs. While other previously discussed techniques would still be applicable at lower levels of abstraction, the ability to quickly analyze a design for unsafe behavior in RTL could save significant time and cost if a Trojan were discovered earlier in the design flow.

This work leverages Artificial Immune Systems (AIS) [8, 9], extending the concept of distinguishing between self and non-self behavior for the purpose of circuit behavior classification. AIS has been widely and successfully used in malware detection in software [10, 11, 12, 13], where sequences of instructions may be considered "unsafe" and may be indicative of malware, as well as pattern recognition and optimization [14, 15]. Both the Negative Selection Algorithm (NSA) [8] and Clonal Selection Algorithm (CSA) [16] are used to identify behavioral patterns in control and data-flow graphs (CDFG) extracted from behavioral Verilog HDL circuit descriptions. The models are trained on RTL Trojan benchmarks obtained from TrustHub [17, 18]. The CDFGs are extracted using PyVerilog [19], an open-source Verilog code parser and static analyzer tool, though any CDFG extraction tool may be used.

Through these experiments, it is demonstrated that AIS can successfully discriminate between designs containing safe and unsafe behavior, by recognizing patterns in Trojan-inserted designs and matching against the design under test. The main contributions of my thesis are as follows:

1. It frames the problem of detecting unsafe behavior in RTL in terms of detecting self- vs non-self-behavior in the design's control and data flow.

2. It presents a complete software tool flow for AIS-based RTL source code analysis, including model generation and behavior classification.

3. It analyzes the efficacy of Negative Selection using partial or whole string matching and Clonal Selection algorithms on binary-encoded CDFGs for detecting unsafe behavior in RTL.

4. It demonstrates how machine learning can be used to detect unsafe behavior in Trojan-included hardware, even if the specific instance of the Trojan has not been previously encountered – similar to an immune system that responds to a foreign cell in the body, even if it has not encountered that specific virus or bacteria before.

To the best of our knowledge, this is the first application of AIS to hardware Trojan detection at a high level of abstraction.

The rest of the thesis is organized as follows: Chapter 2 discusses the background on recent hardware Trojan detection techniques that use machine learning concepts (Section 2.2) and introduce a formal definition of a hardware Trojan, an Artificial Immune System (AIS) and its algorithms (Section 2.3). Chapters 3 and 4 discuss the proposed Trojan detection methodology, experimental results and analysis. Finally, chapter 5 discusses the conclusion and future work.

This thesis extends the work that has previously appeared in IEEE 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)[1] [1].

[1]© 2018 IEEE. Reprinted with permission from Farhath Zareen, Robert Karam, Detecting RTL Trojans using Artificial Immune Systems and High Level Behavior Classification, 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST).

CHAPTER 2

BACKGROUND

In this section, we first define the problem of hardware Trojans and conventional detection techniques. Second, we summarize existing ML-based Trojan detection techniques, though these are primarily anomaly detectors on side channel data. Finally, we define Artificial Immune Systems (AIS) and describe the AIS algorithms used in this work for circuit behavior classification [1].

2.1 Hardware Trojans

One of the major security problems for integrated circuits that has emerged in recent times, hardware Trojans are intentional malicious modifications made to an IC to leak sensitive information or provide back-doors for malicious activity. These modifications can be made at the design or fabrication phases by untrusted sources (eg. foundry or design house) [3].

Specific mechanisms called trigger and payload that vary in size, ranging from a small number for transistors to multi-million transistor designs are employed to activate hardware Trojans. The trigger periodically monitors specific signals in the circuit. In the event of a specific signal change, the payload is activated to allow the malicious change to occur in the circuit. The trigger conditions can sometimes be so rare that they may occur only once is many years, making the Trojan undetectable unless that event occurs again. Two types of Trojan triggers mainly occur – analog or digital. Analog triggers depend on natural occurrences such as temperature, EM emissions etc for activation. Whereas digital triggers consist of combinational and sequential circuits. Combinational Trojans depend on particular rare nodes wherein a rare condition occurs. They are stateless. Sequential Trojans however are typically more difficult to detect as activation conditions

[1]Parts of this chapter was published in © 2018 IEEE 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). Permission is included in Appendix A.

are set to be arbitrary, rendering these types of Trojans difficult to detect with standard verification techniques.

Developing effective defenses against hardware Trojans requires a comprehensive framework that provides a systematic study of their behavior. The first detailed taxonomy for hardware Trojans was developed by Wang, Tehranipoor, and Plusquellic [20]. The Trojan taxonomy was divided into three main categories depending on fundamental Trojan characteristics – physical, activation and action. Different hardware manifestations of the Trojan is described by the physical category. Activation refers to the triggering mechanisms that cause the Trojan to activate and perform malicious functions. Action category describes the malicious behavior, the Trojan has introduced in the circuit. With the sophistication of the hardware Trojan over recent years, nine new attributes were introduced to make the taxonomy more comprehensive. The most detailed taxonomy consists of five categories with each category consisting of different attributes. The main categories include insertion phase, abstraction level, activation mechanism, effects and location. [21] [22] [23].

2.1.1 Conventional Trojan Detection Approaches

Detecting hardware Trojans can be done by several methods: 1) logic testing, 2) side channel analysis, 3) reverse engineering via physical inspection, and 4) using built-in tests such as design for test (DFT) or design for security (DFS) techniques. However, the two main detection methods are logic testing and side channel analysis, as the other methods are not adequate enough to detect the most evasive of Trojans [3]

Logic testing aims to uncover Trojans through a compact testing procedure that aims to trigger as many low-probability conditions as possible within an IC; this methodology, called multiple excitation of rare occurrences (MERO) [5], requires that one finds the optimal set of vectors that causes the nodes with low-probability conditions to be activated. While performing this, it is important to note that certain nodes can be built to resist random-pattern testing, so it would be better to toggle several of them at once to uncover their true behavior. However, since this approach tests behaviour at a minute level, it is not good for detecting large Trojans because logic testing is unlikely to trigger the large number of inputs needed for said Trojans to occur. In addition, another shortcoming to this technique lies in the generation of test vectors that are suitable for Trojan detection.

On the other hand, side channel analysis is better suited for larger Trojans. This technique uses physical parameters or signals (such as current, delay, or thermal) whether a Trojan is present within the circuit; for instance, delays in a circuit can suggest that there were some new elements added to the circuit that should not be there. Typical approaches include transient current analysis, static current analysis, and path delay analysis, which can all be tested individually or collectively.

2.2 ML-based Hardware Trojan Detection Approaches

Several machine learning-based techniques have previously been proposed for Trojan detection, though these primarily seek to identify anomalies and classify circuits based on side channel measurements. Some of these techniques however require a golden reference model for detection to work. One such technique proposed by Iwase et al. [24] is a machine learning classification technique which is trained on differing power consumption to identify Trojan-free or Trojan-inserted designs in AES circuit. They used Discrete Fourier Transforms to convert acquired power consumption waveform data from the time domain to frequency domain as features to train the SVM. Another ML classification technique proposed by Lodhi et al. [25] uses a combination of timing signatures and classification algorithms for detecting HTs. They implemented a self-learning framework which uses their proposed macro synchronous micro asynchronous (MSMA) signature technique for feature extraction. A golden reference model is again used to extract features from MSMA which are then used to train KNN, decision trees, and Bayesian Classifiers. A ML technique that does not classify circuits based on side channel measurements but requires a golden model is proposed in [26], the IC's physical layout is extracted using reverse engineering and imaging techniques. Resulting scans of the IC layers are analyzed, and an SVM is trained to characterize Trojan-inserted and Trojan-free structures in the IC using features obtained from the imaging procedure. This RE technique does not require the need for generating a transistor or gate netlist, but requires high resolution, low noise scans for accurate classification. This may become more challenging with future process technologies and smaller feature sizes.

Other ML techniques that do no require a golden reference model and are not based on side channel measurement are proposed by Hasegawa et al. [27] and Ova et al. [28]. These include an SVM classifier trained to detect Trojans in a gate level netlist, using features such as flip-flop input/output and primary input/output for training, finding that dynamic weighting for SVM training gives an accuracy of 80% [27]. Alternatively, Oya et al. used a score-based classification technique for discerning between Trojan-free and Trojan-inserted netlists. Instead of identifying a netlist as an HT-inserted circuit, they focus on finding Trojan nets within these designs to classify it as a HT-inserted design. This approach enabled successful Trojan detection in certain TrustHub benchmarks. FANCI [29] and VeriTrust [30] are two other state-of-the-art techniques that use non-ML-based design stage verification on gate-level netlists. FANCI utilizes "control values" which depicts how the functionality of certain wires in the IC affects other wires to identify malicious wires that carry potential backdoor trigger signals. VeriTrust uses "tracers" and "checkers" to look for redundant inputs (suspicious Trojan signals) and determine if signals are carried by redundant inputs.

We use Artificial Immune Systems (AIS), an alternative class of ML techniques which have been widely used for malware detection in software. We do not consider side channel data or logic functions, and do not require simulation or silicon measurements of any kind. Instead, we aim to classify *high level behavioral traits* in a circuit, based on its control flow and data flow graph (CDFG), to detect the presence of unsafe and unwanted operations. Implementation specifics are abstracted out at the CDFG level, and the CDFG encapsulates both sequential (control flow) and combinational (data flow) behavior, enabling detection of RTL Trojans.

We do not consider side channel data or logic functions, and do not require simulation or silicon measurements of any kind. Instead, we aim to classify *high level behavioral traits* in a circuit, based on its control and data flow graph (CDFG), in order to detect the presence of such behaviors deemed to be unsafe or result in unwanted operations. Because implementation specifics are abstracted at the CDFG level, and the CDFG encapsulates both sequential (control flow) and combinational (data flow) behavior, it can be trained to identify a range of behaviors often associated with Trojan or Trojan-like behavior.

2.3 Artificial Immune Systems

The field of natural computing is primarily concerned with the theoretical analysis, understanding and modeling of biological phenomena or processes. As a part of this field are classes of algorithms that mimic processes that are observed in nature. Artificial Immune Systems (AIS) are a primary example of such algorithms; these systems are a computationally intelligent class of machine learning algorithms biologically inspired by the vertebrate immune system that aim to learn how to identify negative examples from positive examples. Research on AIS has resulted in many noteworthy algorithms and their applications extend to several real world problems and areas, including anomaly detection, pattern recognition, and optimization. In biological immune systems, the role of the immune system is that of a protector, wherein it fights and removes destructive micro-organisms such as bacteria or viruses and returns the body to a healthy state.

2.3.1 Overview of AIS

AIS is based on the theory that immune systems distinguish the pathogen (any malicious entity) as *non-self* and the body's cells (benign entities) as *self.* Detection of pathogens is described in terms of distinguishing "self" from "non-self" where "self" is synonymous with the normal functioning of the body and "non-self" as an anomaly or abnormality that is foreign to the body. However, not all pathogens need to be eliminated as most are not harmful to the body. Therefore, immune systems characterize "self" as anything that is harmless to the body and "non-self" as any pathogen that causes harm to the normal functioning of the body. The non-self cells are also equipped with appropriate defense mechanisms. The function of the immune system is to learn how to distinguish "self" from "non-self" through the process of self-replication and evolution. Hence, detection and elimination of harmful pathogens is a result of millions of types of immune cells circulating the body and interacting with other cells. The main immune cell taking part in the immune response is the lymphocyte whereas other types of cells called the phagocytic cells are secondary immune cells that help lymphocytes to eliminate pathogens. The two main types of lymphocytes are B cells and T cells which develop and mature in the lymphoid organs. Specifically, B cells develop and mature in the bone marrow whereas T cells develop in the bone marrow, but travel to the thymus to mature. The secondary lymphoid organs act as locations where antigen interactions occur to stimulate an immune response. Lymphocytes contain thousands of receptors on their surface that bind to pathogens based on their chemical structure. The likelihood of chemical bonding increases with their affinity. Immune cells that recognize a pathogen stimulate multiplication and separation of cells to produce clones (antibodies). Therefore, a large population of antibody-producing cells are generated that are specific to the exposed antigen. This reproduction process is called clonal expansion. Memory cells are also generated to stimulate immediate immune response when a similar antigens are exposed in the future. [31] [32]

These immune system inspired concepts of *antibodies* (self and benign entity) and *antigens* (malicious and non-self entity) are applied to the computational understanding of malicious and normal behavior of the system for accurate anomaly detection. Three fundamental immunological theories are the basis of AIS – clonal selection, negative selection, and immune networks. Research has been done on learning and memory mechanisms of the immune system and anomalous entity detection. By utilizing adaptive and innate immune responses, the immune system can fight the attacker. The immunological theories are based on adaptive and innate immune responses. The two basic types of immunity, adaptive and innate fight invading agents (i.e. pathogens). The innate immune response helps in initiation of immune responses and fight all kinds of pathogens in general whereas the adaptive immune response which primarily consists of lymphocytes fights specific pathogens. In this thesis, the research focus has been on anomalous entity detection using both Negative and Clonal Selection Algorithms.

2.3.2 Negative Selection Algorithm

Special white blood cells called T-cells, which are produced in the bone marrow, are primarily responsible for immunity in the body through the elimination of pathogens that are harmful to the body. During their development cycle, they usually undergo a maturation process in the thymus gland referred to as *T-cell tolerance* or *negative selection* [33, 34]. This procedure is important, as it removes "faulty" or harmful cells produced through cell division. From the biological perspective, harmful cells are determined by those T-cells that strongly bind with self-proteins. These T-cells are eliminated if this binding process activates the cells since they would harm self-cells that are native to the body. The recognition of *self* and *non-self* comes into play wherein cells that are self-reactive at the time of propagation are removed. Along the same lines, the *Negative Selection*

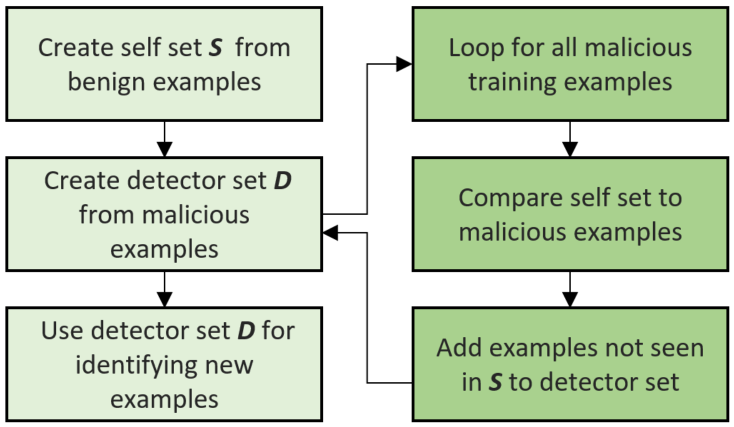

Create self set ***S*** from benign examples
Create detector set ***D*** from malicious examples
Use detector set ***D*** for identifying new examples
Loop for all malicious training examples
Compare self set to malicious examples
Add examples not seen in ***S*** to detector set

Figure 2.1: Overview of the Negative Selection Algorithm as Discussed in Section 2.3.2 [1]

Algorithm is mainly used to detect manipulation of data by viruses. The system is deployed with detectors that are trained to discriminate between manipulated data and the original data.

The Negative Selection Algorithm (illustrated in Figure 2.1) consists of two phases: the *detector generation* phase, and the *detector application* phase. These two phases can also be referred to as the *censoring* and *monitoring* phases, respectively [8]. They are both akin to the training and testing phases of machine learning, respectively. The censoring phase results in the generation of mature detectors that can be used for detecting non-self behavior. Subsequently, the system being protected is monitored for changes by the detectors generated in the censoring stage.

The algorithm (given as Algorithm 1) first defines a set of examples that describe typical IC behavior, denoted by S_{self}, which the AIS needs to know how to distinguish normal self-cells from non-self. The data that is to be protected (contained within S_{self}) is viewed in terms of a binary-encoded string. The string is then split into several l-length substrings to make up the set of S_{self} data. Once S_{self} is populated, we then generate a set of candidate detectors R of a specified length l from the binary-encoded string of CDFGs obtained from the selected Trojan benchmarks. We directly obtain candidate detectors from the Trojan benchmarks to generate a set of competent detectors in the censoring phase. Traditionally, random string generation is used to build the candidate detector set. However, using real examples to build the detector set reduces the likelihood of false negatives.

The candidate detectors in R are then matched to the entire self-set S_{self}. Strings from R that match any of those in S_{self} are eliminated from further consideration. On the other hand,

Algorithm 1: Negative Selection Algorithm

```
Let $R$ be set of strings generated from CDFG encodings;
Let $S_{self}$ be subset of strings that represents self-behavior;
Let $N$ be the desired number of detectors;
Create detector set $S_{non-self}$ made of non-self strings;
do
    forall strings $r \in R$ do
        if $r \notin S_{self}$ then
            Add $r$ to $S_{non-self}$;
        end
        else
            Remove string $r$ from $R$
        end
    end
while $size(S_{non-self}) \leq N$;
forall strings $r \in testcase$ do
    if $r \notin S_{non-self}$ then
        Label case as anomaly;
    end
end
```

the strings that do not match any of the strings in S_{self} become members of the final detector set collection $S_{non-self}$. This is repeated until all candidate substrings in R have been compared to those in S_{self}; at this point, we will have a representative detector set that is able to distinguish familiar cases with those that do not match typical behavior. During testing, changes to self are monitored by continually choosing each detector in R and testing to see if there is any match with strings in $S_{non-self}$. If the self-string matches one of the detector strings in $S_{non-self}$, this indicates that there is a Trojan within the IC.

2.3.3 Clonal Selection Algorithm

As a defense mechanism in vertebrate immune systems, when the body is under attack by antigens, the body massively produces cells that are capable of countering these antigens [35]. The clonal selection principle in immunology describes how the B and T cells of the lymphocytes are reproduced in adaptive immune response through the process of *affinity maturation* [36] – where cells are repeatedly exposed to antigens to develop stronger antibodies that are resistant to threats. Through affinity maturation, these antibodies would then be capable of binding to antigens, and these cells are kept and multiplied into clones to fight against the infection. Some of these cells

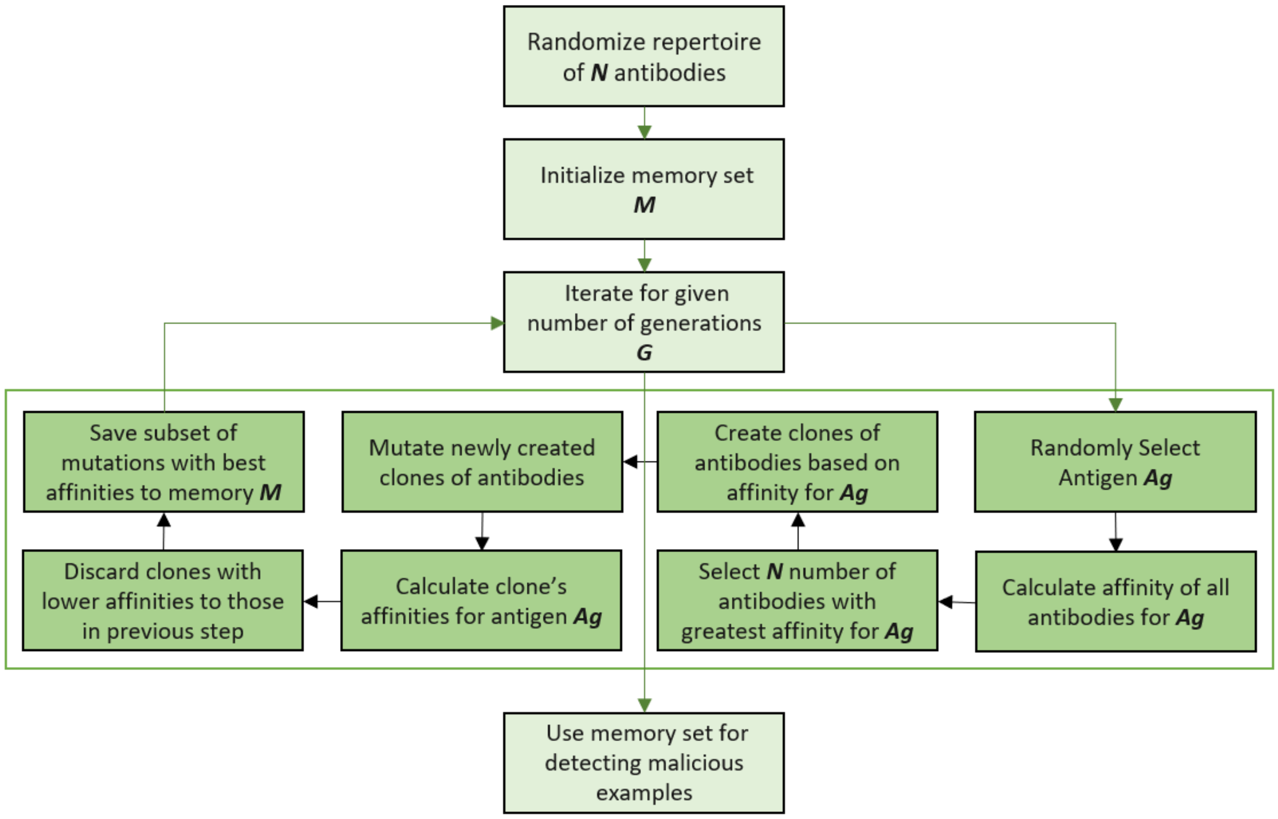

Randomize repertoire of N antibodies
Initialize memory set M
Iterate for given number of generations G
Save subset of mutations with best affinities to memory M
Mutate newly created clones of antibodies
Create clones of antibodies based on affinity for Ag
Randomly Select Antigen Ag
Discard clones with lower affinities to those in previous step
Calculate clone's affinities for antigen Ag
Select N number of antibodies with greatest affinity for Ag
Calculate affinity of all antibodies for Ag
Use memory set for detecting malicious examples

Figure 2.2: Overview of the Clonal Selection Algorithm as Discussed in Section 2.3.3 [1]

are also retained as memory cells that circulate over a period of time to fight remaining antigens that they recognize in the body. The computational *Clonal Selection Algorithm* (also known as CLONALG [16]) is based on this clonal selection theory, emphasizing on learning key attributes and maintaining memory of positive examples. This algorithm is inspired by Darwinian attributes, where only the fittest cells survive and are kept as antibodies to defend against antigen threats and that newer generations tend to vary through mutation.

The Clonal Selection Algorithm (illustrated in Figure 2.2) has mainly been used for pattern recognition and optimization tasks [14]. It learns about threats through the production of antibody *memory cells* that mutate when they encounter antigens and consequently acquire the characteristics of these antigens over time. Algorithmically, the process of affinity maturation [37] is used to randomly generate a set of antibodies that are similar to antigen (or foreign) examples based on the affinity or similarity between the antibody and the antigen. The maturation rate describes the amount of mutation that happens to a particular antibody, and this is directly proportional to the affinity of the cell to the encountered antigen. Antibodies with a high affinity are cloned and mutated with minimal changes as a result of affinity maturation; antibodies with a lower affinity are typically mutated to a higher degree. The cells then mutate to form clones and acquire the characteristics of the antigen. Long-term exposure of these memory cells leads to the final pool of memory cells that consist of all the antigen characteristics they were exposed to, resulting in an optimized pool of memory cells.

The algorithm (given as Algorithm 2) initially generates a random set of antibodies (i.e. benign examples) to produce a set of detectors. These antibodies are then exposed to antigens (i.e. malicious examples) to then determine whether they are suitable for detection or not, which is indicated by a high degree of affinity or similarity to the antigens. With exposure to antigens, the cloned detector set acquires the features of the subjected antigens, and over a period of time, it is able to generate a single optimized clone that reflects all the characteristics of the antigens to which it has been exposed. Higher affinity clones are added to a memory set, which is then used for classifying unseen cases as Trojan-inserted designs, after being exposed to training antigen examples.

The algorithm initially generates a random set of antibodies (self examples) S_{Ab} of a predetermined fixed size N_{pop}, and it generates a set of antigens (non-self examples) S_{Ag}, which are taken directly from Trojan-inserted designs. The algorithm also allocates a subset of generated clones S_{clones} and a memory subset M of clones selected for detection. The process of *generation* takes place, where each antibody in S_{Ab} is iteratively exposed to a pool of antigens S_{Ag}. For each antibody exposed to a randomly selected antigen Ran_{Ag} from S_{Ag}, its affinity value to the antigen is calculated using the Hamming distance; this process is referred to as affinity maturation [37]. Hamming distance is defined as the least number of substitutions needed to modify one string into the other, or the least number of errors that can transform one string into the other [38]. The antibodies with the highest affinity to the subjected antigens are cloned and added to S_{clones}. The rate of cloning ($rate_C$) on a single antibody in S_{Ag} is directly proportional to its affinity; basically, the higher the affinity, the more clones are produced. At the same time, clones are randomly mutated or altered to potentially increase their affinity. The rate of mutation ($rate_M$) however is inversely proportional to the affinity of a given antibody; in other words, the lower an antibody's affinity is, the more mutation occurs on the antibody on each generated clone. The process of affinity maturation is then repeated on the cloned antibodies to measure their new affinity values. The clones with

Algorithm 2: Clonal Selection Algorithm

```
Let $G$ be number of generations (iterations);
Let $M$ be memory set of detector clones;
Let $S_{Ab}$ be set of antibodies;
Let $S_{clones}$ be set of antibody clones;
Let $S_{Ag}$ be set of antigens;
Let $N_{pop}$ be the desired number of antibodies generated for each generation;
Let $rate_C$ be cloning rate and $rate_M$ be mutation rate;
do
    randomly pick an antigen $Ran_{Ag}$ from $S_{Ag}$;
    randomly generate antibodies for $S_{Ab}$ of size $N_{pop}$;
    forall antibody $Ab$ in $S_{Ab}$ do
        calculate affinity to $Ran_{Ag}$;
        if affinity of $Ab$ is above threshold then
            Create clones of $Ab$ using $rate_C$;
            Mutate clones of $Ab$ using $rate_M$;
            Add clones to $S_{clones}$;
        end
    end
    forall clone $C$ in $S_{clones}$ do
        calculate affinity to $Ran_{Ag}$;
        if affinity of $C$ is above threshold then
            Add clone $C$ to $M$;
        end
    end
while for all generations $G$;
Use memory set $M$ for classifying malicious examples;
```

the highest degree of affinity after this affinity maturation process are then saved in the detector memory subset M and the clones with the lowest degree of affinity maturation are replaced with other samples from R_{Ab}. After a certain number of generations G (or simply, iterations of affinity maturation), the memory set M is then ready to be used in detecting abnormalities.

CHAPTER 3

DETECTION METHODOLOGY

The Negative Selection and Clonal Selection Algorithms are used in the detection of hardware Trojans inserted in a high level circuit model, specifically behavioral Verilog HDL. Briefly, a static analysis tool is used to generate a control and data-flow graph from which features are extracted and used for training the AIS. For training and testing purposes, we use a set of RTL benchmarks from TrustHub [17, 18] labeled as either Trojan-inserted (non-self) or Trojan-free (self) [1].

3.1 CDFG Generation and Feature Extraction

The control and data-flow graphs of a program represent the internal sequence of operations or procedures that are performed to execute the program from beginning to end. A *control-flow graph* [39] provides a representation of all the paths that can be traversed through a program during its execution; in this graph, nodes represent basic blocks that express a sequence of consecutive statements, and edges represent any possible flow of control from one node to another. A *data-flow graph* [40] represents the data dependencies between different operations that a program performs and can be viewed as a fundamental expression of a computational structure. Thus, a CDFG fully encapsulates the behavior of the design, including any potentially unsafe or undesired behavior, regardless of how such behavior was incorporated into the design. The underlying assumption, then, is that the CDFG is wholly and correctly extracted from the source. By training an AIS to recognize patterns in the CDFG as potentially unsafe, whether the intent is malicious or not, it can detect non-self behavior in RTL designs. Furthermore, through the use of AIS and the evolutionary nature of the learning process, we propose that by training on example Trojan-included CDFGs, similar (if not exact) behavior in other designs may also be identified; thus, the training itself need

[1]This chapter was published in © 2018 IEEE 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). Permission is included in Appendix A.

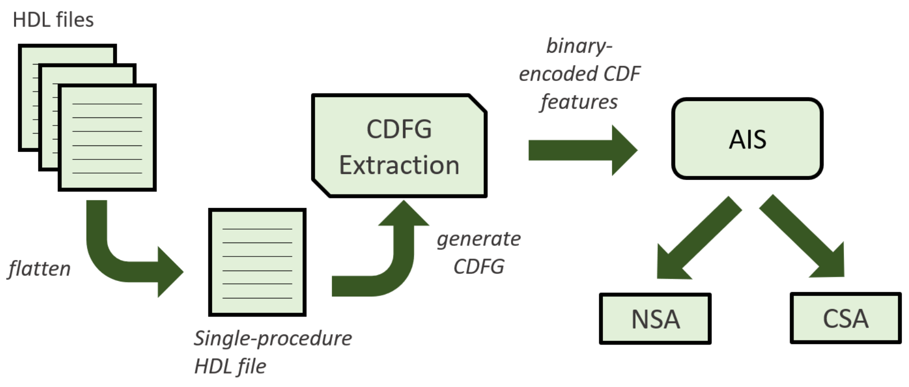


Figure 3.1: RTL Trojan Detection in CDFGs with Artificial Immune Systems [1]

not be comprehensive to have a high degree of accuracy on Trojans that have never been previously encountered Trojans.

The Python tool *PyVerilog* [19] is used to generate CDFGs for each benchmark. It initially constructs an abstract syntax tree representation from behavioral Verilog code. Within the design hierarchy, a tree traversal method is implemented to define signal scope and determine all parameters and constants. It then builds an assignment tree for every signal in the code to provide a complete representation of the control and data flow. As PyVerilog generates CDFGs for single-procedure Verilog files, each design must first be flattened to a single procedure. Control-flow and data-flow analysis is performed on these flattened benchmarks to generate CDFGs. Information regarding nodes and edges, isolated nodes, conditional and directed edges etc. are extracted as features by converting categorical data derived from CDF analysis to an adjacency matrix representation, which is binary encoded as strings for input to the AIS training procedure. The entire pipeline, from CDFG generation to feature extraction, is shown as Figure 3.1.

The format of the binary-encoded string is summarized in Figure 3.2. Specifically, we encode the graph as follows: first, We denote the structure type (input, output, reg, or wire) as a 2-bit *type string*. This highlights certain node types for the learning procedure. We then use the adjacency matrix to append an m-bit *edge connectivity string* derived from a single row of the matrix, where "0" indicates an edge does not exist between two nodes, and "1" indicates an edge does exist. The length of this string is equal to the number of nodes in the graph, m. Nodes with no outgoing edges (isolated nodes) have an edge connectivity string of all 0's. Appended to this is another string which indicates conditions that trigger a change in control flow. This is referred to as a *condition-based*

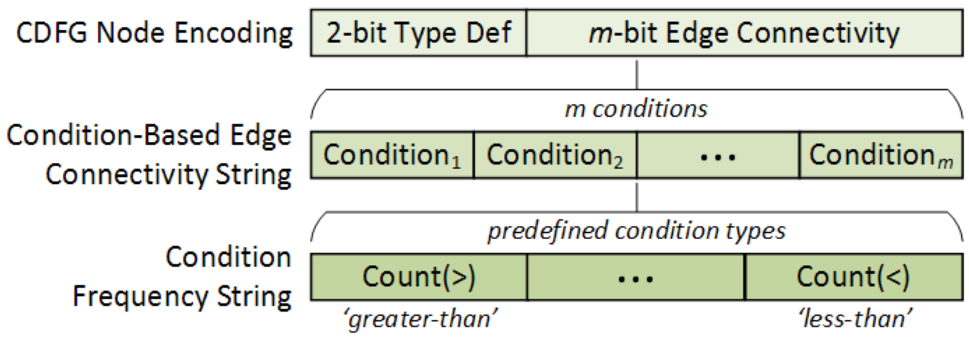


Figure 3.2: Binary Encoding of CDFG Nodes Used for AIS Training [1]

edge connectivity string (as shown in 3.2). We can account for conditions such as 'greater-than', 'less-than' or 'not equal to', etc. Finally, we can generalize the condition-based edge connectivity string by compressing it into a count of the number of edges for each type of condition. This is referred to as the *condition frequency* string.

Node and edge information are essential features in CDFGs, as their connectivity depicts differences in benign and malicious behavior. For example, as Trojan-inserted code essentially contains hidden characteristics, connectivity is generally indistinct whereas benign code is unambiguous and exhibits dynamic connectivity. In addition, there may be certain combinations or sequences of instructions which suggest unspecified functionality. This is where the frequency of conditions together with the flow of instructions of one type to another can be used by the AIS to learn normal and abnormal behavior.

3.2 Benchmarks

The training and testing data used in the experiments were taken from TrustHub [17, 18]. An overview of the benchmarks found in this repository and used in this work can be found in Table 3.1, and detailed descriptions of each benchmark is shown in Table 3.2. This repository consists of a variety of designs classified under different attributes such as levels of abstraction, the phase at which hardware Trojans were inserted, the intended location of the circuits (e.g. if the design is meant for input/output, power, etc.), and the harmful effects of those Trojans (e.g. denial-of-service or leaking of information). The Trojan behavior ranges from making malicious functional changes to leak data or manipulate instruction registers, to activating denial-of-service or degrading overall

Table 3.1: List of Hardware Trojan Benchmarks [1]

Benchmark	**# Trojan-Free**	**# Trojan-Inserted**
AES	21	21
MC8051	7	7
PIC16F84	4	4
RS232	10	10
wb_conmax	2	2
Other	6	0

Table 3.2: Attributes of the Hardware Trojan Benchmarks from TrustHub

Trojan Characteristics		**Benchmarks**				
		AES	***MC8051***	***PIC16f84***	***RS232***	***wb_conmax***
Effect	Leak Information	✓		✓		
	Denial of Service	✓	✓	✓		✓
	Change Functionality				✓	✓
Location	Processor	✓	✓	✓		
	I/O		✓			✓
	Power Supply		✓			✓
Activation	Always On	✓				
	Trigger Condition	✓	✓	✓	✓	✓

performance. For our experiments, we use a total of 50 Trojan-free and 44 Trojan-inserted RTL benchmarks.

3.3 Training and Testing the AIS

In this section, we discuss the AIS training and testing process. The training is done as a leave-one-out cross validation, where we train on all examples but one type of circuit in our benchmarks and test it on the remaining benchmark. This is done for all benchmark circuit types so that we have n AIS detectors for n circuit types. For example, we train one AIS on all circuit types except for AES, and the AES circuit examples would then be used in the testing phase, and for a second AIS, we train on all circuit types except MC8051 and the MC8051 circuit examples would then be used in the testing phase, and so on. We therefore train and test on all the benchmarks on batches of examples in a round-robin fashion until there is no significant change in accuracy or loss. The leave-one-out cross validation technique, therefore, enables testing on unseen benchmarks and as a consequence, the AIS learns a generalization of what makes a circuit a Trojan-inserted design so it can detect unseen cases.

3.3.1 Negative Selection Algorithm

The binary-encoded CDFG features are used as input to the AIS. During the censoring phase (i.e. the training phase), the AIS is trained on Trojan-inserted and Trojan-free examples to develop a mature detector set that contains binary strings which were detected as non-self. This process is done as follows: first, the program initially reads the benchmark CDFG binaries to create 32-bit binary strings which are then used to build the self set (i.e. benign features) from Trojan-free and and the detector set from Trojan-inserted examples. The binary string length is arbitrarily chosen based on the length of the generated CDFG binaries. Larger string lengths reduce feature loss. The self set is then compared against 32-bit strings of the Trojan-inserted binary encodings to build the detector set. The detector set contains strings from those examples which do not match Trojan-free binary-encoded features. This process is repeated until all training examples from the Trojan-inserted benchmarks have been given to the AIS.

After the censoring phase, the detector set is then applied to unseen (i.e. not used for training) binary-encoded CDFG benchmarks to determine whether they are Trojan-inserted or Trojan-free. In the censoring phase, either whole or partial string matching may be used to distinguish self from non-self strings. Using partial string matching shortens the time taken in string comparisons. In our experiments, we tested both to determine whether there is a significant difference in performance.

3.3.2 Clonal Selection Algorithm

As per the Clonal Selection Algorithm process in Section 2.3.3, we begin by defining the size of our population of antibodies and the size of our memory set. For our AIS, the generation phase uses a population size of 100, i.e. the randomly generated antibodies, and a memory set containing 50 detector clones. Each antibody (self example) and antigen (non-self example) is represented by 32-bit length strings. The antibody population set is subjected to an affinity maturation process with the antigen set consisting of 22 Trojan-inserted examples. As we discussed before, the memory set is used for classifying unseen cases, as it will contain antibodies which have a high affinity to the antigen (Trojan-inserted) examples.

The cloning rate determines how quickly the highest affinity antibodies are multiplied and reproduced, while the mutation rate is inversely proportional to the affinity of the antibody we are cloning. The higher the affinity, the lesser the effect of the mutation on the clones, such that the quality of the antibody is preserved. This generation process is continued for 50 iterations. At the end of every generation, we always try to keep the highest affinity out of all antibodies seen in memory and at the end of the generation phase. This ensures that we have the best antibodies for classification of new cases. For testing, we match the antigens (Trojan-inserted examples) to the memory set of high affinity antibodies for detection. If a string has a high affinity (i.e. high overlap), then it is classified as containing the target unsafe behavior.

CHAPTER 4

EXPERIMENTAL RESULTS AND ANALYSIS

For accurate anomaly detection, we must correctly classify both unsafe and safe behaviors. Errors in the first category, where unsafe behavior is not identified, are false negatives; errors in the second category, where safe behavior is identified as unsafe, are false positives. We use the terms correctly classified and incorrectly classified to describe true positive/true negative and false positive/false negative rates respectively. An ideal system would therefore maximize the detection accuracy for unsafe behavior (true positives), while minimizing false negatives and false positives. All of the experiments were run on 64-bit Ubuntu 16.04 with an Intel Core i7 processor and 16GB of RAM [1].

4.1 AIS Detection Accuracy

Both AIS algorithms demonstrated high accuracy in distinguishing between self and non-self behavior in the test dataset (Tables 4.1 and 4.2). With NSA partial string matching, a 10-bit fixed length substring from the 32-bit length string is selected and a sliding window is applied which, for every iteration, moves in position for the censoring and monitoring phases. As observed in Table 4.1, Trojan detection using the partial string matching technique has a detection accuracy range of 70% to 100%. For whole string matching, we use the generated 32-bit strings for training and testing. We observed a detection accuracy ranging from 80% to 100% in Table 4.1. For both techniques, we observe an average false negative rate of 12.6% and false positive rate of 14.8%.

Table 4.2 shows the detection accuracy of the implemented CSA with an observed average false negative rate of 12.8% and false positive rate of 14.7%. Results are shown in the form of a confusion matrix in Tables 4.3 and 4.4 where TT' and FF' represent the true positives and true negatives

[1]This chapter was published in © 2018 IEEE 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). Permission is included in Appendix A

Table 4.1: Results of Negative Selection Algorithm [1]

Benchmarks	**Correctly Classified**	**Incorrectly Classified**
	Partial String Matching	
AES	90.5%	9.5%
MC8051	71.4%	28.6%
PIC16f84	75%	25%
RS232	70%	30%
wb_conmax	100%	0%
	Whole String Matching	
AES	90.5%	9.5%
MC8051	85.7%	14.3%
PIC16f84	100%	0%
RS232	80%	20%
wb_conmax	100%	0%

(correctly classified as either benign or malicious), whereas FT' and TF' represent false positives and false negatives (incorrectly classified as either benign or malicious) respectively.

Table 4.2: Results of Clonal Selection Algorithm [1]

Benchmarks	**Correctly Classified**	**Incorrectly Classified**
AES	90.5%	9.5%
MC8051	85.7%	14.3%
PIC16f84	75%	25%
RS232	80%	20%
wb_conmax	100%	0%

As observed in Tables 4.1 and 4.2, AIS implements the Negative Selection and Clonal Selection Algorithms to efficiently detect the unsafe behavior for which the system was trained – in this case, malicious modifications – in the benchmark circuits test dataset. As with any machine learning-based technique, the accuracy can be improved by providing additional examples of malicious behavior from which the AIS model can learn, as well as AIS parameter tuning. Other features can be considered from the designs that can allow us to potentially yield better results. However, in this case, no attempt was made to train the AIS models beyond a general classification of "unsafe behavior" – properties such the specific structure, size, and integration of the Trojan in the RTL was not considered. In order to identify the type of threat that lies within a design, we would require these details as extra features to feed to the AIS during training. Nevertheless, we believe the results are promising for detecting generally unsafe behavior in CDFGs.

Table 4.3: Confusion Matrix for Negative Selection Algorithm [1]

Partial String Matching				**Whole String Matching**			
Benchmarks		**T′**	**F′**	**Benchmarks**		**T′**	**F′**
AES	**T**	19	2	AES	**T**	19	2
	F	2	19		**F**	2	19
MC8051	**T**	5	2	MC8051	**T**	5	2
	F	2	5		**F**	0	7
PIC16f84	**T**	4	0	PIC16f84	**T**	4	0
	F	2	2		**F**	0	4
RS232	**T**	8	2	RS232	**T**	7	3
	F	4	6		**F**	1	9
wb_conmax	**T**	2	0	wb_conmax	**T**	2	0
	F	0	2		**F**	0	2

Table 4.4: Confusion Matrix for Clonal Selection Algorithm [1]

Benchmarks		**T′**	**F′**
AES	**T**	20	1
	F	3	18
MC8051	**T**	6	1
	F	1	6
PIC16f84	**T**	3	1
	F	1	3
RS232	**T**	8	2
	F	2	8
wb_conmax	**T**	2	0
	F	0	2

Table 4.5: Overview of CDFG Generation and AIS Analysis Times

Benchmarks	**CDFG Generation Time (hrs)**	**File Size (kb) (≈)**	**Analysis Time(secs) (≈)**		
			Negative Selection		**Clonal Selection**
			Partial String Matching	**Whole String Matching**	
AES	30	183.4-269.1	126	118	128
MC8051	28	108.2-126.2	122	110	124
PIC16f84	19	106.0-114.6	125	116	126
RS232	26	166.6-193.8	126	120	127
wb_conmax	15	106.3-115.7	118	110	120

The proposed AIS can be used for narrowing down the quantity of designs that need to be further inspected for different classes of Trojans using another detector, which can take the form of another classifier not limited to AIS. Results can only be considered specific to the Trojan benchmarks used and require more amounts of training for the system to be effective in classifying malicious behavior from normal system behavior. In addition to insufficient training data, CDFG generation also proved to be a bottleneck. Firstly, CDFG generation with PyVerilog required designs to be flattened to single procedures, which was not the case for the benchmarks selected from TrustHub (and which are likely the case for other real designs). Furthermore, as observed in Table 4.5, for more complex benchmarks, the actual process of CDFG generation took significant time, upwards of several days for the most complex benchmarks (the file size in Table 4.5 refers to time taken to generate CDFGs for one file each of the respective benchmarks). Improvements in CDFG generation time for high level synthesis tools can reduce this time and make the flow more tenable.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis, a novel technique is proposed for the detection of hardware Trojans in RTL based on high level circuit behavior. I have employed the concept of an Artificial Immune System, a machine learning technique based on biological immune systems which aims to discriminate between self-behavior and non-self behavior. Binary-encoded CDFGs were extracted from TrustHub RTL Trojan benchmarks for training and testing the AIS. Results indicate that Negative and Clonal Selection, with their evolutionary-like learning processes, are capable of detecting certain behaviors in CDFGs on which the models were trained, indicating the presence of a Trojan in the hardware description.

Future work will consider ways to further improve accuracy by tuning some of the detector parameters. For example, in CSA, the population size, the memory size, cloning rate, and mutation rate can all be adjusted, and in NSA, we can tune the size of the substring for partial string matching and compare varying positions of these substrings. False negatives can be further reduced by selecting features most appropriate in classifying self and non-self behavior, training and testing on larger datasets, improving the encoding process etc. Determining the optimal or ideal variables for both NSA and CSA may provide additional insight. Understanding what specific features of the CDFG are best for classification and why these certain features work the best will provide valuable insight into detecting Trojan-like behavior in CDFGs. Furthermore, exploring the features that can be obtained from these designs that can lead to determining the type of hardware Trojan detected in the design and, more importantly, the location of the hardware Trojan in the design. Reverse engineering the detection process to identifying where the unsafe behavior was detected in the original source would provide hardware designers and system integrators with a valuable tool for improving security of hardware systems during the design stage.

LIST OF REFERENCES

[1] F. Zareen and R. Karam, "Detecting RTL trojans using artificial immune systems and high level behavior classification," in *2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 68–73, Dec 2018.

[2] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware trojan: Threats and emerging solutions," in *High Level Design Validation and Test Workshop, 2009. HLDVT 2009. IEEE International*, pp. 166–171, IEEE, 2009.

[3] 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, 2014.

[4] S. Narasimhan, X. Wang, D. Du, R. S. Chakraborty, and S. Bhunia, "Tesr: A robust temporal self-referencing approach for hardware trojan detection," in *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pp. 71–74, IEEE, 2011.

[5] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "Mero: A statistical approach for hardware trojan detection," in *Cryptographic Hardware and Embedded Systems-CHES 2009*, pp. 396–410, Springer, 2009.

[6] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital ip cores," in *Hardware-Oriented Security and Trust (HOST), 2011 IEEE International Symposium on*, pp. 67–70, IEEE, 2011.

[7] N. Fern and K.-T. T. Cheng, "Detecting hardware trojans in unspecified functionality using mutation testing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 560–566, IEEE Press, 2015.

[8] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer," in *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pp. 202–212, Ieee, 1994.

[9] J. O. Kephart *et al.*, "A biologically inspired immune system for computers," in *Artificial Life IV: proceedings of the fourth international workshop on the synthesis and simulation of living systems*, pp. 130–139, 1994.

[10] P. Zhang, W. Wang, and Y. Tan, "A malware detection model based on a negative selection algorithm with penalty factor," *Science China Information Sciences*, vol. 53, no. 12, pp. 2461–2471, 2010.

[11] E. Al Daoud, "Metamorphic viruses detection using artificial immune system," in *Communication Software and Networks, 2009. ICCSN'09. International Conference on*, pp. 168–172, IEEE, 2009.

[12] Z. Guo, Z. Liu, and Y. Tan, "An nn-based malicious executables detection algorithm based on immune principles," in *International Symposium on Neural Networks*, pp. 675–680, Springer, 2004.

[13] K. A. Al-Sheshtawi, H. Abdul-Kader, and N. A. Ismail, "Artificial immune clonal selection classification algorithms for classifying malware and benign processes using api call sequences," *International Journal of Computer Science and Network Security*, vol. 10, no. 4, pp. 31–39, 2010.

[14] L. N. De Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE transactions on evolutionary computation*, vol. 6, no. 3, pp. 239–251, 2002.

[15] K. C. Tan, C. K. Goh, A. Mamun, and E. Ei, "An evolutionary artificial immune system for multi-objective optimization," *European Journal of Operational Research*, vol. 187, no. 2, pp. 371–392, 2008.

[16] L. N. De Castro and F. J. Von Zuben, "The clonal selection algorithm with engineering applications," in *Proceedings of GECCO*, vol. 2000, pp. 36–39, 2000.

[17] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pp. 471–474, IEEE, 2013.

[18] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, 2017.

[19] S. Takamaeda-Yamazaki, "Pyverilog: A python-based hardware design processing toolkit for verilog hdl," in *International Symposium on Applied Reconfigurable Computing*, pp. 451–460, Springer, 2015.

[20] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pp. 15–19, IEEE, 2008.

[21] S. Bhunia and M. M. Tehranipoor, *The Hardware Trojan War: Attacks, Myths, and Defenses.* Springer, 2017.

[22] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE design & test of computers*, vol. 27, no. 1, 2010.

[23] H. Li, Q. Liu, and J. Zhang, "A survey of hardware trojan threat and defense," *Integration, the VLSI journal*, vol. 55, pp. 426–437, 2016.

[24] T. Iwase, Y. Nozaki, M. Yoshikawa, and T. Kumaki, "Detection technique for hardware trojans using machine learning in frequency domain," in *Consumer Electronics (GCCE), 2015 IEEE 4th Global Conference on*, pp. 185–186, IEEE, 2015.

[25] F. K. Lodhi, I. Abbasi, F. Khalid, O. Hasan, F. Awwad, and S. R. Hasan, "A self-learning framework to detect the intruded integrated circuits," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, pp. 1702–1705, IEEE, 2016.

[26] C. Bao, D. Forte, and A. Srivastava, "On application of one-class svm to reverse engineering-based hardware trojan detection," in *Quality Electronic Design (ISQED), 2014 15th International Symposium on*, pp. 47–54, IEEE, 2014.

[27] K. Hasegawa, M. Oya, M. Yanagisawa, and N. Togawa, "Hardware trojans classification for gate-level netlists based on machine learning," in *On-Line Testing and Robust System Design (IOLTS), 2016 IEEE 22nd International Symposium on*, pp. 203–206, IEEE, 2016.

[28] M. Oya, Y. Shi, M. Yanagisawa, and N. Togawa, "A score-based classification method for identifying hardware-trojans at gate-level netlists," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pp. 465–470, EDA Consortium, 2015.

[29] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: identification of stealthy malicious logic using boolean functional analysis," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 697–708, ACM, 2013.

[30] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "Veritrust: Verification for hardware trust," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1148–1161, 2015.

[31] S. A. Hofmeyr and S. Forrest, "Architecture for an artificial immune system," *Evolutionary computation*, vol. 8, no. 4, pp. 443–473, 2000.

[32] D. Dasgupta, S. Yu, and F. Nino, "Recent advances in artificial immune systems: models and applications," *Applied Soft Computing*, vol. 11, no. 2, pp. 1574–1587, 2011.

[33] "immunological tolerance and autoimmunity,"

[34] J. A. Owen, J. Punt, S. A. Stranford, *et al.*, *Kuby Immunology*. WH Freeman New York, 2013.

[35] F. M. Burnet *et al.*, "A modification of jerne's theory of antibody production using the concept of clonal selection.," *Australian J. Sci.*, vol. 20, no. 3, pp. 67–9, 1957.

[36] G. D. Victora and M. C. Nussenzweig, "Germinal centers," *Annual review of immunology*, vol. 30, pp. 429–457, 2012.

[37] R. G. Weinand, "Somatic mutation, affinity maturation and the antibody repertoire: a computer model," *Journal of Theoretical Biology*, vol. 143, no. 3, pp. 343–382, 1990.

[38] M. X. He, S. V. Petoukhov, and P. E. Ricci, "Genetic code, hamming distance and stochastic matrices," *Bulletin of mathematical biology*, vol. 66, no. 5, pp. 1405–1421, 2004.

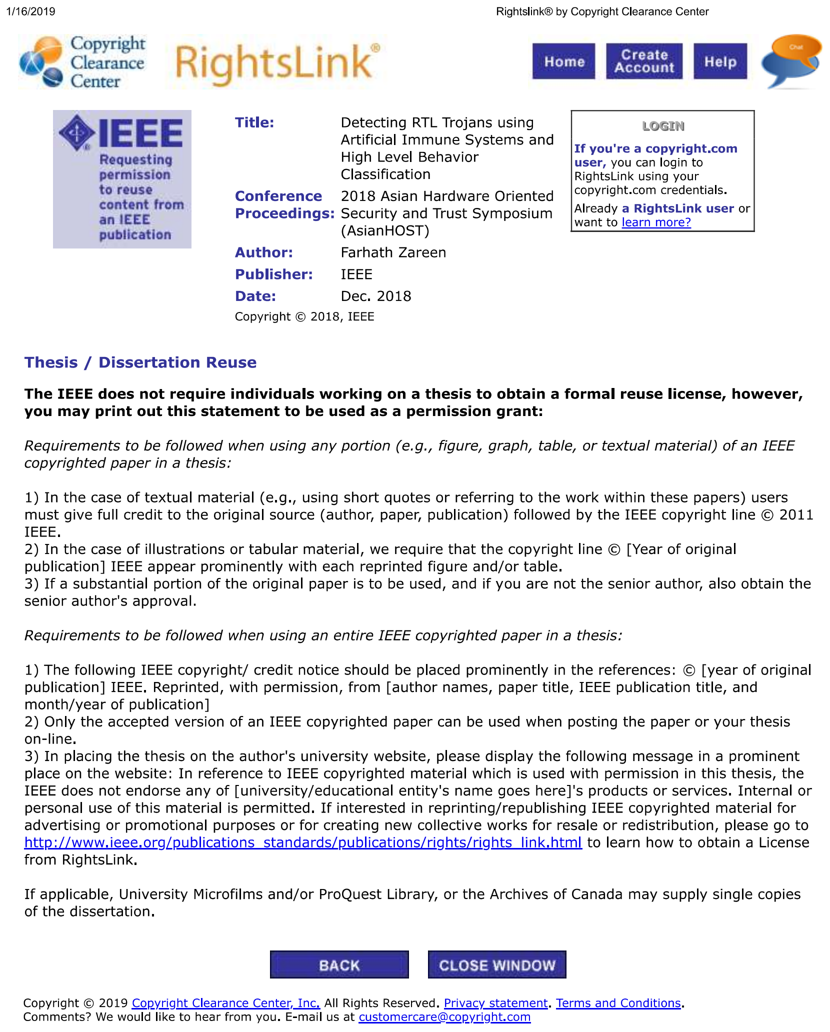
[39] F. E. Allen, "Control flow analysis," in *Proceedings of a Symposium on Compiler Optimization*, (New York, NY, USA), pp. 1–19, ACM, 1970.

[40] M. C. Williamson and E. A. Lee, "Synthesis of parallel hardware implementations from synchronous dataflow graph specifications," in *Conference Record of The Thirtieth Asilomar Conference on Signals, Systems and Computers*, pp. 1340–1343 vol.2, Nov 1996.

APPENDIX A

COPYRIGHT PERMISSIONS

Copyright permission for content used in Chapter 1, 2, 3, 4, 5 and 6. © 2018 IEEE. Reprinted with permission from Farhath Zareen, Robert Karam, Detecting RTL Trojans using Artificial Immune Systems and High Level Behavior Classification, 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST).



1/16/2019 Rightslink® by Copyright Clearance Center

Copyright Clearance Center RightsLink® Home Create Account Help

IEEE Requesting permission to reuse content from an IEEE publication

Title:	Detecting RTL Trojans using Artificial Immune Systems and High Level Behavior Classification
Conference Proceedings:	2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)
Author:	Farhath Zareen
Publisher:	IEEE
Date:	Dec. 2018

Copyright © 2018, IEEE

LOGIN

If you're a copyright.com user, you can login to RightsLink using your copyright.com credentials.
Already **a RightsLink user** or want to learn more?

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW

Copyright © 2019 Copyright Clearance Center, Inc. All Rights Reserved. Privacy statement. Terms and Conditions.
Comments? We would like to hear from you. E-mail us at customercare@copyright.com