Malware Detection in Embedded Devices using Artificial Hardware Immunity

Farhath Zareen (fzareen@usf.edu)
University of South Florida

Mateus Augusto Fernandes Amador
University of South Florida

Robert Karam
University of South Florida

Research Article

Keywords: IoT Security, Artificial Immune Systems, Machine Learning, Hardware Malware Detectors

Posted Date: April 13th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2758367/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.
Malware Detection in Embedded Devices using Artificial Hardware Immunity

Farhath Zareen · Mateus A. Fernandes A. · Robert Karam

1 Introduction

Rapid growth of digital electronics and wireless communications has enabled the usage of embedded systems in our daily lives – from smart home and personal digital assistants, to more critical healthcare and military applications. Internet of Things (IoT) devices are embedded with technology and connect wirelessly to a network with the ability to transmit data. IoT devices are constant targets to malicious attacks as they typically have long in-field lifetimes, providing potential attackers with both remote and physical access. Malware is malicious software that threatens the IoT devices, as it is carefully designed to infect a device and impair the target system [24]. Moreover, as the metamorphic and polymorphic nature of malware makes it hard to detect, it can rapidly spread across diverse platforms [9] in networked environments. A prime example is the October 2016 Distributed Denial of Service (DDoS) attack caused by the Mirai Botnets [20], a type of malware that turned IoT devices running a Linux kernel into bots to launch a large-scale DDoS attack, causing a widespread internet shutdown. These type of malicious attacks can target more critical infrastructure, including electric grids, food production infrastructures, and military systems, threatening our daily life. According to the 2021 SonicWall annual threat report, 56.9 million attacks were launched on IoT devices (a 66% increase in IoT malware attacks compared to 2019 [1]). Traditional security techniques such as signature-based and behavior-based anti-virus software cannot be applied to securing IoT devices due to resource constraints such as low computing power and limited energy supply [26, 31, 32, 36, 37]. A hardware supported solution can tackle these design challenges, as it can streamline
the process of detecting and responding to malware attacks.

Current state-of-the-art research in IoT security against malware attacks explores the use of Hardware Performance Counters (HPCs) and machine learning (ML) models to build hardware malware detector (HMD) \[5,16,38\]. Since malware functionality typically does not change in hardware, it is more difficult to evade a HMD; the immutability of hardware therefore serves as the root of trust. However, many embedded devices cannot support malware detectors due to limited or no hardware monitoring infrastructure \[34\]. Moreover, design challenges of HMDs include small logic area and power overhead for implementation on processor, and small detection latency which includes reading HPC and running ML classifiers. While HMDs provide high accuracy for small systems where real-time output is important, due to these inherent constraints, we are investigating a complimentary HMD technique for microprocessors that leverage \(n\)-gram opcode sequences and concepts of Artificial Immune Systems (AIS) for detecting malicious activity. \(N\)-gram opcode sequences can provide contextual analysis of the behavior of malware in real-time for signature-less detection of malware and can be computationally efficient in analyzing large data. AIS based systems require less computational power, time and lower latency overheads compared to traditional ML techniques, making them well suited for resource constrained IoT environments. This motivates us to investigate alternative solutions where we aim to answer the following research questions:

1. Is it possible to achieve comparable accuracy results for malware detection with \(n\)-gram opcode sequence analysis as it is with HMDs that may use HPCs?
2. Can AIS be used to reduce the on-chip ML inference such as power, area and latency compared to traditional ML-based HMDs?

In this work, we present a Hardware Immune System (HWIS) that serves as a real-time anti-malware solution for low-power, resource constrained and network facing embedded devices. Specifically, we make the following novel contributions:

1. Introduce a novel and highly effective AIS-based botnet detector for RISC-V binaries and evaluate the detection accuracy on malware samples in simulation.
2. Analyze trade-offs in static botnet classification using \(n\)-gram opcode sequences, evaluating full and partial matching on 5- to 15-gram sequences, varying class boundaries, and sequence frequency analysis.
3. Propose an optimal subset of 128 6-grams with class threshold \(T_c\) that gives a detection accuracy of 96.7\% with F1-score of 0.96 on real malware samples.
4. Implement the proposed hardware AIS, including an efficient analog partial string comparator, and evaluate the power, performance, and area using a combination of SPICE with 32nm low power Predictive Technology Models (PTM) \[42\] and the Synopsys 32nm EDK.

The rest of the paper is organized as follows: In Section 2 we outline the threat model for embedded IoT devices and discuss the background on traditional hardware-based malware analysis techniques and AIS based ML classifiers. We provide detailed description of the methodology in Section 3, Section 4 presents experimental results and analysis, and Section 5 discusses these results in context of related work and trade-offs in more detail. Finally we conclude in Section 6.

2 Background

In this section, we describe the threat model, discuss recent HMDs, and provide background on static malware analysis and ML techniques used in this paper.

2.1 Threat Model

IoT devices have limited amount of memory (Flash, SRAM, and EEPROM) and processing capacity. Due to their resource-constrained nature, IoT devices are unlikely to run a sophisticated anti-virus tool. Attack models in the IoT realm work under the assumption that a vulnerable device is not fully protected (such as with no password or antivirus tool, or a default password). These vulnerabilities can be exploited to install malware and launch a wide range of attacks such as:

1) Information leakage wherein sensitive information is leaked by malware to eavesdrop and perform acts of espionage for financial gain, 2) Incorrect code/system execution where adversaries deploy malware that can execute incorrect code leading to system communication failure, 3) DDoS attacks, where malware can flood a target system with traffic causing significant degradation in performance, leading to system failure and shutdown, and 4) Adversarial attack, a new type of attack that targets the malware detector tool itself to launch attacks, leading to unpredictable consequences such as failing to prevent malware or terminating a benign program unintentionally \[21\]. In this paper, we address DDoS attacks caused by botnets and introduce a
novel approach to detect botnet behaviors in embedded IoT.

2.2 Hardware Malware Detectors

HMDs analyze the processor’s internal components and attempt to classify malicious behavior based on changes in the processor’s state. Much of the current HMD research focuses on analyzing built-in HPCs – hardware counters that count the hardware events and track important hardware activity within the CPU - and building ML models to detect malicious and benign code execution [5, 10, 38]. However, using HPCs to predict malware may lead to higher false positives and system performance slowdowns [5], and may be constrained to specific malware types. Additionally, underlying assumption for HPCs is that a clear distinction can be made in the micro-architectural values measured between a malware-infected systems and benign systems. However, there is no concrete evidence that high-level malicious activity is reflected in low-level micro-architectural events [43]. An alternative to HPCs is analyzing the instruction flow itself using opcodes extracted from the processor pipeline in real-time. In this paper, we show that this is a viable approach both from the perspective of classification accuracy, as well as hardware implementation.

2.3 n-gram Opcode based Malware Detection

Empirical evidence [4, 13, 39] shows that attackers utilize existing malware codes and make modifications to add new behavior. Opcode (operational code) specifies the operation to be performed within the CPU datapath. Several studies have shown that frequency of opcodes have been used to discriminate a malicious software from a trusted one [6, 14]. Additionally, n-grams have been widely used for malware detection [3, 8, 29]. In this paper, we investigate the effectiveness of sequences of opcodes, instead of frequency of opcodes, in detecting malware, and in particular, botnets. We use these opcode sequences, extracted through static code analysis, to train the AIS, with the goal of real-time hardware-assisted malware detection.

2.4 Artificial Immune Systems

AIS is a class of ML that is based on bio-immune-system inspired concepts of antibodies (self and benign entity) and antigens (malicious and non-self entity). AIS has been applied to solve problems in the domain of anomaly/intrusion detection [2, 30, 40], pattern recognition and several optimization problems. They have successfully been applied to protect TCP/IP/WSN/MANET networks in the dynamic IoT realm [23, 28, 33]. Our research focuses on investigating learning and memory mechanisms to classify malware in hardware using AIS, specifically, Negative Selection Algorithm (NSA). Earlier studies on the implementation of an HWIS was proposed by Bradley et al. [7], to build bio-inspired fault tolerant systems using a finite state machine representation of the system to be protected. In this work however, we investigate the feasibility of using HWIS as an anti-malware solution in embedded IoT wherein variable length opcode sequence matching is performed between an antibody (AB) detector set and unknown samples built from run-time execution patterns to detect malicious activity.

3 Methodology

The primary goal of this research is to evaluate the efficacy of AIS NSA in identifying malware (botnets, in particular), and develop an area- and energy-efficient circuit-level implementation that can integrate in an IoT-targeted microprocessor. To begin, we performed static analysis on malware samples (botnets) and extracted n-gram opcode sequences. These sequences were fed into the NSA training model to produce a set of AB detectors which were tuned to discriminate between benign and malicious code. These detector sets were tested on unseen samples, using F1-score as a single unified metric to compare the efficacy of different detector sets, n-gram lengths, and subset sizes within the design space. Finally, a circuit-level implementation of the detector was co-designed to evaluate the impact on area and power.

3.1 Software Simulation

We created a feature set based on binary n-gram opcode sequences of benign and malware code, which were used by the AIS for training and testing purposes. The process is illustrated in Fig. 1. Broadly, we adopted the following steps:

1. Collected malicious and benign samples in C and assembled to RISC-V instructions using the RISC-V toolchain.
2. Converted RISC-V assembly opcodes to their binary equivalent using the RISC-V GNU compiler toolchain.
3. Split the binaries into n-gram opcode sequences, where n ranges from 5 to 15.
4. Applied AIS NSA for training and testing to distinguish between malware and benign applications.

3.1.1 Data Pre-processing

The RISC-V GNU compiler tool chain, which is the GNU GCC cross-compiler for RISC-V ISA, was used to obtain RISC-V assembly code from the C samples using the -S flag. Each generated assembly file was then converted to its 32-bit binary string equivalent. The RISC-V ISA defines a 7-bit opcode field, which were extracted from the binary strings to create \( n \)-gram opcode string sequences for building the feature set, which in this case is the AB detector set. This detector set was then used to test the AIS against unknown malware samples. Training and testing phases were performed using 10-fold cross validation over 100 iterations.

3.1.2 NSA Training and Testing

NSA typically consists of two phases: 1) Censoring and 2) Monitoring phases. The censoring phase corresponds to training whereas monitoring phase tests the classifier [11]. At the end of the censoring phase, mature antibodies used to distinguish non-self behavior from self behavior are generated. Subsequently, the system being protected can be monitored for changes by the AB detector set generated in the censoring stage (Algorithm 1).

At the start of the training process, each \( n \)-gram binary string generated from the benign samples form a self set, which characterize normal behavioral patterns. After populating the self set, a set of candidate antibodies that exhibit malicious behavior were identified from exposure to the malware samples. The candidate antibodies were then matched to the entire self set. Any string that matches with strings in self was eliminated. On the other hand, the strings that do not match any of the strings in self were added to the detector set comprising of antibodies. This process was repeated until all candidate opcode sequence strings were evaluated. After completion of the censoring phase, a representative AB detector set comprising of malicious behavioral traits was obtained. This AB detector set was then used in the monitoring phase to test the AIS.

To make the final detector set more robust against inaccurate classification, we implemented a class threshold \( (T_c) \) technique that determined the hit-count of strings in the testing sample. \( T_c \) served as a boundary to distinguish between malicious and benign behavior. For example, a \( T_c \) of 0.01 required that 1% or more of a testing sample’s \( n \)-gram strings matched with those in the detector set. Thus, this threshold defined number of detector hits to be met to classify unseen samples as malicious. Results for class threshold-

---

**Algorithm 1:** Creating a detector set using NSA

Let \( R \) be \( n \)-gram str set generated from binary encodings;
Let \( S_{self} \) be subset of strings representing self-behavior;
Let \( S_{non-self} \) be final detector set consisting of \( M \) non-self strings;

```plaintext
forall strings \( r \in R \) do
  if \( r \notin S_{self} \) then
    Add \( r \) to \( S_{non-self} \);
  end
end
```

Let \( T \) be list of \( n \)-gram strings generated from test case binary encodings;
Let \( H \) be number of detector hits required to classify sample as malicious;
Let \( h \) be counter for number of detector hits;

```plaintext
forall strings \( t \in \) testcase \( T \) do
  if \( t \in S_{non-self} \) then
    \( h = h + 1 \);
  end
  if \( h \geq H \) then
    Label case as anomaly;
    return;
  end
end
```
ing technique on 6-gram opcode sequences are shown in Table 2 and discussed in more detail in Section 3.3

### 3.1.3 Downsizing a Detector Set

The size of a detector set depends greatly on the size of \( n \) for \( n \)-gram strings, where the larger the value of \( n \), the more possibilities for unique strings exist. This can lead to power and area issues when porting a detector to a resource constrained embedded device. We therefore tested the efficacy of detector subsets using a fixed AB size of \( M \), which is significantly smaller than the original number of antibodies \( N \). In this work, values of \( M \) ranging from \( 2^4 \) to \( 2^8 \) are evaluated and compared in accuracy to the full detector set. To determine which of the \( N \) antibodies should be selected to form a detector subset, we perform a historical study of strings in the detector set over each trial to build a final detector subset consisting of the top \( M \) frequently seen opcode-sequence strings. Fig. 2 shows frequency of opcode sequences for a 6-gram detector set. Historical matching data from 200 testing trials was used to determine the top strings in the detector subset. This approach allows for a detector subset that can more easily be ported to an embedded device. It can also be used in the reverse engineering process to analyze the most frequently seen opcode instruction sequences exhibiting malicious behavior to determine opcode-commonality in different botnet families. Results from the subset testing are shown in Table 3 and are discussed in more detail in Section 3.2.2

![Fig. 2 Opcode sequence frequency for 6-gram detector sets.](image)

### 3.2 Hardware Implementation

The proposed HWIS was implemented in hardware with the proposed CPU integration shown in Fig. 3(a). During execution, the HWIS receives each opcode in \( IR[6:0] \) and builds packets for analysis. As soon as HWIS detects malicious activity, it interrupts the CPU through the flag signal to begin a recovery service routine. Example routines may vary from simply notifying another network node, administrator or gateway of the issue depending on the IoT architecture, to more complex recovery behavior which is implementation dependent. Depending on the deployment strategy (bare metal/RTOS, lightweight kernel), recovery strategies may vary. In all cases, malicious activity detection had three main steps running simultaneously:

#### 3.2.1 Opcode collection

Opcodes were extracted from \( IR[6:0] \) and incorporated into a sliding window packet (PKT) as shown in Fig. 3(b). The PKT was implemented as a 7n-bits shift-register that left-shifted 7-bits each clock cycle. Thus, holds a string \( s \) of \( n \) opcodes in sequence each cycle. The complement, \( \neg s \), was continuously computed for future use in the AB comparison.

#### 3.2.2 Antibody comparison

The string \( s/\neg s \) stored in the packet ran through an \( m \)-column AB memory array (Fig. 3(b)) used for malware detection. All AB columns were compared to the strings \( (s/\neg s) \) in parallel. If the string \( s/\neg s \) matched with any of the antibodies, then hit was asserted.

Each AB (Fig. 3(c)) held \( n \)-rows AB opcode sub-cells array (AB\(_{op} \)) one for each 7-bit opcode. Each AB opcode sub-cell (Fig. 3(d)) contained 7 memory cells, as static random-access memory (SRAM) bitcells, connected to a 7-bit equality comparator, implemented with seven 2-input XNOR gates (Fig. 3(f)) and one 7-input AND gate (Fig. 3(e)). This compared the respective 7-bit opcode in the string \( s \) from the current packet to its respective AB memory sub-cell. A 100% match asserted the associated match line \( (match_k = V_k) \), charging the respective match line capacitor \( C_{in} \) in Fig. 3(g).

These \( C_{in} \) capacitors were part of a partial comparator (PartComp) in Fig. 3(g). The \( C_{in} \) and a \( C_{bias} \) capacitors were connected to a \( V_{in} \) line, which connected to gates of a CMOS inverter as shown in Fig. 3(g). This configuration converted the CMOS inverter to a multi-input floating-gate MOS (FGMOS) inverter as described in [22]. The \( V_{in} \) line had a voltage value equivalent to its trapped charge \( Q_{in} \) plus the sum of capacitance times their voltages all divided by the total sum of capacitance \( C_T \) [22]. By inserting the variables from (Fig. 3(g)), the voltage on \( V_{in} \) was expressed as:

\[
V_{in} = \frac{C_{bias}V_{bias} + \sum_{k=0}^{n-1}C_{in}V_k + Q_{in}}{C_{bias} + n * C_{in}}
\] (1)
where $n$ is the total number of opcodes in $s$ (i.e., $n$-gram), and assuming negligible transistor parasitic capacitance.

Note that all $V_k$ voltages were connected to the AB sub-cell match line in Fig. 3(c), which only was either high ($V_{dd}$) or low (0V). Additionally, $V_{bias}$ was grounded (0V) and the trapped charge $Q_{in}$ can be removed using a layout-based technique during the fabrication process as suggested in [27]. This simplifies (1) to:

$$V_{in} = \frac{\text{match}_op \cdot C_{in} \cdot V_{dd}}{C_{bias} + n \cdot C_{in}}$$  \hspace{1cm} (2)$$

Where $\text{match}_op$ was the number of matching opcodes in $s$ with the respective AB.

The transistors had a threshold voltage $V_{th}$ where the transistors turn on/off. Hence, if $V_{in} > V_{th}$, then the first transistor outputted low at $V_x$; otherwise, it outputted high at $V_x$. Then, a second CMOS transistor inverted, intensified, and sped up this $V_x$ voltage transition. In other words, if $V_{in} > V_{th}$, then $\text{hit} = 1$; otherwise, $\text{hit} = 0$. However, this $V_{th}$ cannot be modified for different desired comparison opcode thresholds $T_{op}$ since it depends on the physical characteristics of the transistors used. Alternatively, by modifying the bias capacitor $C_{bias}$ size in (2), then $V_{in}$ can be configured lower at design time. Thus, if $T_{op}$ is the minimum desired number of opcode matches for a hit, then by replacing $\text{match}_op = T_{op} - \delta$ and $V_{in} = V_{th}$ in (2) we can approximate the desired bias capacitance $C_{bias}$ as follows:

$$C_{bias} = C_{in} \cdot (T_{op} - \delta) \cdot \frac{V_{dd}}{V_{th} - n}$$  \hspace{1cm} (3)$$

Where $\delta$ was approximately 0.5 to set the $V_{th}$ between $\text{match}_op = T_{op}$ and $T_{op} - 1$. This $\delta$ was be slightly differ to improve and balance the hit rising and falling edge latencies. $\delta$ can be calculated during SPICE simulation testing.

3.2.3 Hit count

After the AB array sent a hit, a detector circuit (Fig. 3(b)) counted the number of sequential hits in a hit-count register. However, for every cycles without any hit asserted, the hit-count was decremented. As soon as the hit-count reached threshold ($[m \cdot T_c]$), interrupt flag was raised to start the CPU’s recovery service routine. Specific functions within a recovery service routine would necessarily vary in different implementations, and would depend on various factors including the deployment strategy, presence of an OS, IoT architecture, and others.

4 Results

Correct classification of both malicious and benign characteristics is required for accurate anomaly detection. In this work, we used the terms correctly classified to describe accuracy, which is the fraction of samples that were classified to their true class (true positive (TP) and true negative (TN)) out of all samples, i.e., TP and TN summed with false positive (FP) and false
negative (FN). We also evaluated the performance of our AIS model in terms of accuracy, precision, recall, and F1-score, and visualized this using the ROC curve and Area Under the ROC Curve (AUC) (Fig. 4). These metrics are defined by (4)-(7).

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}
\]

(4)

\[
\text{Precision} = \frac{TP}{TP + FP}
\]

(5)

\[
\text{Recall} = \frac{TP}{TP + FN}
\]

(6)

\[
\text{F1-score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}
\]

(7)

![ROC curve for 6/6 n-gram malware class for varying subset sizes at \(T_c = 0.05\).](image)

Fig. 4 ROC curve for 6/6 n-gram malware class for varying subset sizes at \(T_c = 0.05\).

4.1 AIS Detection Accuracy

Table 1 shows detection accuracy and F1-scores for n-grams of lengths ranging from 5 to 15. Based on these results, we chose 6-grams for our hardware design due to the smaller AB detector set size and high accuracy. We also tested several subset sizes of the AIS AB detector set to evaluate trade-offs in accuracy as we reduce the number of detectors. In Table 2, we observed that as the threshold increases, the detection accuracy drops with an increase in the percentage of incorrectly classified samples. This is mainly because malware samples are likely to have non-malicious strings (i.e. not all malicious strings will be a hit in the detector set – since NSA uses self set to only isolate malicious strings, malicious samples still have benign strings) and benign samples may sometimes have detector set hits, although rarely as the length of n-grams and threshold percentage increase. Table 3 shows the accuracy for subsets 16-256 and is compared to the full detector set of 329 antibodies. As the subset size decreases, we also observed varying results, where a smaller size of 128 and 256 detectors performed better than all 329 detectors based on detector hit frequency. This is due to the selection of the most frequently observed detectors across several trials of cross validation, which would also eliminate less frequent detectors that may not necessarily suggest malicious behavior. However, removing too many detectors can be detrimental, as creating detector sets of sizes of 16, 32, and 64 exhibited lower performance due to fewer samples. This would then result in fewer hits to the detector set. Our results are also impacted by the disassembly process from C code to RISC-V assembly code, as some system calls were not define and incompatible instructions were replaced with a NOP instruction \texttt{ADDI x0, x0, 0}. This could also be a potential cause for mis-classification of malicious activity.

**Table 1** Results of whole string matching using NSA for \(T_c = 0.01\)

<table>
<thead>
<tr>
<th>n-gram</th>
<th>Unique strings</th>
<th>Correctly Classified%</th>
<th>Incorrectly Classified%</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>88</td>
<td>75.0</td>
<td>25.0</td>
<td>0.66</td>
</tr>
<tr>
<td>6</td>
<td>329</td>
<td>95.1</td>
<td>4.9</td>
<td>0.95</td>
</tr>
<tr>
<td>7</td>
<td>940</td>
<td>93.7</td>
<td>6.3</td>
<td>0.94</td>
</tr>
<tr>
<td>8</td>
<td>2045</td>
<td>91.2</td>
<td>8.8</td>
<td>0.92</td>
</tr>
<tr>
<td>9</td>
<td>3445</td>
<td>91.2</td>
<td>8.8</td>
<td>0.92</td>
</tr>
<tr>
<td>10</td>
<td>4688</td>
<td>91.8</td>
<td>8.2</td>
<td>0.92</td>
</tr>
<tr>
<td>11</td>
<td>5600</td>
<td>92.6</td>
<td>7.4</td>
<td>0.93</td>
</tr>
<tr>
<td>12</td>
<td>6375</td>
<td>94.4</td>
<td>5.6</td>
<td>0.95</td>
</tr>
<tr>
<td>13</td>
<td>6892</td>
<td>95.0</td>
<td>5.0</td>
<td>0.95</td>
</tr>
<tr>
<td>14</td>
<td>7264</td>
<td>95.3</td>
<td>4.7</td>
<td>0.95</td>
</tr>
<tr>
<td>15</td>
<td>7643</td>
<td>96.0</td>
<td>4.0</td>
<td>0.96</td>
</tr>
</tbody>
</table>

**Table 2** Results of opcode-sequence string matching using NSA for 6-gram with varying \(T_c\)

<table>
<thead>
<tr>
<th>(T_c)</th>
<th>Correctly Classified%</th>
<th>Incorrectly Classified%</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>95.1</td>
<td>4.9</td>
<td>0.95</td>
</tr>
<tr>
<td>0.02</td>
<td>93.9</td>
<td>6.1</td>
<td>0.93</td>
</tr>
<tr>
<td>0.03</td>
<td>87.2</td>
<td>12.8</td>
<td>0.85</td>
</tr>
<tr>
<td>0.04</td>
<td>72.2</td>
<td>27.8</td>
<td>0.61</td>
</tr>
<tr>
<td>0.05</td>
<td>68.8</td>
<td>31.2</td>
<td>0.54</td>
</tr>
</tbody>
</table>
Table 3 Results of subset opcode-sequence string matching using NSA for 6-gram with hit count ≥ 1

<table>
<thead>
<tr>
<th>Detector Size</th>
<th>Correctly Classified%</th>
<th>Incorrectly Classified%</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>64.2</td>
<td>35.8</td>
<td>0.43</td>
</tr>
<tr>
<td>32</td>
<td>67.8</td>
<td>32.2</td>
<td>0.51</td>
</tr>
<tr>
<td>64</td>
<td>78.8</td>
<td>21.2</td>
<td>0.72</td>
</tr>
<tr>
<td>128</td>
<td>96.7</td>
<td>3.3</td>
<td>0.96</td>
</tr>
<tr>
<td>256</td>
<td>97.2</td>
<td>2.3</td>
<td>0.97</td>
</tr>
<tr>
<td>329</td>
<td>95.1</td>
<td>4.9</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Table 4 Power and Area for 6-gram AB Array

<table>
<thead>
<tr>
<th>Component</th>
<th>Qty.</th>
<th>Power (µW)</th>
<th>Area (µm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>42</td>
<td>6.74E-03</td>
<td>1.60E02</td>
</tr>
<tr>
<td>INV1</td>
<td>6</td>
<td>1.28E00</td>
<td>7.62E00</td>
</tr>
<tr>
<td>NOR3</td>
<td>6</td>
<td>2.82E00</td>
<td>1.68E01</td>
</tr>
<tr>
<td>NAND3</td>
<td>12</td>
<td>6.74E00</td>
<td>3.35E01</td>
</tr>
<tr>
<td>XNOR2</td>
<td>42</td>
<td>3.10E01</td>
<td>2.13E02</td>
</tr>
<tr>
<td>Comparator</td>
<td>1</td>
<td>1.61E00</td>
<td>2.93E00</td>
</tr>
</tbody>
</table>

Total (1 AB) [Ovhd. %] 4.35E01 [1.90E-02%] 4.34E02 [4.06E-02%]

Total (128 AB) [Ovhd. %] 5.57E03 [2.42E00%] 5.56E04 [5.20E00%]

Total (329 AB) [Ovhd. %] 1.43E04 [6.22E00%] 1.43E05 [1.34E01%]

4.2 Power, Performance, and Area

Table 4 shows the hardware components used per AB, their power consumption, area, and respective overheads. The overhead was based on an example generic RISC-V core with 230mW power and 1.07mm² area. The power and area were calculated using 32nm low power PTM model on SPICE and Synopsys 28/32nm library respectively, except for the partial comparator. For a 6-gram partial comparator (Fig. 3(g)), it required 2 inverters, 6 $C_{in}$ capacitors (one for each opcode match line) and one $C_{bias}$ capacitor with size at most 5 * $C_{in}$ from (3) using $T_{op} = 6$, $V_{dd} = 1V$, $V_{ih} = 509mV$, $m = 6$. If $C_{in}$ area is 0.035µm² [13] and each inverter size is 1.27µm² (from Synopsys 28/32nm library), then the partial comparator area for 6-gram is approximately 2.93µm². The power and area of the AB array grew approximately linearly to the number of AB. For example, a 128 AB subset of 6-gram detectors had a power of 5.57mW (2.42% overhead) and area 5.56E04µm² (5.20% overhead) (Table 4), whereas the full set of 329 ABs had a power of 14.3mW (6.22% overhead) and area 1.43E05µm² (13.4% overhead). These results demonstrated that using smaller detection subsets is ideal for lower power and area overheads compared to using the full-set, especially considering there was no significant impact on the accuracy. Furthermore, when considering PKT and detector blocks in Fig. 3(b), it only added a power of 343µW (+0.15% overhead) and area of 560µm² (+0.053 overhead) to the 6-gram HWIS design. These overhead values were not as significant as the AB array with ≥100 AB.

From a timing standpoint, the HWIS was not situated within the CPU datapath, and instead branched off the datapath and ran in parallel. Hence, it was sufficient for the match circuit to operate at least as fast as the main CPU datapath. The critical path latency through the AB memory was found to be approximately 640ps using SPICE simulation, with the partial comparator contributing the majority of the delay (540ps, around 82%). If also considering the path delay from the packet (85ps) and the detector to register the hit input (85ps in Fig. 3(b)), then the total required latency for the propose HWIS design was 826ps. This suggested a maximum clock frequency of about 1.2GHz. Although using the maximum clock frequency may only allow time for the detector (Fig. 3(b)) to register the hit input but not enough time to send the flag interrupt signal to start the CPU’s recovery mode, this does not cause a significant overhead as the recovery mode can be delayed by 1 CPU clock cycle. For faster matching, it is possible to replace the partial comparator with a fully digital comparator that does not provide a threshold match feature, and may incur additional overhead for longer n-grams.

4.3 Partial Comparator Behavior

Fig. 3(a)-(c) show the SPICE simulation for the partial comparator (Fig. 3(g)) implementing 4/6, 5/6, and 6/6 matching. The $C_{bias}$ values used were 71fF, 27fF, 46fF, with $\delta = 0.6, 0.58, 0.62$ respectively. All configurations used $C_{in} = 10fF$, $V_{dd} = 1V$, $V_{bias} = 0V$, $V_{ih} = 0.509V$, and 32nm low power CMOS PTM [12]. The $V_{in}$ voltage was proportional to the number of matching opcode $match_{op}$ (or charged $C_{in}$). When $V_{in}$ crossed the transistor threshold voltage $V_{th}$, it switched $V_{out}$ on/off as shown in Fig. 5. By setting higher $C_{bias}$, it can reduce $V_{in}$ for the same $match_{op}$ values, allowing higher opcode matching thresholds $T_{op}$. For lower $T_{op}$ it will required to increase $V_{bias}$ voltage and calculate $C_{bias}$ with (3) but then divide by $1 - \frac{V_{th}}{V_{in}}$, as follows:

$$C_{bias} = \frac{C_{in} * ((T_{op} - \delta) * \frac{V_{dd}}{V_{in}} - n)}{1 - \frac{V_{th}}{V_{in}}}$$  (8)
Malware Detection in Embedded Devices using Artificial Hardware Immunity

5 Discussion

5.1 Improving Detection Accuracy

Through our experiments we show that our technique is successful in accurately detecting botnet behavior. While the opcode-based malware detection approach is effective, more efficient features can be extracted that describe malicious patterns in malware to improve detection accuracy. For example, longer instruction sequences may be used such as opcode + destination register, opcode + source registers etc. To evaluate which $n$-gram features are most effective for training the AIS, dynamic analysis may be performed. We observed during software simulations that, for longer $n$-grams, the detector size tends to be larger. While this improves accuracy, it may result in higher power and area overheads when implemented in hardware. Therefore we analyzed historical data of the frequency of opcode sequences to determine which of the $N$ antibodies should be selected to form a detector subset. These top strings then formed a detector subset. One advantage of this approach is that it can easily be mapped to an embedded device. Additionally, reverse engineering can be performed to analyze which of these frequently seen sequences exhibit

---

**Fig. 5** Waveforms for (a) 4/6 Partial comparator, (b) 5/6 Partial comparator, and (c) 6/6 Partial comparator.

**Table 5** A summary of recent HMD based IoT security techniques

<table>
<thead>
<tr>
<th>Ref</th>
<th>Data Source</th>
<th>Attack Type</th>
<th>Detection Technique</th>
<th>Platform</th>
<th>Power (mW)</th>
<th>Area (ns)</th>
<th>Accuracy (%)</th>
<th>F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>[35]</td>
<td>HPC</td>
<td>Botnet, Trojan, Rootk</td>
<td>RCNN</td>
<td>65nm CMOS</td>
<td>3.82E-02</td>
<td>2.40E-01 mm²</td>
<td>96.8</td>
<td>0.96</td>
</tr>
<tr>
<td>[12]</td>
<td>HPC</td>
<td>Worm, Virus, Trojan, Backdoor</td>
<td>ML Classifiers</td>
<td>Xilinx Virtex 7</td>
<td>-</td>
<td>156 Slices* (JRip)</td>
<td>90.0</td>
<td>0.93</td>
</tr>
<tr>
<td>[37]</td>
<td>HPC</td>
<td>Android Malware, Side-channel</td>
<td>ML Classifiers</td>
<td>Xilinx Virtex 7</td>
<td>3.50E02 (JRip)</td>
<td>156 Slices* (JRip)</td>
<td>91.5</td>
<td>0.91</td>
</tr>
<tr>
<td>[25]</td>
<td>IoT-23</td>
<td>Malware</td>
<td>NN</td>
<td>Zynq 7000 SoC</td>
<td>1.88E03</td>
<td>28248 Slices*</td>
<td>99.4</td>
<td>0.99</td>
</tr>
<tr>
<td>[10]</td>
<td>ETB</td>
<td>Malware</td>
<td>ML Classifiers</td>
<td>Xilinx 7</td>
<td>1.67E03</td>
<td>-</td>
<td>86.7</td>
<td>0.98</td>
</tr>
<tr>
<td>[41]</td>
<td>Opcodes</td>
<td>Botnets</td>
<td>AIS</td>
<td>Xilinx Virtex 7</td>
<td>1.14E2</td>
<td>10081 Slices* (KNN)</td>
<td>96.7</td>
<td>0.96</td>
</tr>
<tr>
<td>This work</td>
<td>Opcodes</td>
<td>Botnets</td>
<td>AIS</td>
<td>32nm CMOS</td>
<td>5.91E00</td>
<td>5.62E-02 mm²</td>
<td>96.7</td>
<td>0.96</td>
</tr>
</tbody>
</table>

* Area for FPGAs is calculated using the sum of utilized unit slices (i.e., LUTs + FFs + DSPs).
malicious behavior, allowing for further understanding of botnet behaviors to apply better detection mechanisms and improve accuracy.

Although hardware-supported malware detectors have advantages such as real-time detection and a low power and performance overheads adversaries have devised evasive techniques to bypass HMD defense mechanisms. As demonstrated by [19, 21], it is feasible to reverse engineer HMDs to replicate system behavior. Due to the self-learning nature of AIS, it can be used as a potential solution to prevent such adversarial attacks where the detector set can be randomized as well as retrained by refreshing the hit counter. These potential solutions will be explored in the future to improve the AIS to prevent such attacks. Moreover, while we focused on botnets as a specific class of malware in this work, the proposed model can be trained on other malware families. The proposed approach can 1) serve as one of several layers - this one at the hardware level - for improved security assurance and 2) be expanded to other classes of malware that may target resource-constrained IoT devices.

5.2 Implementation of Antibody Match Circuit

The circuit-level implementation contains a comparison mechanism that supports partial matches, e.g., matching fewer than \( n \) opcodes in sequence. However, it is currently serving as an analog full comparator, with \( C_{\text{bias}} \) chosen such that all six opcodes in the packet must match a particular AB to signal a hit. While a pure digital string comparison would be faster, and consequently reduce the critical path delay, the match circuit already meets the delay requirements. Moreover, the area characteristics favor the proposed implementation over the digital comparison, while the power was found to be comparable. Finally, a purely digital comparison would not easily allow for partial matching, at least without significant additional area or latency requirements. Partial matching technique also requires further investigate in the future as a means to improve redundancy against adversarial attacks, or for potentially supporting other classes of malware besides botnets.

5.3 Memory Technologies

The current iteration of the AB memory functions as a kind of Content Addressable Memory (CAM) [17] where parallel search is needed, but decoding of the match line generating the hit is not. The antibodies are stored in SRAM bitcells, which natively provide the true and compliment of the current value. These are in turn used by the particular XNOR implementation to reduce the overall transistor count (Fig. 31). A nanoscale non-volatile memory would be preferable to SRAM in this implementation, especially as the antibodies will not need to be rewritten after power loss, and external memory would not be occupied by the permanent AB storage. However, this will still require careful consideration of the trade-offs. In particular, write endurance and write current are of less concern, because it is not anticipated the antibodies would need to be updated with significant frequency. Read latency and read current should be minimal, as should area. As the complement of the stored AB value may not necessarily be available, an alternate memory may incur additional overhead due to one additional inverter per bitcell; however this may balance out by freeing non-volatile memory otherwise reserved for the permanent storage of the AB memory contents.

5.4 Related Works

A summary of recent HMD-based implementations is presented in Table 5. A mixed signal AI prototype that acts an HMD and accurately detects Malware and micro-architectural attacks from HPC traces using novel reservoir computing architecture is implemented in [35]. This reservoir computing HMD uses raw HPC traces as inputs to a reservoir computing neural network (RCNN). The authors implement their technique on a prototype RCNN fabricated in 65nm CMOS process. Their technique achieves a classification accuracy of 96.8%, precision of 0.94 and recall of 1 on Malware dataset.

In [12], the authors propose Adaptive-HMD, which is an ML-based framework for online malware detection using micro-architectural events collected from HPC registers. Their online malware detection process consists of an ensemble of ML-detectors and the selection model aims to find the best performing ML model for detecting a particular class of malware at runtime, given their observed HPC features. Their base ML detectors included SMO, MLP, OneR, RepTree, and JRip, out of which results for their best performing model, JRip, are shown in Table 5. Their technique achieves a detection rate of \( \approx 94\% \). To analyze the hardware implementation cost, the authors leverage Vivado HLS compiler to develop the HDL implementation of the classifiers on Xilinx Virtex 7 FPGA. Adaptive-HMD’s implementation cost consists of two parts – cost of the selection model and selected base model. Implementation cost of their technique is shown in Table 5.

When detecting malicious patterns at the hardware level in edge devices, there are fewer computa-
tion resources for processing complex workloads. To aid in solving this issue, the authors in [37] have proposed a lightweight hardware-assisted micro AI enabled countermeasure against emerging malware and side-channel attacks for securing modern edge devices. In particular, their technique applied various types of ML classifiers and compared them across different evaluation metrics such as detection accuracy, F-measure, ROC curve analysis, computational latency, and hardware overhead to determine the most accurate and cost-efficient ML classifiers to enable on-device micro AI countermeasures for detecting signature of emerging cyber-attacks including malware and side channel attacks and securing modern edge devices at hardware level. Hardware implementation of their technique makes use of Xilinx Vivado Design suite to synthesize ML classifiers for Xilinx Virtex 7 FPGA. Latency and power estimation are collected at 10ns clock cycle time (Table 5).

Similarly, [26] presents an efficient anomaly-based IDS framework for IoT gateways and edge devices using NN on the new IoT-23 dataset. Table 5 illustrates hardware resource usage of their technique, which is an FPGA implementation on a PYNQ-Z2 board using a Zynq-7000 SoC and Vivado HLS compiler. Their technique achieves a detection accuracy of 99.7%. In [10], the authors repurpose embedded trace buffers (ETB) which is an on chip circular memory buffer used for post-silicon validation available in modern processors, for malware detection and classification. This technique incurs zero area overhead to the original microprocessor core and achieves detection accuracy of 86.7% against malware. Their hardware implementation consists of two alternatives – on-chip and off-chip. In an on-chip implementation, one of the cores runs the ML classifier whereas in the off-chip implementation, the ML-classifier can be placed in an FPGA or ASIC co-processor.

While these recent works predominantly use HPC traces for malware detection and performs FPGA-based circuit-level implementations, our work makes use of n-gram opcodes while leveraging AIS for detection. We also evaluate hardware realization feasibility through a circuit-level implementation of our design. We utilized a combination of SPICE with 32nm low-power PTM and Synopsys 32nm EDK and compared the power/area/latency overhead with RISC-V CPU implementation. While we cannot perform direct comparison with many parameters, qualitatively our technique is compared to the state of the art in Table 5. We compare FPGA based hardware implementation of AIS [11] with our design. Table 5 compares both implementations where the FPGA implementation utilized a total of 4661 LUTs, 5420 FFs, 0 DSPs, 0 BRAMs, and consumed 11.4mW, or about a 8.5%, 11.8%, 0%, 0%, and 0.6% overheads, respectively. In comparison with [35], to evaluate lower area/latency overheads, we used traditional general scaling techniques with dimensions scaling factor $\kappa = 65/32 \approx 2.03$ and voltage scaling factor $\nu = 1.20$. The equivalent power ($x[1/\nu^2]$), area ($x[1/\kappa^2]$), and latency ($x[1/\kappa]$) for 32nm CMOS are 26.53mW, 0.0582mm$^2$, and 12.32$\mu$s, respectively. Our implementation outperformed their design in terms of area ($x[1.036]$ smaller) and latency ($x[14.8E3]$ faster). Although their power may appear negligible compared to our designs, the significantly different clock frequencies used affected the power values. Instead, Energy Delay Product (EDP) ($= \text{Energy} \times \text{Latency} = \text{Power} \times \text{Latency}^2$) gave a more accurate comparison: our design has an EDP of 4.07E-21 J*s, while theirs has 4.02E-15 J*s for equivalent 32nm CMOS, an efficiency improvement of nearly 10$^6$. Our developed AIS-based technique performs static analysis to measure and compare the efficacy of different detector sets, n-gram lengths, and subset sizes within the design space. The circuit-level implementation of our design achieves low power, area, and latency overheads for seamless integration into an IoT-targeted microprocessor and achieves high detection accuracy with no delay impact and fewer computational resources.

6 Conclusion

In this paper, we have presented a novel ML-based approach for introducing security in microprocessor architectures with hardware as root-of-trust, by leveraging concepts of Artificial Immune SystemS to facilitate a built-in Hardware Immune System (HWIS). HWIS can serve as a real-time stand-alone anti-malware solution for low-power, resource constrained and network facing embedded IoT devices. We conducted software simulations where we performed static analysis on n-gram opcode sequences to train and test the AIS. We then performed circuit-level SPICE simulations and reported power, area, and delay overheads based on a combination of SPICE 32nm low power PTM [12] and Synopsys 28/32nm EDK library estimates, compared to a 28nm RISC-V CPU as a baseline. Our approach demonstrates a detection accuracy of 95.1%, F1-score of 0.95, power overhead 6.37%, and area overhead 13.45% for full detector set of size 328. Alternatively, for lower hardware overhead, we found an accuracy of 96.7%, F1-score of 0.96, power overhead 2.42%, and area overhead 5.20% for a detector subset of size 128. Regardless of the set size used, our implementation has no delay impact on the processor using a maximum clock speed of
1.2GHz. Our results indicate that AIS with its evolutionary nature and self-learning capabilities, can efficiently detected botnet behavior with low power, area, and performance overheads. This technique can also be used to detect various types of other malware behavior in HMDs. Future work will consider ways to address these attacks and increase security assurance for IoT devices. We will consider longer n-grams with a lower match threshold, pairing with performance counters, and utilizing dynamic analysis techniques which may provide greater assurance against adversarial and other attacks.

Declarations

Competing Interests: The authors declare no competing interests.

Authors’ Contributions: All authors contributed equally.

Availability of data and material: The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Ethical Approval: not applicable

Funding: not applicable

References

Electron Devices Meeting, pp 24.7.1–24.7.4, DOI 10.1109/IEDM.2011.6131608


