Journal of Systems Architecture 109 (2020) 101827

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

IoT Device security through dynamic hardware isolation with cloud-Based update

Festus Hategekimana[a], Taylor JL Whitaker[b,*], Md Jubaer Hossain Pantho[b], Christophe Bobda[b]

[a] University of Arkansas, USA
[b] University of Florida, USA

ARTICLE INFO

Keywords:
Internet of things security (IoT)
Isolation and protection
Cloud-based learning
Network traffic classification
Field programmable gate arrays (FPGA)

ABSTRACT

This work proposes a novel approach to provide comprehensive security to IoT devices. Our approach is based on a reconfigurable hardware-based isolation and protection mechanism (IPM) that operates as a dynamic separation unit between devices and network, far from potential software manipulation. The IPM analyses communications for malicious activities and prevents damage to the IoT device. The IPM leverages a central cloud-based authority to broaden the scope of traffic analysis beyond that of a singular IoT device. The central server evaluates logs from all IPM-protected IoT devices to improve their defense mechanisms and periodically upgrade device IPMs through a remote secure provisioning mechanism. The IPM achieves a 98.68% detection rate when evaluated against a Neptune DoS attack.

1. Introduction

Progress in embedded computing technology and the increased availability of broad-band connection have led to the growth of network-connected devices, the logical result of which is the global availability of small devices through the internet. In this regard, Internet of Things is being used for general scenarios where internet connectivity and computing capability empower objects that may use sensors and actuators to interact with their environment [10,15,26]. The advent and fast growth of Internet of Things (IoT) creates immense opportunities and benefits for our society. Smart home appliances such as smart lighting and smart meters are becoming mainstream. IoT can enable smart cities through data captured from thousands of networked sensors spread across a geographic region and can improve patient care in the healthcare sector through intelligent and networked devices around patients. IoT can also be useful in the energy sector to increase the level of power system automation and remote accessibility to deliver information to a wide range of users in real-time and also to control the number of tasks involved to streamline operations and performance [22]. Besides services to communities, IoT is predicted to be a future driving force in the economy. IDC for instance expects the worldwide market for IoT solutions to grow at a 20% CAGR from $1.9 trillion in 2013 to $7.1 trillion in 2020 and Gartner estimates that IoT product and service suppliers will generate incremental revenue exceeding $300 billion in 2020 [40]. Many enterprise-technology firms are betting on growth in the emerging Internet-of Things to offset declines elsewhere [32].

While the benefits of IoT are undeniable, the reality is that security is not keeping up with the pace of innovation [39]. The advent of new protocols such as IPv6 whose fundamental goal is to allow every networked device to have globally unique IP address is turning traditional isolated embedded devices such as printers, microscope, and other sensors in IoT devices directly available over the internet. As opposed to workstations and servers, most of these devices lack proper protection mechanisms since they were designed with price, efficiency and power constraints in mind. Also, they have very limited capability to run complex protection tools such as anti-virus and firewalls. As consequence, billions of devices are being made available without protection in internet, at the mercy of malicious programs such as botnets. Successively hijacking those devices can lead the theft of sensitive data and loss of consumer privacy, interruption of business operations, slowdown of internet functionality through large-scale distributed denial-of-service attacks, and potential disruptions to critical infrastructure. Because the dependence on properly functioning networks to drive so many life-sustaining activities, IoT security is now considered a matter of homeland security by the department of homeland security [39].

Proposed security methods in IoT devices follow the same embedded system approaches, all of which tackle a special case, including intrusion detection, reference monitors, cryptography algorithms, dynamic infor-

* Corresponding author.
E-mail addresses: fhategek@uark.edu (F. Hategekimana), t.whitaker@ufl.edu (T.J. Whitaker), mpantho@ufl.edu (M.J. Hossain Pantho), cbobda@ece.ufl.edu (C. Bobda).

https://doi.org/10.1016/j.sysarc.2020.101827
Received 10 February 2020; Received in revised form 17 April 2020; Accepted 19 June 2020
Available online 29 June 2020

mation flow tracking, and hardware-based security approaches [46]. Because of the increasing availability of IoT devices in cloud services [10], IoT systems are often highly complex, requiring end-to-end security solutions that span cloud and connectivity layers, and support resource constrained IoT devices that often aren't powerful enough to support traditional security solutions. A comprehensive approach to IoT security is therefore required in order to protect communications, devices integrity, backend and cloud infrastructure [51].

In this work, we present a novel approach to provide comprehensive security to IoT devices. Our approach is based on a reconfigurable, hardware-based isolation and protection mechanism (IPM), first introduced in the work of [25], that operates as a dynamic separation unit between devices and network, far from potential software manipulation. The IPM analyzes communications for malicious activities and prevents damage to the system. The IPM is directly connected to a central server in the cloud where suspicious activities and logs from all IPM-protected IoT devices of network traffic are analyzed to improve their defense mechanism. The proposed IPM design supports a remote secure provisioning feature to allow run-time partial reconfiguration. The IPM achieves a 98.68% detection rate when evaluated against a Neptune DoS attack.

The work reported in this paper makes the following key contributions:

- A prototype IPM design and implementation as system on chip (SoC) in FPGA.
- A prototype design and implementation of centralized cloud server for life analysis of defense threat and defense update.
- A prototype IPM design runtime reconfiguration of defense mechanism.
- IPM performance evaluation against Neptune DoS attack.

The sections of this paper are organized as followed. In Section 2 we discuss related works relevant to our efforts. Section 3 introduces the primary components of the proposed Global Distributed Protection Mechanism. Section 4 describes the Isolation and Protection Module while Section 5 describes the central learning server architecture. A prototype IPM design and implementation and a prototype centralized learning server implementation details and evaluation are discussed in Section 6. Section 7 concludes the paper and discusses the direction of our future effort.

2. Related works

IoT devices and technologies are essentially embedded devices extended with internet communication capabilities. Security approaches are therefore based on traditional embedded system technologies including reference monitors, cryptography, dynamic information flow tracking (DIFT), intrusion detection solutions and compile-time code instrumentation.

Most reference monitor related solutions are based on executable code scanning and comparison to predefined models. The security subsystem operates in parallel with each processor in [30]. Hardware-assisted security monitors [34,42,59] are based on the concept of sensing deviation in program execution at run-time by comparing behavior against a static model for the purpose of detecting code modification attacks. Hash-based patterns for basic blocks comprising of fewer instructions have been used in [34]. Similarly, [59] proposed a security solution for a multi-core processor based real-time embedded systems that is based on running monitoring program on a dedicated core.

Cryptography algorithms such as the advanced encryption standard (AES) [38] and RSA [44] have been used in embedded systems to protect integrity. Various security mechanisms for the IoT devices have been discussed based on cryptography [21]. Though, resource and low power constraints can limit the practicability of some cryptographic approaches for IoT security.

Dynamic Information Flow Tracking (DIFT) has been effectively used to ensure protection against software-based attacks [18,47,50]. The idea is to mark non-trusted input sources and track the flow of information supplied to the program through these input sources. All tainted values, i.e. data values that flow though non-trusted inputs are tracked. If the tainted value leads to any untainted data value during program execution then an alert signal is generated reflecting an unauthorized access. Hardware-based security solutions such as [34,42,59] are not generic and require either dedicated hardware modules or specific modifications in the processor pipeline or cache architecture. Similarly, DIFT based hardware-assisted solutions such as Secure Program Execution [50] and Dynamic Tainting [18,47,50] are based on tagging data coming from untrusted sources and then tracking their usage as the application executes. These techniques require modifications in application's data, processor architecture, and memory layout, thus not practical for IoT systems that must fulfill low-cost and low-power requirements.

Intrusion detection techniques such as those in [4,43] requires implementation of TCP/IP stack and signature storage which is difficult to achieve with limited resource in microcontroller-based sensor nodes. Vector based classification methods [31] require storing of valid dataset on the system and then performing intrusion detection analysis which is again not feasible due to higher memory footprint and performance overhead. Different anomaly-based intrusion systems reliant on neural networks and compiler-based techniques for the detection of illegal memory accesses (IMA) have been presented in the literature. Neural networks, and machine leanring in general, have been used to detect anomalies in the program behavior by classifying system-call sequences [23]. Recent surveys [7,11] have presented an extensive overview of anomaly detection techniques and showed that the neural networks have been tested and implemented in the field of network systems largely[53]., for instance, has presented a comprehensive overview of intrusion detection techniques using computational intelligence methods such as fuzzy systems, evolutionary neural networks, artificial immune systems, artificial neural networks (ANN), and swarm intelligence. Additionally, in [13], neural networks are used to predict the flow of traffic in a given time window within a network and detect any intrusion in the system[14]. successfully deploys an autoencoder and deep learning solution for DDoS detection. The authors of [61] step away from neural networks and apply the C4.5 algorithm to generate a decision tree but also utilize a larger set of data beyond just basic packet information for detecting traffic-flooding network attacks. The main problem with existing machine learning based intrusion detection solutions is their complexity which limits their deployment on low-capability IoT systems. Our proposed system does not dismiss the advances of the mentioned works, but rather suggests an architecture in which these solutions may be more successful in the context of IoT devices.

Many software-based dynamic memory bug detection techniques have been proposed in the literature. These methods vary in implementation level, memory utilization, run-time overhead, types of bugs detected, probability to detect bugs, supported architectures and many other features. Referent-object based approaches are discussed in[3,28,45,60]. They work at source-code level by maintaining a separate table, using different data structures, to record bounds of each memory allocation. This table is then used to verify memory accesses by performing table lookups at run-time. These techniques differ in the implementation and handling of record tables. In [5,6] LibsafePlus resp. TIED have been implemented to handle static allocations and dynamic information such as stack size and heap allocations. These details are then used at run-time to detect any overflow. SAFECod [17] operates at source-code level and instruments loads and stores to prevent illegal memory accesses by using points to analysis and type-inference to find type-safe regions of the heap. Similarly, PAriCheck [60] computes bounds and assigns label to each fixed size. Pointer-based approaches [24,36,37] associate base and size metadata with every pointer and insert run-time checks manipulating metadata information during load/store operations of pointer values. CCured [37] com-

bines instrumentation with static analysis to insert run-time checks and removes redundant checks at compile time. Unlike prior pointer-based approaches that modify pointer representations and object layouts [24], SoftBound [36] records the bounds information in a disjoint metadata which is accessed via explicit table lookups on loads and stores of pointer values only. AddressSanitizer [48] verifies whether each allocated memory block is safe to access by creating shadow memory around stack and global objects to detect overflows.

Almost all the solutions previously listed are deployed and operate at the source code level with some privileged access to the operating system. This protection model plays a critical role as a security solution in today's (resource unconstrained) systems such as workstations and servers but cannot be used in IoT systems with restricted resource and energy. For instance, the work of [12] proposes an IoT Manager suitable for large-scale deployments of IoT sensor devices, yet unfortunately does not account for security at the endpoints (IoT devices) of the smart city system, likely due to the extra considerations introduced with devices with resource restrictions. The problem is further exacerbated with disappearing physical boundaries in companies or smart cities, which have their network increasingly distributed in several locations with remote access to servers and other devices [8]. The advent of IoT will allow remote access to small devices such as microscopes and other lab devices, most of which lack proper protection mechanisms since they were designed with price, efficiency, and power constraints in mind. Also, they have very limited capability to run complex protection tools such as antivirus and firewalls. Successively hijacking exposed IoT devices will give access to sensitive company and organization internal data, allow botnets to better hide and efficiently recruits new hosts. For instance, a botnet planted into a printer of a research organization will go undetected and allow the botmaster to have access to all documents, including important research results sent to that printer. Software-implemented protection mechanisms that rely on traffic data analytics to detect botnets become useless when the target has been hijacked. In addition, traffic analysis is a computationally expensive problem to solve with software implementations. It requires a large amount of resources (CPU time, memory) which degrades overall system performance. There are a few existing hardware-based solutions, Network Intrusion Detection Systems (NIDS), Cisco's 4200 series [41], IBM's Proventias GX5108C-V2 [41] for example, though they are very expensive and cumbersome to be used as protection in IoT system.

Our approach in this work is hardware based and operates far from the software level, which prevents complete system hijacking. The independent analysis of traffic data by the IPM takes place far from the reach of a malicious component that controls the system software, thus preventing damage to the system or stealing of sensitive date. Collected data from the IPM is used by a central server in the cloud to implement live learning and update the defense mechanism of IPM connected to it. In the next section, we first explain our overall concept before presenting the internal architectures and operational modes of our defense mechanism.

3. Global distributed protection mechanism

Fig. 1 illustrates the global mechanism and infrastructure proposed in this work for protection of IoT devices. The main defense component is the isolation and protection module (IPM) used to bridge all communications between the IoT device and the internet. The goal of the IPM, explained in detail in Section 4, is to decouple the protection mechanism from computing at local IoT nodes, thus allowing device protection to continue to work, even in case of device hijacking by a malicious piece of software. The IPM is a hardware module that can be used in three different configurations. The first is a stand-alone, where the IPM is connected separately between the device and the network, similar to an off-the-shelf WiFi key that connects a device such as Arduino or Raspberry Pi to a wireless network. In the second configuration, the IPM is an IC module used as part of the printed circuit board that forms the IoT hardware. The third configuration is similar to the second one. However, the IPM is a soft IP module integrated into a system on chip that powers the IoT device. In all three cases, all connections between the device and the network flow through the IPM where they are analyzed for malicious activities to prevent damage on the IoT-controlled system. Our global distributed protection infrastructure features a server in the cloud that collects data on suspicious activities from all connected IoT as they unfold in order to learn the behavior of potential attackers. This knowledge is the used to update the IPM defense strategy by means of partial reconfiguration.

To elaborate on the policies and rules, we need to understand the behavior of potential attackers, preferably live as they unfold and use the knowledge to update the defense mechanism within the IPM of each device connected to the internet. For instance, once an IoT node in our cluster is hijacked and becomes part of a botnet, all activities in that botnet will be reflected on that node and the hardware isolation and protection mechanism will discretely collect all communication data between the node and the Botmaster. All data collected by cluster nodes will be analyzed to understand the botnet's activities and anticipate its next steps. The approach we propose is similar to the Blue Coat cloud-based method used in [8] for botnet detection and defense. While our method automatically collects data from live botnets, Blue Coat requires users to submit URLs and webpages of known antivirus and other malicious applications to their WebPulse cloud service. The cloud then performs threat analysis and uses the results to enhance defense strategies of single nodes. There are two main problems with this approach. First, identifying malicious URLs and suspicious activities requires a level of expertise that most computer users do not have, let alone the majority of consumers purchasing off-the-shelf IoT solutions. Second, the inference of knowledge from URLs and data collected from users is done off-line and does not consider any live activities of botnets. Our approach solves the first issue with the Blue Coat cloud solution by automatically collecting data from communication between botnets and infected devices for processing without raising any suspicion on the Botmaster's side. Because the IPM is implemented in hardware, analysis within the IPM happens in real-time and valuable information is sent to the inference server, which can be used to detect patterns of activities in botnets. Knowledge gained from the inference server is then reused to adapt the defense mechanism within the IPM.

4. Isolation and protection module

Fig. 2 illustrates our proposed Isolation and Protection Module (IPM) structure. As discussed in the previous section, the IPM main policy is to monitor the network traffic between the IoT device and the network and only allow legitimate traffic to pass. To implement this, the IPM relies on a *packet classification* mechanism and a *runtime reconfiguration* mechanism.

Packet Classification The IPM's packet classification scheme performs two complementary traffic analysis tasks as illustrated in 2. The first task is a signature-based traffic inspection that uses a pattern matching algorithm to mine for known blacklisted domains (IPs), malware signatures, and blatant TCP flag violations. In this stage, packets found to possess any of the signatures under monitoring are discarded. Generally, this packet inspection model doesn't perform well for 0-day (unknown) attack signatures. To alleviate this deficit, the remaining packets are then forwarded to a scoring function which is trained to recognize packets attributes which are out of the expected range of the network behavior (profile-based detection). The scoring function gives scores to each packet's attributes and compares to a threshold value. If the packet's aggregated score (summation of all attributes scores) is less than the threshold, the packet is discarded. The remaining packets are considered to be more likely than not legitimate traffic and are then forwarded to the IoT device as shown by Fig. 3. The decision mailbox logs each packet processing transaction (packet and corresponding [at-

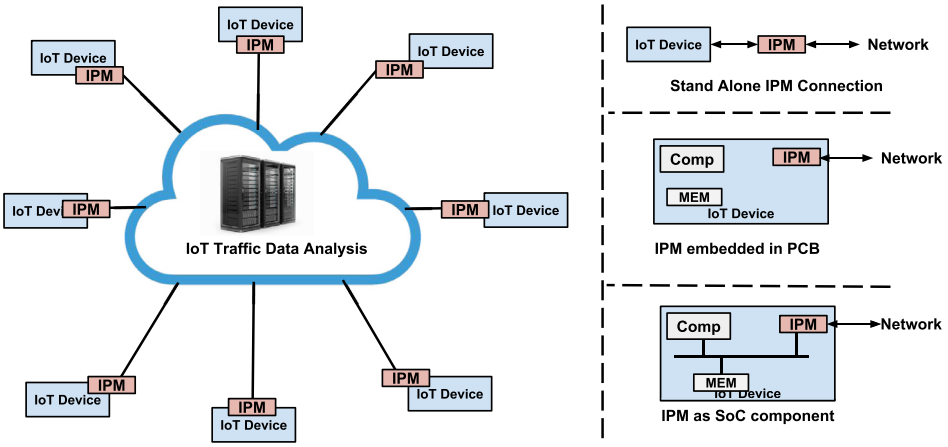


Fig. 1. Global Data Protection Mechanism.

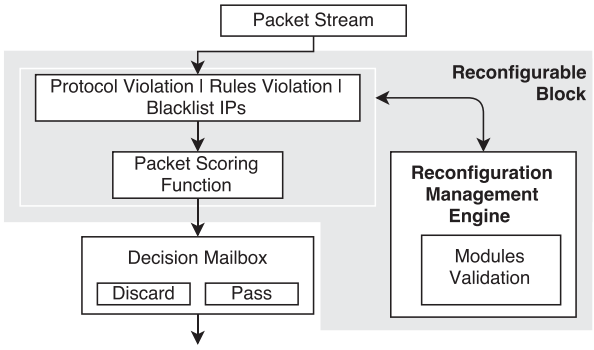


Fig. 2. Proposed IPM Structure.

tributes, scores] pair). This log is then sent out to a centralized cloud server (introduced in Section 3) for further analysis.

Runtime Reconfiguration

A key design goal of the IPM is the ability to allow runtime reconfiguration of 1) blacklist signatures with the most up-to-date signatures as they become available, and 2) the scoring function with a more efficient scoring function through remote secure provisioning. To perform updates, the IPM periodically initiates a secure communication channel to a centralized cloud server and downloads the updated signatures pool copy or a new scoring function bitstream. A delegate partial reconfiguration function on the IPM is then called to initiate the runtime partial reconfiguration process. To avoid the man-in-the-middle attack situation, the reconfiguration management engine implements a module validation scheme which authenticates and verifies the proper identity of the centralized server. This feature can also be extended to allow a "secure provisioning" mechanism where only signed binaries can be downloaded.

5. Cloud-Based online learning

In order to provide effective analysis of network traffic, a system must be capable of adapting to new and unknown threats while continuing to improve detection methods for previously encountered threats.

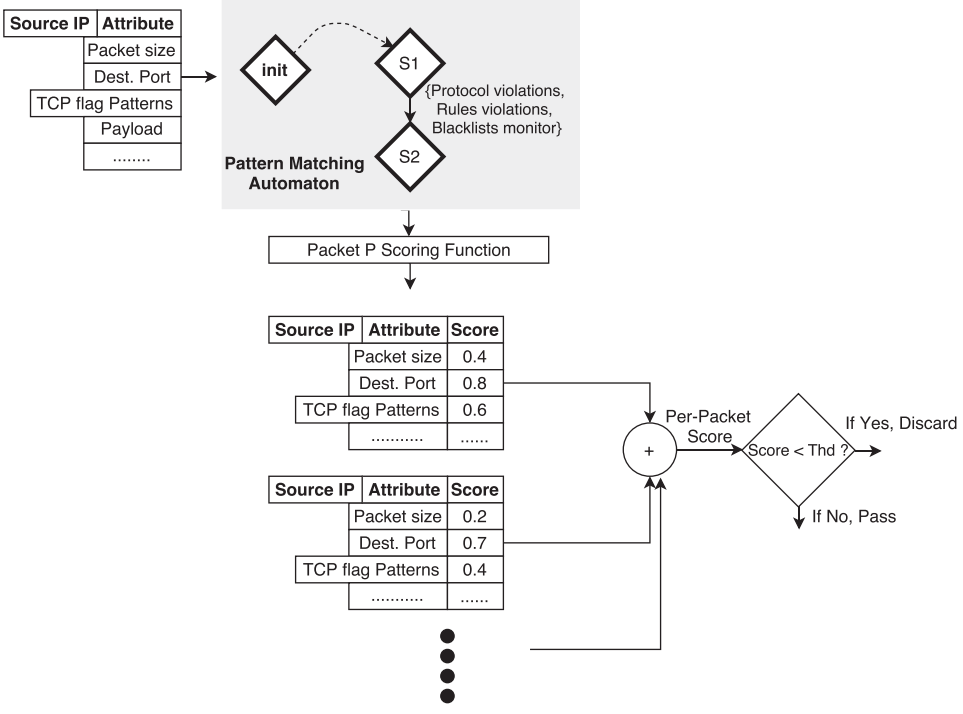


Fig. 3. Packets Classification Process.

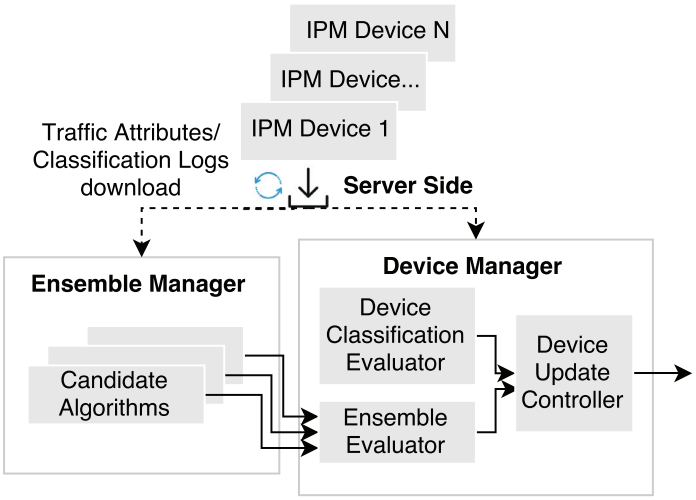


Fig. 4. Proposed Cloud Structure.

Our solution leverages a central cloud-based authority to broaden the scope of traffic analysis beyond that of a singular IoT device to that of many, relieves resource constrained devices of computationally expensive performance evaluation methods, and makes available numerous configurations for device packet classification. The proposed cloud structure is illustrated in Fig. 4. The inclusion of the online learning enables more flexibility to the end-user resource constrained IOT device. Additionally, the cost of the implemented IOT device protection can be minimized through proper partitioning of online-learning and local decision making.

By periodically collecting the decision mailbox logs from each device, the server can construct network traffic profiles and analyze close to real-time data which consists of logged packets with corresponding attribute/score pairs. The device's mailbox determines the acceptance or denial of each packet based on a threshold controlled by the server. The analysis of the traffic profiles produces a set of resulting packet classification scores and acceptance decisions that are used as a basis of comparison to the products of the device's classification, and the threshold may be adjusted according to the performance of the current report window.

Without significant concern for resource limitations in the cloud service, a number of configurations for packet classification algorithms, in parallel, will be trained and evaluated over the received network profiles. This set of algorithms is referred to as the Candidate Algorithms Ensemble (CAE). These configurations can include various candidate packet classification algorithms which may be transferred to IoT devices when the traffic profiles and corresponding decisions of packet acceptance indicate poor performance after repeated threshold updates. With many established learning algorithms proven in network packet classification and threat detection, the algorithms chosen are not of current concern of this paper. Rather, we intend to propose an architecture capturing the constraints of IoT devices while providing the processing performance required for effective detection and mitigation. The network profiles are used as inputs to the CAE to obtain the classification results based on a voting-scheme. The ensemble of algorithms will be polled for their results and the mode of each attribute score set will be considered the consensus, the set of most popular results is used to adjust the threshold of the device. The ensemble's decisions are also logged in order to calculate statistics on each algorithm's performance to provide an evaluation metric for device IPM configuration updates.

6. System design and implementation

In this section, we discuss the design and the implementation of our proposed security architecture. The section starts with the design, implementation, and the evaluation of a prototype IPM. This is followed by a discussion of the implementation and the evaluation of a prototype centralized learning architecture.

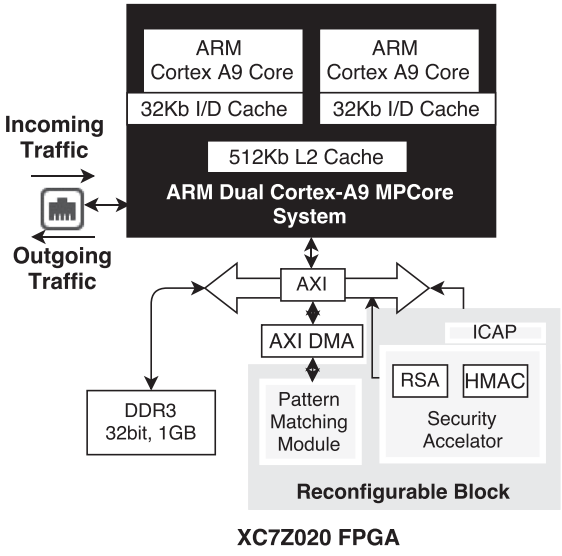


Fig. 5. IPM Implementation. The IPM was implemented as a SoC on Xilinx's XC7Z020 FPGA.

6.1. IPM SoC Implementation

To evaluate the viability of our proposed IPM architecture, we designed and implemented it as SoC on FPGA because of FPGA's ability to support runtime partial reconfiguration. We used Xilinx's ZC702 evaluation board which embeds the Xilinx's all programmable Zynq XC7Z020-1CLG484C system-on-FPGA [57] as the IPM implementation host. The evaluation board also features Ethernet PHY RGMII interfaces, 1GB DDR3 component memory (four 256 Mb x 8 SDRAMs), and various configuration interfaces (JTAG, USB, etc.) [57].

We designed the IPM system application as software/hardware co-design. Fig. 5 shows our design architecture. We implemented the Ethernet MAC layer, a custom protocol stack module, the pattern matching logic, and the security accelerators on the programmable logic. The scoring function, the client-server communication application, and the reconfiguration manager function were partitioned to run on the Zynq ARM processor.

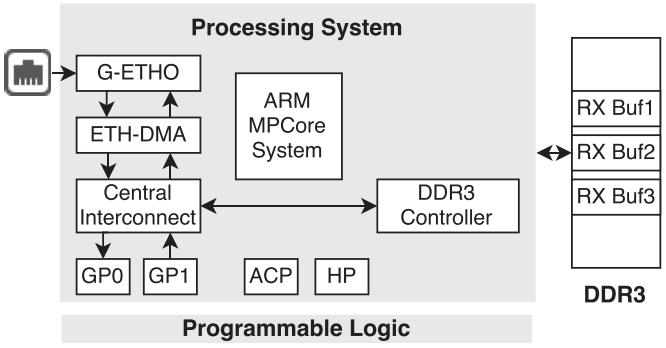


Fig. 6. The Zynq PS packets reception process.

Table 1
XC7Z020 Resources Utilization.

Resources	Utilization Percentile
Slice LUTs	30,813 out of 53,200 (57.92%)
Slice Registers	43,208 out of 106,400 (40.61%)
BRAMs	26 out of 140 (18.57%)
Dynamic Power	2.004 W

The Ethernet PHY Interface on the ZC702 is directly tied to the Zynq XC7Z020 processing system (Zynq PS) MAC [58]. This means that packets are received through the Zynq PS and can therefore be diverted to the programmable logic for payload inspection. The Zynq PS packets reception process goes as followed. As Fig. 6 illustrates, the packets are received on the Ethernet PHY interface FIFO (G-ETH0) are then read by the Ethernet DMA controller (ETH-DMA). The Ethernet DMA uses separate TX and RX lists of buffer descriptors, with each descriptor describing a buffer area in system memory. The ETH-DMA writes the received packets to pre-allocated buffer descriptors in system memory. These RX buffer descriptors are kept into the RX receive queue. The ETH-DMA RX buffer queue register points to these buffer descriptors to continuously copy the packets to the memory.

In our implementation, we partitioned the pattern matching logic to run on the Zynq programmable logic (PL). To support the client-server communication functionality, we also partitioned a 256-bit RSA and a 256-bit HMAC-SHA1 IP cores from opencores.org, the open-source repository for hardware implementations, to run on the Zynq PL.

To implement the core functionality of our software partition, we used Xilinx's Light Weight IP (lwIP) which is an API (application programming interface) to a light open source implementation of the TCP/IP protocol suite [55]. The lwIP library supports two interfaces: raw API and BSD style Sockets API [54]. In our application, we chose to use the RAW API because of its callback style interface. Since Xilinx's lwIP supports threaded application, a microkernel is needed to support threads scheduling and synchronization. In our implementation, we used the FreeRTOS Xilkernel as the Xilinx's application note Xapp1026 in [54] recommends.

Table 1 shows the FPGA's resources utilization. A large usage of LUTs and registers is due to the use of large key widths for both the RSA and the HMAC. The chosen algorithms can be optimized for IOT resource restrictions with alternate asymmetric cryptography functions such as elliptic curve cryptography (ECC) implementations, though the on-hand implementations effectively served to demonstrate the client-server authentication.

6.2. IPM Client application

We use the lwIP library to create a basic TCP client for securely connecting to the cloud learning service. The client application handles all server communication which can be organized into three primary tasks; 1) gathering public credentials of the server, 2) initializing a new secure session with an authentication handshake, and 3) exchanging encrypted data, i.e. device traffic and classification logs with returned updates to the local classification configuration for packet monitoring. This is demonstrated in Fig. 7. The importance of maintaining the integrity of the connection between IoT end points and the central learning server is of key importance. The steps in the following section proposes a possible scheme for meeting this requirement, one which resembles the Transport Layer Security (TLS) protocol with considerable simplifications [2]. The steps of these three tasks may be substituted for other cryptographic methods, but with the device resource constraints in mind. Our implementation is acknowledged to contain possible vulnerabilities but remains sufficient for our proof of concept system.

Key Exchange Considering the availability of the RSA and HMAC IP cores, we utilize RSA key set to simplify the network interactions. In order to acquire the server's public key, a simple 'hello' message is generated to contain the client's public RSA key. The server is expected to respond with a similar 'hello' message containing its public key, which is subsequently stored in DDR3 memory for use by the RSA IP core. With knowledge of each other's public credentials, the client may continue to perform an authentication handshake with the server. The key exchange process is necessary only as required by the end-system, for a sophisticated system would schedule this with systematic key distributions while our concept only requires a single key exchange.

Authentication Handshake A server connection is initialized by the client who first generates a secret key for AES 256-bit encryption, used to encrypt a 'Connect' message for the server [38]. We utilized a light AES library in our C software application. In order to share the secret key, it is encrypted using the server's public RSA key, done with a write to the RSA core's input memory index, followed by a read out of the encrypted cipher. Finally, we package the encrypted key and encrypted message along with an RSA signature for the server to verify. The second step involves the server verifying the transmission integrity and decrypting the key and message. Upon verifying, the server will generate the public parameters of Diffie-Hellman (DH) key exchange to be shared with the device, with which both server and client may generate the secret key used for message encryption. The server encrypts a response message with the generated secret key and also provides a hashed message authentication code (HMAC) signature, along with a client public key encrypted secret for hashing, that allows the client to verify the server's response message. Upon receiving the packaged public parameters for DH, an encrypted response message, HMAC signature, and encrypted HMAC key, the client verifies the signature, with our HMAC core, and subsequently decrypts and verifies the response message. The handshake is now complete upon the generation of secret keys by both parties, establishing a session. Though relatively secure, the authors must disclose that this authentication scheme was implemented as a means to demonstration. The work of [52] provides a survey of IoT authentication mechanisms and the resource and security considerations associated with each. Future work will focus on an in-depth analysis of the handshake as it provides vital infrastructure for security in the IoT devices and network as a whole.

Traffic Processing Assuming verification of both parties was successful, the secret keys generated via DH are used to encrypt, with AES 256, all proceeding transmissions of the session. Using the above handshake method, we can ensure secure communications between the client IoT devices and the learning server. The client will now periodically read from traffic logs with the corresponding classification results, with the number of reads and interval being determined how often the client wishes to share traffic data. Upon receiving the results, the server will process the results and send updates to improve the classification algorithm locally on the device. The server will now send an updated threshold upon every transmission of a device's traffic profile. The client device will only share its network profile once per session considering our current reporting window of 10 minutes. Thus, our session expires after the server response, requiring the authentication process to be done

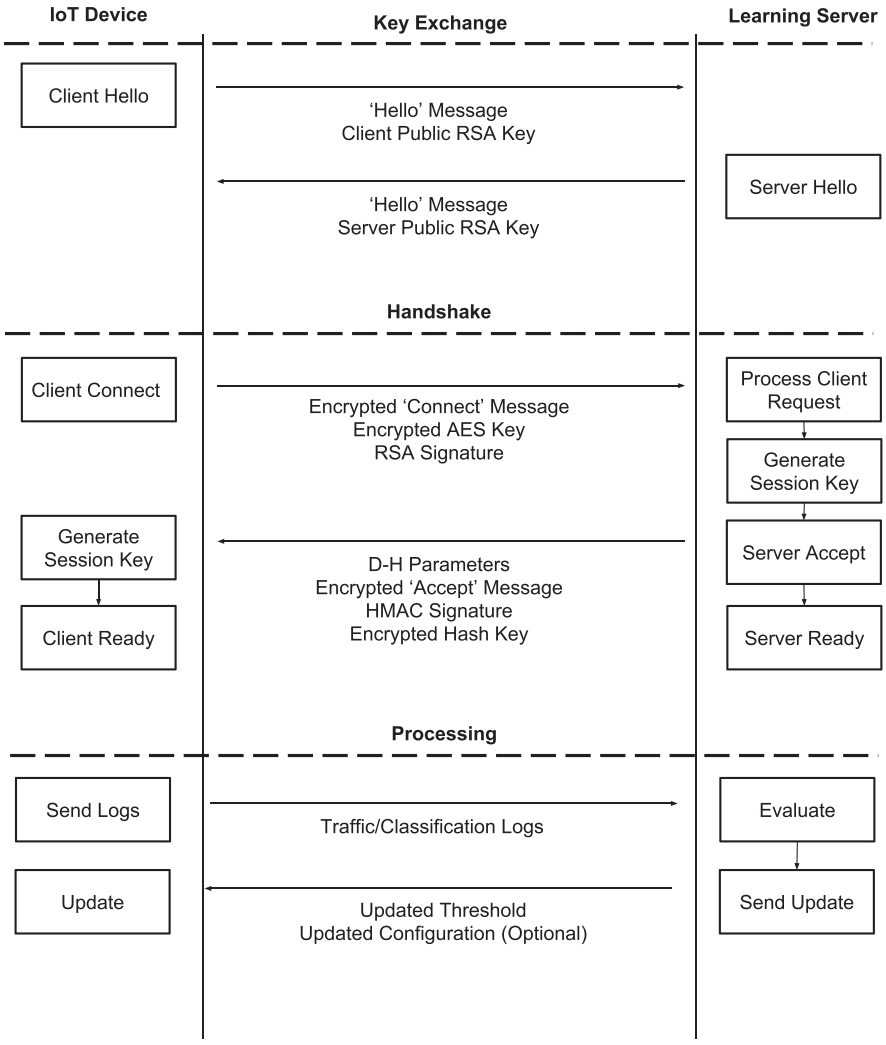


Fig. 7. Network Interactions.

each time. In the case of bad performance of a classifier algorithm as determined by the server, the server's response will contain a configuration update from which the programming logic may reconfigure.

6.2.1. Packets classification implementation

We implemented a packet classification scheme illustrated in Fig. 3. The classification process is performed in two complementary stages: 1) Signature-based inspection and 2) Profile-based classification.

Signature-based Inspection

A pattern matching problem can be defined as finding a pattern P of length M in a text T of length N where M<N. Pattern matching algorithms are classified depending on their direction of search (forward matching vs. backward matching) and the number of patterns they can detect within text T (single pattern detection vs. multiple pattern detection). Forward matching consists in moving P against T from left to right whereas backward matching does the opposite [9]. In our case, we used a multiple pattern detecting forward-matching algorithm in which each pattern represents a known signature or packet feature. Fig. 8 illustrates a functional representation of the pattern matching mechanism.

A preprocessing module maps incoming 8-bit characters into 5-bit characters to take care of case insensitivity; a comparator module decodes incoming ASCII characters and sends them to the automaton. The automaton moves from one state to another according to previously received characters and the currently received character using a common prefix search method to reduce the number of states in the automaton. A simple example illustrates the idea: Suppose a Payload P contains four

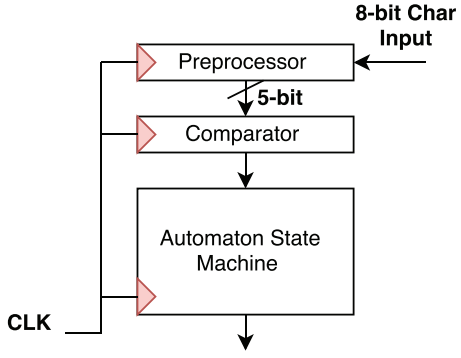


Fig. 8. A simplified block diagram of the pattern matching module.

words party, paris, beef, before. Using a common prefix search method, the algorithm builds up these four words using a forward matching approach as Fig. 9 indicates.

This common prefix search method is very efficient because it does not build an automaton for each pattern we are detecting; this dramatically reduces the number of flip-flops needed for implementation.

Profile-based Inspection During the profile-based inspection stage, incoming network traffic is compared to "learned" network profile to check potential anomalies. Identifying a feature set likely to reveal the

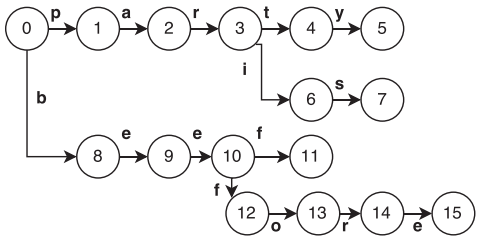


Fig. 9. Example of the common prefix search method.

presence of a particular attack remains a key challenge to the design of efficient network classification solutions. Choosing features should solely rely on features' significance and their capabilities in terms of revealing the presence of the targeted attack [49]. Fortunately for us, Kayacik and al, did extensive work on DoS activities, the test application we chose for IPM evaluation, and were able to point out 27+ network traffic features that, if carefully analyzed, would most likely reveal the presence of a DDoS activity [29]. A detailed explanation on the significance of these features in the context of DDoS detection is given in [29].

Scoring Function In this application example, we employed a simplistic version of the support vector machine (SVM) algorithm. The algorithm assigns numerical values (scores) to the packet's attributes (or features vectors) and compare the vectors' average score to the reference score and makes a decision on the vectors (legitimate traffic vector OR DDoS traffic vector). Before we describe how the scoring is done, let's formalize it first:

Let $\vec{S} = \vec{A} + \vec{B}$ be a training set of m number of samples where $\vec{A}$ is the subset of n "legitimate" traffic samples and $\vec{B}$ is the subset of p ($p = m - n$) "DoS" traffic samples. Let $\vec{W}$ be the average value of the feature in a normal traffic and $\vec{Y}$ be the average value of the same feature in a DoS traffic. Let N be the number of packets attributes our system is collecting.

We calculate the average value $Wi_{(k)}$ of feature k in all legitimate traffic samples

$$Wi_{(k)} = \frac{1}{n}\sum_{i=1}^{n} Ai_{(k)} \tag{1}$$

and calculate the average value $Yi_{(k)}$ of the same feature in all DoS traffic samples.

$$Yi_{(k)} = \frac{1}{p}\sum_{i=1}^{p} Bi_{(k)} \tag{2}$$

Let $\vec{Z}$ be a reference "plane" (threshold) that separates $\vec{W}$ and $\vec{Y}$ classes and leaves the maximum distance between itself and the classes. For each feature k, the reference plane can be calculated as:

$$Zi_{(k)} = \frac{1}{2}(Wi_{(k)} + Yi_{(k)}) \tag{3}$$

During the training phase, our scoring function learns parameters $Wi_{(k)}$, $Yi_{(k)}$, and $Zi_{(k)}$ of each feature using Eqs. 1, 2, and 3. Let $\vec{X}$ be an N-features traffic packet we are looking to classify. For each feature value $X_{(k)}$ of $\vec{X}$, if:

- $X_{(k)} > Z_{(k)} \wedge W_{(k)} > Z_{(k)}$,

then the corresponding feature $X(k)$ is a normal traffic feature (assign a -1 score). Otherwise, if :

- $X_{(k)} < Z_{(k)} \wedge W_{(k)} < Z_{(k)}$,

then the corresponding feature $X(k)$ is a DoS traffic feature (assign a +1 score).

The more a traffic vector scores towards -N, the likelihood that the vector belongs to normal traffic increases. The more it scores towards +N, the likelihood that the vector belongs to a DoS traffic increases. The algorithm's pseudocode is shown below.

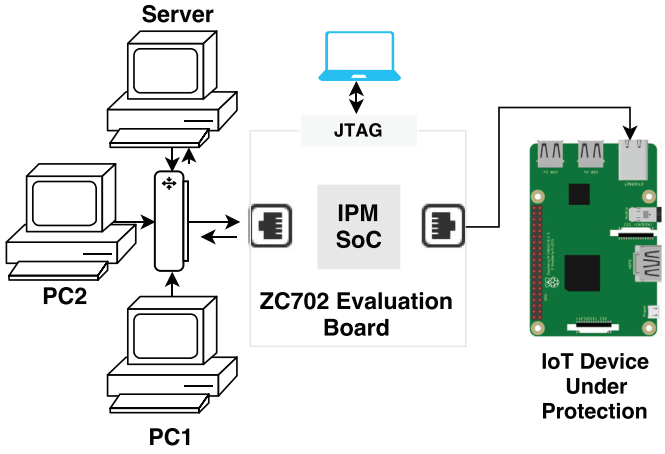


Fig. 10. Test Setup. Local network created among an IPM running on ZC702, 2 PCs, desktop Python server, and a Raspberry Pi3. Custom network traffic is generated using Nmap and Netcat tools from PC1 and PC2.

Scoring Function Training To train our scoring function, we employed the KDD benchmark DoS dataset. We divided the dataset into training, testing, and validation sets. A typical ratio of training-test-validation set of 80%-10%-10% was used. For sets balancing purposes, we used equal number of normal traffic and DoS traffic samples for the test and validation sets. The validation runs results are shown in Table 2. The results include true and false positives, true and false negatives, and the time it took to make a decision on each packet. Though our resulting detection time indicates the lack of a true real-time packet classification, we believe future work can alleviate this delay. Our implementation on the ZC702 captures packets from the ethernet port that has a direct connection to the Zynq processing system and it is hypothesized that a significant classification speedup can be realized with attaching our Pattern Matching Module to a programmable logic connected ethernet port. This would remove a dependency on the AXI bus to retrieve packets as well as any PS introduced overhead.

Scoring Function Detection Evaluation

To evaluate the IPM scoring function classification accuracy, we setup a local network between two computers, the IPM running on Xilinx's ZC702 evaluation board, a desktop Python server, and a Raspberry Pi3 as the IoT device under protection. Fig. 10 illustrates the setup. On this local network, we simulated a light traffic DoS using both *Nmap* and *Netcat* tools. *Nmap* provides a flexible way to design a custom traffic by customizing packets payloads and transfer rates [33]. We used *Netcat* to establish TCP connections between the machines running *Nmap* and the Xilinx ZC702 evaluation board hosting the IPM. The created traffic was a Neptune DoS attack where the source IPs purposely fail to complete a TCP handshake with the IPM by refusing to send ACK packet [27]. We chose this application because it is one of the simplest, and yet still popular DoS attacks [27]. Monitoring the 10 Neptune DoS-traffic revealing features, described in detail in [29], the IPM scoring function performance is shown in Table 3.

Fig. 11 shows the Receiver-Operating Characteristics (ROC) graph for the IPM scoring function. In a ROC graph, *TP rate* (the ratio of positives correctly classified over total positives) is plotted as a function of *FP rate* (the ratio of negatives incorrectly classified over total negatives). Informally, a ROC graph depicts tradeoffs between the benefits (true positives) and the costs (false positives) [20]. The upper left corner of a ROC graph, (0,1), provides the best classification performance with no false positives [35]. The IPM function places at (0.009, 0.992).

Signature-based Detection Performance

To test the signature-based portion, we employed the same tools described in section to create a non-encrypted traffic to the IPM with custom payload. The payload was the EICAR (European Institute for Com-

Table 2
5-Folds Validation Results.

DDoS Attack Type	Legitimate Traffic Detection Rate	DDoS Traffic Detection Rate	FP	TN	FN	TP	Time
Neptune	98.68%	99.34%	68.6	5988.4	40	6017	0.83s
Smurf	87.18%	100.00%	8.2	55.8	0	64	2.05s
Back	100.00%	96.66%	0	96	3.2	92.8	0.79s
Teardrop	100.00%	99.34%	0	91	1	90.4	0.38s
Pod	98.68%	100.00%	1	19.4	0	20	0.38s
DDoS (all)	99.99%	95.62%	0	6332	276.6	6055.4	7.16s

Table 3
Detection Performance on DDOS Traffic.

DDoS Attack Type	Legitimate Traffic Detection Rate	DDoS Traffic Detection Rate	FP	TN	FN	TP	Time
Neptune	98.68%	99.39%	80	5977	37	6020	0.83s

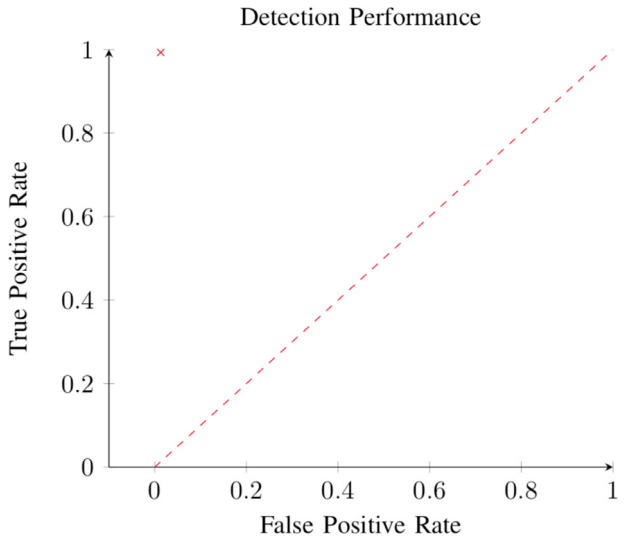


Fig. 11. The Receiver-Operating Characteristics (ROC) graph for the IPM scoring function.

puter Anti-Virus Research) malware signature. The malware.txt file can be found on [19]. Beside custom payload, we also tested TCP flags violations (SYN and FIN both set for example). The pattern matching logic detected both successfully. The pattern matching logic has a 100% detection accuracy when it has apriori knowledge of the payload and the traffic is non-encrypted. We leave the problem of encrypted traffic to future development.

6.2.2. *Runtime partial reconfiguration*

We scheduled the server-IPM communication to occur at periodic 10 mins time intervals. Ideally, it is during this server-IPM communication that the server schedules the reconfiguration process and pushes a signed bitstream update to the IPM node. In the current version of our implementation, we mimicked server-authorized reconfiguration by re-synthesizing pattern matching logic with new payload signatures to form partial bitstreams on a local host machine. The used signatures originate from Deepend Research's blacklist library for compromised URLs [16]. Table 4 shows a few of the signatures we used.

Using the partial reconfiguration flow in [56], the newly synthesized pattern matching logic bitstream is loaded from the host through the JTAG and is swapped on the fly in the place of the old pattern matching logic while the static design continues to operate. The overhead associated with reconfiguration is calculated as the ratio between the partial bitstream size over the JTAG bandwidth (66 Mb/s) [56]. For our 4.1Kb partial bitstream, the total reconfiguration overhead was 64s. Running

Algorithm 1 Packets Classification.

```
1: procedure CLASSIFY( )
2:     w ← weightvector
3:     for i = 0 to samples do
4:         r ← 0
5:         for j = 0 to features_selected dogoto loop
6:         end for
7:             goto decision
8:     end for
9:     loop:
10:    if (testing_matrix(i, j) > w(j)) then
11:        if (legitimate_mean(j) > w(j)) then return r–
12:        else return r + +
13:        end if
14:    end if
15:    if (testing_matrix(i, j) < w(j)) then
16:        if (legitimate_mean(j) < w(j)) then return r–
17:        else return r + +
18:        end if
19:    end if
20:    decision:
21:    if r > 0 then return DoS
22:    end if
23:    if r < 0 then return Legitimate
24:    else return Undetermined
25:    end if
26: end procedure
```

Table 4
Partial Bitstreams Signatures.

Malware Family	Pattern/URI
FakeAV Privacy Center	/css/new-mobile.css
GameVance Adware	/aj/updtah.php
Ardamax keylogger	250-smtp.mail.yahoo.com
Sweet Orange EK	/ip/ch/investor.php?setup=20
AdWare Kraddare.IL	/bv/config.php

the same test scenario described in 6.2.1 with the newly installed pattern matching logic, the IPM successfully detected all these signatures.

6.3. *Cloud server implementation*

We implemented our server in the Python language to leverage the plethora of libraries to cover the requirements of the proposed central learning hub. Another benefit of Python is ease of creating a TCP server with the built-in low-level networking interface module, 'socket.' In order to complement the IoT devices, the server holds an RSA key set and

Table 5
LIBSVM Detection Performance.

DDoS Type	Legitimate Traffic Detection Rate (IPM)	Legitimate Traffic Detection Rate (LibSVM)	DDoS Traffic Detection Rate (IPM)	DDoS Traffic Detection Rate (LibSVM)	Time (IPM)	Time (LibSVM)
Neptune	98.68%	99.99	99.39%	99.95	0.83s	3.93s

shares the public credentials upon requests, processes incoming requests for secure connections, evaluates traffic from IoT nodes, and serves updates when deemed appropriate. The server builds a pool of devices for which it stores public credentials and generated secrets for ongoing sessions as more devices connect to the central server, which is required for managing numerous secure connections. The server also sets a threshold for each device to fine tune the scoring algorithms at each device.

The authentication process is explained in detail in 6.2 and is accomplished with our Python server with the use of a cryptography library, PyCrypto [1]. All communications after the handshake are encrypted and our server decrypts with the use of the same library. The most crucial tasks of the server are the evaluation and update of the client IoT nodes.

Server Classification Evaluation

A major benefit of the cloud-based learning is the lack of constraints on the server resources. With this in mind, it is our intent to maximize the ability for learning, so we propose an ensemble of evaluation algorithms. Our current implementation handles a single configuration with the SVM as the main scoring algorithm. We used LibSVM library to implement the SVM algorithm and repeat the same test scenario in 6.2.1. Our proposed system assigns the central server the task of analyzing traffic logs from the IPM that are shared periodically. With our server leveraging the ease of Python, in our tests the IPM echoes each packet sent to it back to the server. The server logs the traffic trace for 10 minutes and then proceeds to classification, mimicking a periodic upload from IPM devices. The traffic log sharing scheme is irrelevant in this case, though, as we are only testing the server's classification and only need it to serve the purposes of demonstrating a proof-of-concept. The server's LibSVM performance is shown in Table 5.

Client Update The server manages a threshold value within the context of a device. This threshold is first updated by the server upon the first encounter of a newly connected IoT device, for the assumption is made that upon a first configuration of a device packet classifier, a threshold will be provided. For our implementation with the SVM algorithm in both the device and the cloud, the classification performance was consistent such that a threshold update was never required from the server. The second result output from the server's analysis of a client's network traffic is the decision to update the configuration of the classification algorithm at the client. Though our implementation currently lacks this functionality, our solution calls for a bitstream representation of each algorithm being evaluated at the server, that may be downloaded upon the need of replacing a poorly performing classifier.

7. Conclusion

In this work, we explored the use of a reconfigurable hardware-based isolation and protection mechanism (IPM) that operates as dynamic separation unit between devices and network for securing IoT devices. We discussed the use of a central cloud-based authority as an extension of the IPM intelligence to broaden the scope of traffic analysis beyond that of a singular IoT devices. We showed how employing a cloud server in this regard allows us to decouple the task of packet classification (and score computation) from that of updating the signatures pool to allow them to be done in parallel at different time periods.

This work also presented a prototype implementation of the IPM as a SoC on FPGA and a prototype implementation of a centralized cloud server. The IPM was evaluated in terms of size, Neptune DoS attack detection, and runtime reconfiguration overhead. Preliminary tests results show promise for securing IoT devices. Our future efforts will focus on testing our implementation at large scale at multiple points of a live network for scalability purposes. This includes an increase in the number of IoT devices connected to the central server as well as an increase in candidate algorithms on the server.

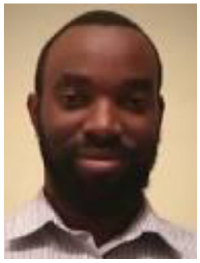
Declaration of Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Pycrypto - the python cryptography toolkit.
[2] Rfc 6101 - the secure sockets layer (ssl) protocol version 3.0.
[3] P. Akritidis, M. Costa, M. Castro, S. Hand, Baggy bounds checking: an efficient and backwards-compatible defense against out-of-bounds errors., in: USENIX Security Symp., 2009, pp. 51–66.
[4] S.O. Amin, M.S. Siddiqui, C.S. Hong, S. Lee, Rides: robust intrusion detection system for IP-based ubiquitous sensor networks, Sensors 9 (5) (2009) 3447–3468.
[5] K. Avijit, P. Gupta, Binary rewriting and call interception for efficient runtime protection against buffer overflows, Software: Practice and Experience 36 (9) (2006) 971–998.
[6] K. Avijit, P. Gupta, D. Gupta, Tied, libsafeplus: Tools for runtime buffer overflow protection., in: USENIX Security Symposium, 2004, pp. 45–56.
[7] M. Bhuyan, D. Bhattacharyya, J. Kalita, Network anomaly detection: methods, systems and tools, Communications Surveys Tutorials, IEEE 16 (1) (2014) 303–336.
[8] BlueCoat, Symantec cloud-delivered web security services, 2013.
[9] C. Bobda, T. Lehmann, Efficient Building of Word Recognizer in Fpgas for Term-document Matrices Construction, in: R. Hartenstein, H. Grnbacher (Eds.), Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing, Lecture Notes in Computer Science, 1896, Springer Berlin Heidelberg, 2000, pp. 759–768, doi:10.1007/3-540-44614-1_80.
[10] A. Botta, W. de Donato, V. Persico, A. Pescap, On the integration of cloud computing and internet of things, in: Proceedings of the 2014 International Conference on Future Internet of Things and Cloud, 2014, pp. 23–30, doi:10.1109/FiCloud.2014.14.
[11] I. Butun, S. Morgera, R. Sankar, A survey of intrusion detection systems in wireless sensor networks, Communications Surveys Tutorials, IEEE 16 (1) (2014) 266–282.
[12] L. Calderoni, A. Magnani, D. Maio, Iot manager: an open-source iot framework for smart cities, J. Syst. Archit. 98 (2019) 413–423, doi:10.1016/j.sysarc.2019.04.003.
[13] C. Callegari, S. Giordano, M. Pagano, Neural network based anomaly detection, in: Proceedings of the Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on, 2014, pp. 310–314, doi:10.1109/CAMAD.2014.7033256.
[14] F.O. Catak, A.F. Mustacoglu, Distributed denial of service attack detection using autoencoder and deep neural networks., Journal of Intelligent & Fuzzy Systems 37 (3) (2019) 3969–3979.
[15] P. David, T. Scott, The internet of things:an overview, 2015.
[16] DeepEnd, Library of malware traffic patterns, february 2015[Online; accessed September 2006].
[17] D. Dhurjati, S. Kowshik, V. Adve, SAFECode: Enforcing alias analysis for weakly typed languages, in: Proceedings of the 27th ACM SIGPLAN Conference on Programm. Lang. Design and Imp., ACM, New York, NY, USA, 2006, pp. 144–157, doi:10.1145/1133981.1133999.
[18] I. Doudalis, J. Clause, G. Venkataramani, M. Prvulovic, A. Orso, Effective and efficient memory protection using dynamic tainting, Computers, IEEE Trans. on 61 (1) (2012) 87–100.
[19] EICAR, Eicar intended use, 2012, [Online; accessed 16-February-2015].
[20] T. Fawcett, An introduction to roc analysis, Pattern Recogn. Lett. 27 (8) (2006) 861–874, doi:10.1016/j.patrec.2005.10.010.
[21] J. Granjal, E. Monteiro, J. Sa Silva, Security for the internet of things: a survey of existing protocols and open research issues, Communications Surveys Tutorials, IEEE 17 (3) (2015) 1294–1312, doi:10.1109/COMST.2015.2388550.
[22] C.M.W. Group, Security guidance for early adopters of the internet of things (IoT), 2015.
[23] S.-J. Han, S.-B. Cho, Evolutionary neural networks for anomaly detection based on the behavior of a program, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 36 (3) (2005) 559–570.
[24] N. Hasabnis, A. Misra, R. Sekar, Light-weight bounds checking, in: Proceedings of the Tenth International Symposium on CGO, in: CGO '12, ACM, New York, NY, USA, 2012, pp. 135–144, doi:10.1145/2259016.2259034.

[25] F. Hategekimana, P. Nardin, C. Bobda, Hardware/software isolation and protection architecture for transparent security enforcement in networked devices, in: Proceedings of the 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2016, pp. 140–145, doi:10.1109/ISVLSI.2016.32.
[26] B.A. Huberman, Ensuring trust and security in the industrial iot: the internet of things (ubiquity symposium), Ubiquity 2016 (January) (2016) 2:1–2:7, doi:10.1145/2822883.
[27] Imperva, Ddos attack glossary: Syn flood, April, 2000[Online; accessed August-2015].
[28] R.W. Jones, P.H. Kelly, Backwards-compatible bounds checking for arrays and pointers in c programs., in: Proceedings of the 3rd Int. Workshop on Automatic Debugging, Citeseer, 1997, pp. 13–26.
[29] N. Kayacik, M. Heywood, Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets, in: Proceedings of the 3rd Annual Conference on Privacy, Security and Trust (PST), 2005.
[30] G. Kornaros, D. Pnevmatikatos, A survey and taxonomy of on-chip monitoring of multicore systems-on-chip, ACM Trans. Des. Autom. Electron. Syst. 18 (2) (2013) 17:1–17:38.
[31] W. Li, P. Yi, Y. Wu, L. Pan, J. Li, A new intrusion detection system based on KNN classification algorithm in wireless sensor network, Journal of Elect. and Comp. Engineering (2014).
[32] A. LOTEN, It spending cuts spare the internet of things, 2016.
[33] G. Lyon, Nmap - free security scanner for network exploration, 2014, [Online; accessed 2-March-2015].
[34] S. Mao, T. Wolf, Hardware support for secure processing in embedded systems, Computers, IEEE Transactions on 59 (6) (2010) 847–854.
[35] O. Meltem, D. Caleb, G. Iakov, A.-G. Nael, P. Dmitry, Malware aware processors: A framework for efficient online malware detection, in: Proceedings of the 2015 IEEE 21st International Symposium on High Performance Computer Architecture, in: HPCA, 2015.
[36] S. Nagarakatte, J. Zhao, M.M. Martin, S. Zdancewic, Softbound: highly compatible and complete spatial memory safety for c, in: Proceedings of the ACM Sigplan Notices, 44, ACM, 2009, pp. 245–258.
[37] G.C. Necula, J. Condit, M. Harren, S. McPeak, W. Weimer, Ccured: type-safe retrofitting of legacy software, ACM Trans. Program. Lang. Syst. 27 (3) (2005) 477–526.
[38] NIST, Advanced encryption standard:U.S. national institute of standards and technology (NIST): Federal information processing standards publication (fips pubs) 197, 2001.
[39] U.D. of Homeland Security, Strategic principles for securing the internet of things (IoT), 2016.
[40] G. Press, Internet of things by the numbers: Market estimates and forecasts, 2014.
[41] ProventiaWorks, Ibm proventia network intrusion prevention system (ips) gx5108, 2012.
[42] M. Rahmatian, H. Kooti, I. Harris, E. Bozorgzadeh, Hardware-assisted detection of malicious software in embedded systems, Embedded Systems Letters, IEEE 4 (4) (2012) 94–97.
[43] S. Raza, L. Wallgren, T. Voigt, Svelte: real-time intrusion detection in the internet of things, Ad Hoc Netw 11 (8) (2013) 2661–2674.
[44] RSA, Public-key cryptography standards (PKCS): RSA cryptography specifications version 2.1, 2003.
[45] O. Ruwase, M.S. Lam, A practical dynamic buffer overflow detector, in: Proceedings of the 11th Annual Network and Distributed System Security Symposium, 2004.
[46] A. Saeed, A. Ahmadinia, A. Javed, H. Larijani, Intelligent intrusion detection in low power iots, ACM Trans. Internet Technol. 16 (4) (2016), doi:10.1145/2990499.
[47] E.J. Schwartz, T. Avgerinos, D. Brumley, All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask), in: Proceedings of the Security and Privacy (SP), 2010 IEEE Symposium on, IEEE, 2010, pp. 317–331.
[48] K. Serebryany, D. Bruening, A. Potapenko, D. Vyukov, Addresssanitizer: A fast address sanity checker, in: Proceedings of the USENIX ATC, 2012, 2012.
[49] R. Sommer, V. Paxson, Outside the closed world: On using machine learning for network intrusion detection, in: Proceedings of the Security and Privacy (SP), 2010 IEEE Symposium on, 2010, pp. 305–316, doi:10.1109/SP.2010.25.
[50] G.E. Suh, J.W. Lee, D. Zhang, S. Devadas, Secure program execution via dynamic information flow tracking, SIGARCH Comput. Archit. News 32 (5) (2004) 85–96.
[51] Symantec, An internet of things reference architecture, 2016.
[52] M. Wazid, A.K. Das, R. Hussain, G. Succi, J.J. Rodrigues, Authentication in cloud-driven IoT-based big data environment: survey and outlook, J. Syst. Archit. 97 (2019) 185–196, doi:10.1016/j.sysarc.2018.12.005.
[53] S.X. Wu, W. Banzhaf, The use of computational intelligence in intrusion detection systems: a review, Appl Soft Comput 10 (1) (2010) 1–35, doi:10.1016/j.asoc.2009.06.019.
[54] Xilinx, Lightweight IP application examples, [Online; accessed 8-November-2016].
[55] Xilinx, Standalone LWIP library, [Online; accessed 6-November-2016].
[56] Xilinx, Vivado design suite user guide: Partial reconfiguration, [Online; accessed 10-November-2016].
[57] Xilinx, Zc702 evaluation platform, [Online; accessed 6-November-2016].
[58] Xilinx, Zynq ps-pl packets redirection, [Online; accessed 6-November-2016].
[59] M.-K. Yoon, S. Mohan, J. Choi, J.-E. Kim, L. Sha, Securecore: A multicore-based intrusion detection architecture for real-time embedded systems, in: Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th, 2013, pp. 21–32.
[60] Y. Younan, P. Philippaerts, L. Cavallaro, R. Sekar, F. Piessens, W. Joosen, PAriCheck: an efficient pointer arithmetic checker for c programs, in: Proceedings of the 5th ACM Symp. on Info., Comp. and Comm. Security, ACM, 2010, pp. 145–156.
[61] J. Yu, H. Kang, D. Park, H.-C. Bang, D.W. Kang, An in-depth analysis on traffic flooding attacks detection and system using data mining techniques., J. Syst. Archit. 59 (10) (2013) 1005–1012.



Festus Hategekimana received his Ph.D. degree in Computer Engineering from the University of Arkansas in 2018. His research interests are in hardware assisted system security and its applications to cloud computing and storage.

Taylor JL Whitaker received a Bachelor of Science in Computer Science in 2016 and a Master of Science in Computer Engineering in 2019, both from the University of Arkansas. He is currently enrolled at the University of Florida as a Ph.D. in Computer Engineering candidate. Taylors research efforts in the Smart Systems Lab at UF include hardware security, computer architecture, and robotic applications.

Md Jubaer Hossain Pantho received a Bachelor of Science in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology (BUET). In 2018, he received his Masters degree in Computer Engineering from the University of Arkansas. He worked at Samsung R&D Institute Bangladesh and Xilinx inc. Currently, he is pursuing his Ph.D. in Computer Engineering at the University of Florida. His research efforts in the Smart Systems Lab at UF include hardware security, computer vision, and reconfigurable computing.

Christophe Bobda Professor Bobda received the License in mathematics from the University of Yaounde, Cameroon, in 1992, the diploma of computer science and the Ph.D. degree (with honors) in computer science from the University of Paderborn in Germany in 1999 and 2003 (In the chair of Prof. Franz J. Rammig) respectively. In 2005 Dr. Bobda was appointed assistant professor at the University of Kaiserslautern. From 2007 to 2010 Dr. Bobda was professor at the University of Potsdam and from 2010 to 2018, a professor at the Univeristy of Arkansas. Currently, Dr. Bobda serves as a professor at the University of Florida.