Towards A Layered and Secure Internet-of-Things Testbed via Hybrid Mesh

Tyler Jones, Aniket Dali, Manoj Ramesh Rao, Neha Biradar, Jean Madassery, Kaikai Liu
Computer Engineering Department
San Jose State University (SJSU)
San Jose, CA, USA
Email: tylerjones365@gmail.com,
{*aniket.dali, manojrameshrao.rameshrao, neha.biradar, jean.madassery, kaikai.liu*}*@sjsu.edu*

***Abstract*—The Internet of Things (IoT) is an emerging technology that aims to connect our environment to the internet in the same way that personal computers connected people. As this technology progresses, the IoT paradigm becomes more prevalent in our everyday lives. The nature of IoT applications necessitates devices that are low-cost, power-sensitive, integrated, unobtrusive, and interoperable with existing cloud platforms and services, for example, Amazon AWS IoT, IBM Watson IoT. As a result, these devices are often small in size, with just enough computing power needed for their specific tasks. These resource-constrained devices are often unable to implement traditional network security measures and represent a vulnerability to network attackers as a result. Few frameworks are positioned to handle the influx of this new technology and the security concerns associated with it. Current solutions fail to provide a comprehensive and multi-layer solution to these inherent IoT security vulnerabilities. This paper presents a layered approach to IoT testbed that aims to bridge multiple connection standards and cloud platforms. To solve challenges surrounding this multi-layer IoT testbed, we propose a mesh inside a mesh IoT network architecture. Our designed "edge router" incorporates two mesh networks together and performs seamlessly transmission of multi-standard packets. The proposed IoT testbed interoperates with existing multi-standards (Wi-Fi, 6LoWPAN) and segments of networks, and provides both Internet and resilient sensor coverage to the cloud platform. To ensure confidentiality and authentication of IoT devices when interoperating with multiple service platforms, we propose optimized cryptographic techniques and software frameworks for IoT devices. We propose to extend and modify the existing open-source IDS platforms such as Snort to support IoT platforms and environments. We validate the efficacy of the proposed system by evaluating its performance and effect on key system resources. The work within this testbed design and implementation provides a solid foundation for further IoT system development.**

***Keywords*-6LoWPAN, AWS IoT, Internet of Things**

I. Introduction

The Internet of Things (IoT) promises a multitude of interconnected endpoints capable of providing real-world data in real time. Gartner predicts 20.4 billion connected devices by the year 2020, with consumer spending expected to reach $3 trillion. Light fixtures, medical prosthetics, and a multitude of sensors are examples of today's most commonly used connected devices. This technology has the power to enrich how we interact with the world around us by interconnecting many of the electronic devices we use in our daily lives. This interconnected nature allows for the automation of processes and the collection of data in ways never before possible.

Adopting the Internet of Things means increasing the number of connected devices behind a networks firewall. Due to the resource-constrained nature of these IoT devices, they typically lack any native security, and fuse IP and low-power radio technologies, inheriting vulnerabilities from both [2]. These vulnerabilities make IoT devices ideal attack vectors for any malicious actor looking to gain access to a network. Cryptographic techniques work well to protect the IoT network traffic, but can really only be used as the first line of defense [1]. This creates the need for a layered approach in order to protect all layers of the network.

In this paper, we focus on IoT testbed implementation and the security concerns surrounding the IoT and cloud system. We propose a layered IoT system design including sensing, preprocessing in IoT device, IoT gateway, and connect to existing IoT cloud platform, for example, Amazon AWS IoT. To solve current deficiencies of the interoperability problem in IoT system, we propose a mesh inside a mesh IoT network architecture. The first mesh is the sensor network mesh powered by the 6LoWPAN and IETF RPL. We utilize the TI SensorTag 2 as the node of the sensor mesh. The SensorTag equipped with ten sensors, and the communication protocol can be multi-standard (Zigbee, 6LoWPAN, and BLE) by just changing the firmware. The second mesh is the low cost Wi-Fi mesh network. Instead of deploying a traditional Wi-Fi infrastructure with 2-layers, we utilize low complex mesh Wi-Fi network to supplement the limited cable coverage. We leverage the low cost Raspberry Pi 3 as the Wi-Fi mesh node as well as the 6LoWPAN "edge router". In the mean time, all the IoT data packets will be transmitted to the Amazon AWS IoT and trigger multiple AWS cloud services for further data processing, for example, AWS Kinesis, DynamoDB, QuickSight. We further explore some of the key vulnerabilities in IoT networks, the reasons behind those vulnerabilities, and how an interested individual might mitigate these concerns. We demonstrate the clear need for security in this typical multi-layer IoT setting. We

978-1-5386-7244-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICIOT.2018.00010

propose our layered IoT security system design, its physical implementation, and an evaluation of the IDS. We evaluate our proposed system in terms of efficiency and performance by conducting emulated attacks. The paper concludes by discussing the challenges and obstacles of the project and suggests some solutions to these problems. Lastly, the paper touches on the future work of the project and what it means to the future of IoT.

II. Motivation

A. IoT Technologies and Standards

IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) uses IEEE 802.15.4 as a link and physical layer protocol and requires a border router/gateway to create the network. It defines encapsulation and header compression mechanisms in much the same way standard network protocols do [5]. The key feature that sets this technology apart from its competitors is the fact that it is the only IoT communication technology with true IP capability, allowing all IoT devices to have their own unique IPv6 address. This provides IPv6 identity and Internet connectivity to every node of a wireless sensor network (WSN) [6]. Nodes in the 6LoWPAN are capable of creating a scalable and self-organizing mesh network, which is important when considering the scalability and flexibility demanded by IoT. Adding to that flexibility is the capability of devices within a 6LoWPAN to be application agnostic. These factors make 6LoWPAN a strong contender to dominate the IoT sphere.

The communication protocol used by an IoT network is key to how it operates, but a network is only as secure as its weakest link, and IoT devices represent a multitude of security vulnerabilities for any network. Currently, there are no comprehensive solutions to mitigate this concern, and recent attempts point toward the need for a multi-level solution that incorporates an Intrusion Detection System (IDS) [6]. Cryptographic techniques could be used as the first line of defense, but these can only protect 6LoWPAN from external attackers [1]. In the chance that an IoT device becomes compromised, these cryptographic techniques become useless for the malicious nodes that are internal to the network. These malicious nodes then have the potential to become attack vectors for the entire network, making their discovery crucial to maintaining a healthy and secure network.

B. Primary Issues and Previous Work

The relative novelty of IoT and 6LoWPAN means that few solutions exist for a comprehensive security solution. In 2013, researchers from a leading research institute in Sweden known as SICS developed SVELTE, one of the first known IDS for 6LoWPAN IoT devices [7]. Their system is a host-based IDS that primarily targets spoofed or altered information, sinkhole, and selective forwarding attacks [7]. The project set an important benchmark, but never made its way to physical hardware and was only implemented on Contiki's network simulator Cooja [7].

Another notable contribution pushing 6LoWPAN security forward is the demo on IDS frameworks for IoT and 6LoWPAN [6]. The project focuses on denial of service attacks and uses the open-source Suricata IDS along with Scapy to simulate attacks [6]. The project also adds the Prelude open-source SIEM project to monitor and manage network activity [6]. Much like SVELTE, this project presents an excellent basis for 6LoWPAN security, but again, it lacks any implementation in real hardware. This lack of a hardware implementation makes these previous works more theoretical than foundational and necessitates a solution that is implemented in actual hardware.

C. Proposed Solution

In a layered approach, multiple avenues are taken to design the IoT testbed and protect its data. Data that is transmitted on the un-trusted internet uses cryptographic techniques to keep the data secure. In our implementation, message queue telemetry transport (MQTT) is used to relay sensor data and attack alerts to our Amazon Web Services (AWS). MQTT is an open-source application layer protocol invented by IBM that uses a publish/subscribe architecture [8]. AWS's IoT platform takes advantage of X.509 certificates to establish public and private keys in order to secure the MQTT traffic [9]. We interface the edge node to Amazon's implementation of MQTT broker, i.e., AWS IoT, to trigger multiple cloud services. In our current implementation, we leverage Amazon Kinesis Fire Hose for the big data streaming, Kinesis Analytics for real time data processing, Amazon S3 for data storage, and QuickSight for data visualization. We setup these flows as the application template. Users or developers can build their own application based on our testbed as the template.

As we approach the local 6LoWPAN, we must rely more heavily on detecting signatures and anomalies in the network traffic. Pre-existing research suggests sniffing network traffic near the border router and sending the data to an IDS for further examination [2], [7]. In this way, we are able to create a hybrid IDS edge architecture, which is more suitable for the resource-constrained IoT nodes [10]. To accomplish this, we implement a model based on a edge border router capable of processing and collecting network data, and small edge sensor nodes with restricted computational power that are monitored by an anomaly and signature-based IDS.

III. System Architecture and Design

The implementation of the project can be broken down into three distinct phases: Constructing the 6LoWPAN border router for IoT devices, configuring the MQTT broker with a bridge to AWS IoT, and implementing the edge-based Intrusion Detection System. The basic architecture of our implemented system can be seen in Fig. 1. Each

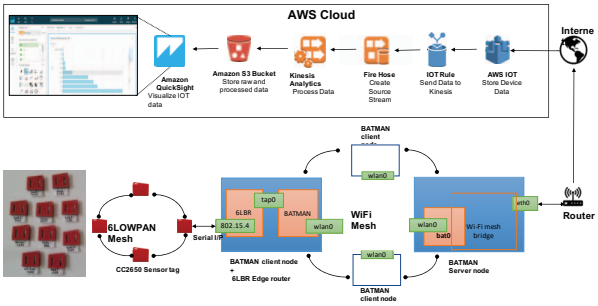


Figure 1. Basic System Architecture.

phase builds upon the previous, making completion of all three phases critical to the project's success. The first two phases produce a fully functioning IoT network where sensor data is persisted in the cloud. These two phases of the project establish a real-world use-case and give context to the development of the third and final phase, the IDS.

A. Border Router and Sensor Nodes

The 6LoWPAN border router utilizes a total of three pieces of hardware for its implementation. The border router itself is hosted on a Raspberry Pi3 Model B, but it needs additional hardware to communicate wirelessly with the end IoT nodes. For this purpose, we use a TI CC-2650 SensorTag combined with a TI CC-DEVPACK-DEBUG debugger to create the external 802.15.4 radio [11], [12]. The 802.15.4 radio allows the nodes in the 6LoWPAN to communicate with the border router. The edge IoT devices that make up our IoT network are all TI CC-2650 SensorTags [11].

Every piece of hardware in the system is running some variant of the Contiki operating system and the Cetic 6LBR project [13]. The Raspberry Pi hosts Cetic's deployment ready 6LoWPAN border router known as the 6LBR. The 802.15.4 radio is built using Cetic's Slip-Radio module, and the IoT sensors are all running Cetics Web-Demo, which includes all of the software needed for the sensors to publish their sensor data via MQTT messaging.

The 6LBR project provides a springboard for a fully functioning IoT network, complete with its own web-server. Through this web interface, the border router provides a way to configure nearly any aspect of the device. It also provides a way to access each connected sensor independently, and either view its data or make changes to its configuration. Additionally, the web interface provides a place to view the WSN's Direction-Oriented Directed Acyclic Graph (DODAG) in real time. From this view, it is possible to see the self-organizing mesh network in action. The edge router feature is impressive in its offerings, but more work is needed to persist the sensor data in the cloud.

B. MQTT Broker

The MQTT broker is hosted by the Raspberry Pi as well and is responsible for propagating our sensor data and IDS alerts to the cloud. This piece of the system was implemented using the open-source Mosquitto MQTT broker [14]. Each sensor in the network is an MQTT client that publishes its sensor data to the local IPv6 address of the broker at port 1883. Similarly, the IDS acts as a client, but it publishes intrusion alerts to the broker's local IPv6 address. Once the broker receives these MQTT messages, it filters them based on topic and publishes them to their intended subscriber.

In our implementation, AWS is the end subscriber of all MQTT messages. A bridge was configured between our local Mosquitto broker and AWS's IoT platform to ensure reliable and secure transmission of the MQTT messages. The configuration file for the bridge includes information about publish/subscribe topics, the AWS endpoint, and security certificates. The certificates are used for authentication and to encrypt the traffic with Transport Layer Security (TLS). Once AWS's IoT platform receives the messages, a rules engine filters the messages based on topic and processes the data accordingly. Both sensor data and intrusion alerts are stored in tables located in AWS's NoSQL database DynamoDB. The data is inserted into the tables based on the rule associated with the data's MQTT topic and contents. Because the data is persisted in an AWS database and updated in real-time, sensor data and intrusion alerts can easily be accessed by any application through a simple query.

C. Intrusion Detection System

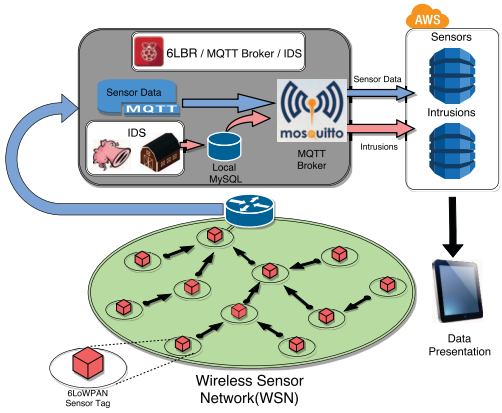


Figure 2. High Level System Architecture.

As mentioned earlier, the IDS would be trivial without the implementation of the 6LBR and MQTT broker; Fig. 2 shows a high-level diagram of the implemented system architecture, complete with the IDS. The project takes advantage of the free and open-source IDS toolkit known

as Snort [15]. Snort, which is now backed by Cisco, was chosen because it is more than capable of the real-time traffic analysis our system demands [15]. Furthermore, it offers the ability to create custom rules that can be fully defined by the user. This level of customization enabled Snort to be used as a backbone to our IDS, while rules are customized to attack types relevant to our WSN.

The format of Snort rules can be broken down into two primary sections, the rule header, and the rule options [16]. The rule header is concerned with the action to be taken and various network options associated with it. The rule options, on the other hand, are more concerned with what parts of the packet should be inspected and what will be displayed/logged in the case of an intrusion event. Putting the two sections together we can create a basic Snort rule in the following format:

alert tcp any any ->any any (msg:“Sample alert”;)

Where the rule parts are separated into action, protocol, src-IP, src-Port, direction-operator, dst-IP, dst-Port, and rule-options respectively.

The options section of the Snort rule may also contain a content flag. Snort looks inside of the packets themselves to determine if the payload contains the string listed under the content portion of the rule options [16]. In order to determine if the packet contains the information listed, Snort implements its own version of the powerful Aho-Corasick pattern matching algorithm [17].

The Aho-Corasick pattern matching algorithm works to create a finite state machine (FSM) from the keywords found in the content section of the Snort rule options, and then using its pattern matching machine, it processes the text in a single pass [17]. This approach results in the pattern matching machine having a time complexity of O(n) [17], which is key to the performance of Snort. The FSM used in Aho-Corasick represents all possible states of the system with information about acceptable state transitions [18]. In this way, the system starts at an initial state, accepts an input event, and moves the current state to the next correct state based on the input event that was given. A study [18] on optimizing the Aho-Corasick algorithm simplifies the algorithm used in Snort to the following:

Algorithm 1: Aho-Corasick based algorithm

```
while input == next-input do
    state = state-transition-table[ state, input ]
    if patterns matched in this state then
        process patterns...
    end if
end while
```

In the default configuration, Snort intrusion alerts are stored in log files, but add-on software was incorporated to store alerts in a relational database for easier access. The Barnyard2 software package offers the ability to persist intrusion events in a relational database by providing a dedicated spooler capable of handling Snort’s unified2 binary output format [19]. The free and open-source software works to efficiently store intrusion alerts in the Raspberry Pi’s local MySQL instance. From here, alerts can be accessed through simple database queries. In this way, a MySQL trigger was written to run a lightweight Python script upon intrusion alert insertions into the database. The trigger publishes specific alert data as an MQTT message to our local MQTT broker. As discussed in the previous section, these MQTT messages are then published to the AWS IoT platform, where they are ultimately persisted in the DynamoDB database.

D. Attack Emulation and Dataset

Two separate types of denial-of-service (DoS) attacks are used to target different components of the system. Sensor nodes are attacked through repeated Internet Control Message Protocol (ICMP) Echo Request messages. In our case, this is done with the Linux Ping6 command directed at a single sensor node. Ping6 is similar to the standard Ping command, except it is directed toward IPv6 rather than IPv4 addresses. The border router, on the other hand, is attacked through a process of opening a plethora of partial HTTP requests with its web server. The SlowLoris penetration testing tool is used to accomplish this goal [20].

SlowLoris is a Perl script designed to open thousands of partial HTTP requests with a web server and keep those requests open for an extended period of time [21]. Web servers typically have a finite number of HTTP requests they can handle at any given time, and SlowLoris causes the server to attempt to fulfill its maximum number of requests. In this way, the HTTP requests effectively serve as a DoS-style attack on the border router.

The datasets for this project include both the sensor data with attack alerts from the IDS and the evaluation results. The sensor data and attack alerts are persisted in database tables on AWS and provide insight into how the system operates. The evaluation results enrich this data by providing numerical measurements that reflect the impact of attack events on our hardware. From reviewing this data, conclusions can be drawn about how certain attacks effect sensor nodes and the border router over time. Furthermore, this data can shed light on the efficacy of the system and can be used to compare our implementation to others with a similar design.

IV. System Evaluation

A. 6LoWPAN IoT Network

To demonstrate the IoT capabilities of sensor nodes to communicate with AWS IOT cloud via 6LOWPAN edge

router, we setup the experimental environment as shown in Fig. 3. In deployment map 1, the distance of node 2 from edge device is 421 inches; the distance of node 3 from edge device is 359 inches; the distance of node 1 from edge device is 255 inches. In deployment map 2, the distance of node 2 from edge device is 568 inches; the distance of node 3 from edge device is 335 inches; the Distance of node 1 from edge device is 573 inches. The node 2 in deployment map 2 is two hops away from the edge device.

The setup consists of five primary functional blocks, mesh of sensor nodes, edge device, MQTT client, Internet connection and cloud platform. The mesh of sensor nodes is formed by Texas Instruments CC2650 sensor tag nodes. All the nodes are running a 6LOWPAN web server application based on Contiki 3.0. The web server application is configured to periodically publish the sensor values to the AWS IoT. The edge device hardware is formed by 6LOWPAN SLIP radio and Raspberry PI 3. The edge device runs 6LBR edge router software and AWS MQTT bridge to channel all the AWS packets from 6LBR to AWS IOT. The edge device is configured in the router mode as shown in the Fig. 4, it forms and manages the 6LOWPAN network via the serially attached SLIP radio. The Internet connectivity to the edge device is provided by the ethernet. A network switch is connected in between the edge device and the computer to maintain the similar network prefix, to configure the nodes and monitor the traffic from the 6LOWPAN sensor network in real time.

We also designed a management framework to track the health and status of the network. We collect all the battery temperature and battery voltage level of sensor nodes as shown in Fig. 5. In case some nodes' status is abnormal, we can take immediate actions. As shown in Fig. 4, we also record the total traffics of the sensor nodes and get to know the overall statistical traffic information.

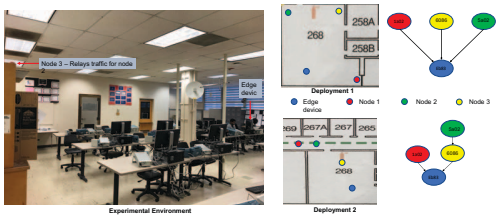


Figure 3. The deployment of the sensor mesh network.

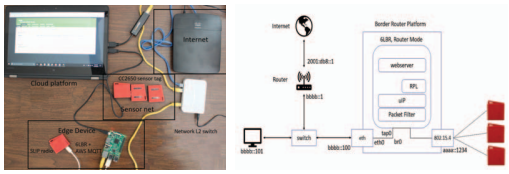
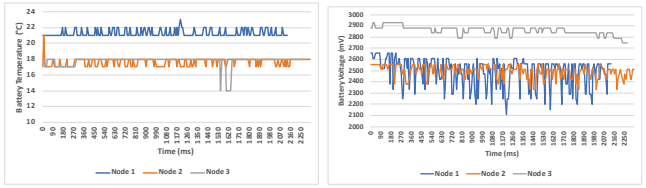
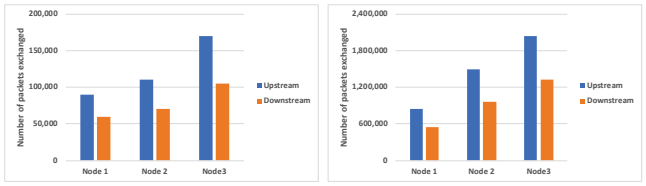


Figure 4. The setup of the sensor mesh network.



(a) Battery temperature (b) Battery voltage

Figure 5. The battery temperature and battery voltage level of the sensor nodes.



(a) Deployment map 1 (b) Deployment map 2

Figure 6. The recorded traffics of all the sensor nodes.

B. WiFi Mesh Network

To extend the network coverage, we formed an ad-hoc mesh Wi-Fi network using BATMAN -adv on the border router, i.e., Raspberry Pi 3B. The internet access will be shared across all the nodes. To analyze the effect of number of hops on internet speed, we setup an experiment as shown in Fig. 7. which consists of four primary functional blocks, Raspberry Pi 3B node running B.A.T.M.A.N adv, router, Internet connection and a PC to configure each node using SSH. The Ad-hoc mesh of nodes is formed by Raspberry PI 3B nodes. The Ad-hoc Wi-Fi mesh is configured by installing batman-adv 2018 version for raspberry by Open-Mesh. Node in network 1 is connected to internet via ethernet by the router. Nodes in network 2 and 3 do not have access to internet and are connected to a PC, to configure and log data via SSH. Similarly, Node 1 is managed by a PC in network 1. All the three nodes are configured to use on board Broadcom BCM43438 wireless wlan0 interface to participate in the Ad-hoc Wi-Fi mesh.

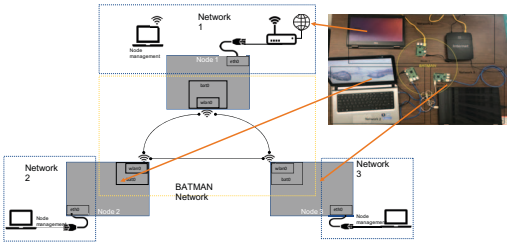


Figure 7. The setup of the WiFi mesh network.

To automate the experiment setup process, we developed a test scrip that can automatically configure the Ad-hoc Wi-Fi mesh. Specifically, we configure wlan0 to participate in

wireless-mesh network and form a bridge between eth0 and bat0 in server mode. An interface bat0 is initiated with same IP address leased from DHCP server to participate in the network connected to the internet. The mesh is configured to use channel 8 of 2.4 Ghz spectrum (2.447Ghz). wlan0's configuration is changed from managed by DHCP service to ad-hoc mode with MTU of 1532 bytes. Server node is configured to 10mbit/2mbit up down link rates. Another test script is developed to activate and configure Batman-adv for client node, i,e., activate the batman-adv, and configure wlan0 to participate in wireless-mesh network, and assign client node an Ip address in the range of default gateway node. An interface bat0 is initiated with same IP address leased from DHCP server to participate in the network connected to the internet. The mesh is configured to use channel 8 of 2.4 Ghz spectrum (2.447Ghz). wlan0's configuration is changed from managed by DHCP service to ad-hoc mode with MTU of 1532 bytes. In the final steps of the test script, we verify whether all the nodes are on the same network by checking the configuration of wireless interfaces and verify that the Cell radios on all three nodes matches. We finally get the trans global debug table, which shows the neighbors on the network, the network Id of their bat interfaces and the hardware MAC id of the physical interface contributing in the network.

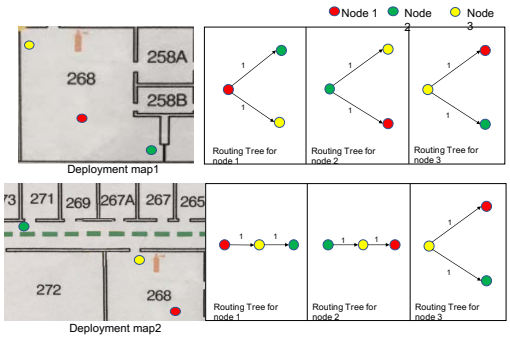
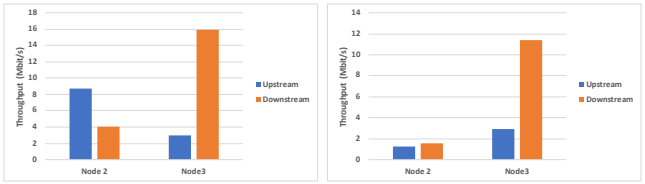


Figure 8. The deployment map and routing tree of the WiFi mesh network.

To simulate the real world deployment, we deployed these three nodes with sufficient distances in two indoor environments as shown in Fig. 8. In deployment map 1, Node 2 and 3 are placed at the extreme end of the room 268, away from the node 1 to experiment effect of distance on the routing table and throughput. The Room area is 484 square feet. The Distance of node 2 from node1 is 492 inches. The Distance of node 3 from node1 is 255 inches. In deployment map 2, the Distance of node 2 from node1 is 712 inches. The Distance of node 3 from node1 is 304 inches.

Fig. 9 shows the throughput analysis for upstream and downstream throughput of node 2 and node 3 to reach node 1. Fig. 10 and Fig. 11 show the latency for node 1, node 2 and node 3 to reach other BATMAN nodes in deployment map 1 and map 2, respectively. These results can demonstrate the effectiveness of our ad-hoc Wi-Fi network for our IoT testbed.



(a) Deployment map 1 (b) Deployment map 2

Figure 9. The network throughtput.

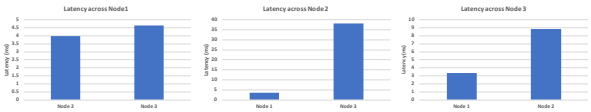


Figure 10. The latency between different nodes in deployment map 1.

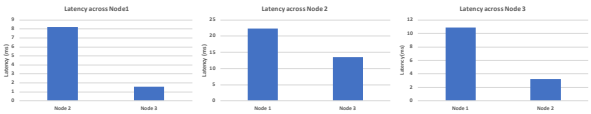


Figure 11. The latency between different nodes in deployment map 2.

C. Secure Edge Gateway

In order to evaluate the efficiency and reliability of the IDS solution, we focus on common DoS-style attacks and their impact on our IoT hardware. DoS attacks are implemented on both the 6LBR and the sensor nodes in order to understand various attack types and their repercussions to the rest of the system, both when the IDS is active, and when it is not. In our implementation, a Raspberry Pi 3 acts as our border router, MQTT broker, and it hosts our IDS. The TI sensortag with ARM Cortex M3 microcontroller acts our sensor node. Any and all network traffic outside of the local 6LoWPAN must pass through this border router in order to reach any of the IoT nodes. We will be evaluating the capabilities of a Raspberry Pi or similar embedded system for operating as a border router, MQTT broker, and running a full IDS concurrently.

1) DoS Attacks to the sensor node and border router: The sensor nodes are the most resource-constrained components in our system. We generate the IPv6 attacks to the sensor node via MQTT messaging and evaluate the impact of the attack. A single unit of attacker consists of the Ping6 ICMP attack for every second. When we launch four attackers to one sensor node simultaneously, the sensor is typically only able to publish approximately 60% of its MQTT messages. In most cases, five or more attackers directed at a single

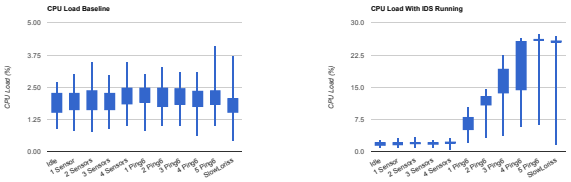
sensor will render it incapable of publishing any MQTT messages. These metrics vary substantially depending on the state of the sensor and thus were not a part of the evaluation. This information has been included only to demonstrate how easily the resource-constrained sensors can be disabled through simple DoS-style attacks.

The border router has considerably more computing power than the sensor nodes, so the SlowLoris penetration tool is used to aid in a DoS attack. Because all possible requests are occupied through SlowLoris, the border router experiences a strain on its resources and is unable to serve any further requests. This attack can also indirectly affect other system resources surrounding the web server, such as the border router's ability to act as an MQTT broker or functioning IDS.

In order to build a useful dataset, we monitor multiple events for their effect on CPU load and RAM usage. First, a baseline was established by taking measurements while sensors were actively sending data to AWS IoT. We begin with the system idle (zero sensors) and sequentially add sensors until four are running concurrently. Each sensor node sends MQTT messages at an interval of approximately three seconds. After all four sensors are consistently sending data, we begin running attackers directed at a single sensor. We begin with one attacker, and add attackers until five Ping6 attacks are running concurrently. As noted in the previous section, running five attacks typically disables the sensor, so we limit the evaluation to five active windows. Lastly, we execute the SlowLoris DoS attack ti tge border router. This is done with all four sensors transmitting their data, and with zero ICMP requests to the sensors. To establish a baseline from which we draw comparisons, the tests were completed both with the IDS active and with it deactivated. CPU load and RAM usage were collected through a custom Python script. The script outputs both metrics every second, providing a large dataset.

2) Evaluation Results: The results of our evaluation were illuminating for two key reasons. They show the effects of DoS-style attacks on a WSN with virtually no security, and they provide insight into the capabilities of resource-constrained hardware for running an IDS locally. The data illustrates that running basic DoS-style attacks, although disruptive to specific parts of the system, had little overall effect on total CPU load and RAM usage for the border router with no security in place. Activating the IDS, however, significantly increased resource consumption, especially in the event of an attack.

From Fig. 12a it is clear that the DoS attacks implemented in the evaluation had little effect on overall CPU load for the Raspberry Pi. Even during various attacks, the CPU never reached 5% total utilization, and the majority of the data points have it near 2% in all cases. When running the IDS, these numbers increase significantly in the case of an active attack. Fig. 12b demonstrates that as the number of detected



(a) Baseline CPU Load (b) CPU Load With IDS Running

Figure 12. a) 6LBR Baseline CPU Load; b) 6LBR CPU Load With IDS Running.

attack events by the IDS increases, so does the overall CPU load, raising the total load to nearly 30% in some cases. We can also see that in the case of four Ping6 commands, the majority of the data-points cover a significant range. This increased range may be correlated to the loss of MQTT messages received from the sensor. It suggests that when CPU load is in the lower part of that range, MQTT messages are still being processed, but may not be at higher CPU loads.

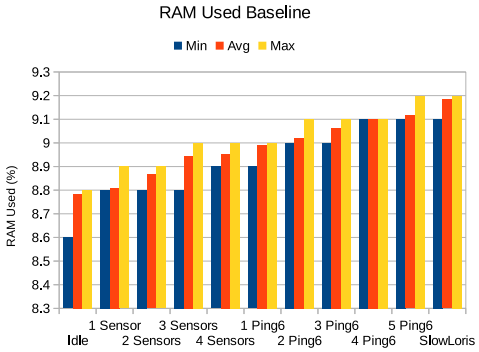


Figure 13. 6LBR Baseline RAM Usage.

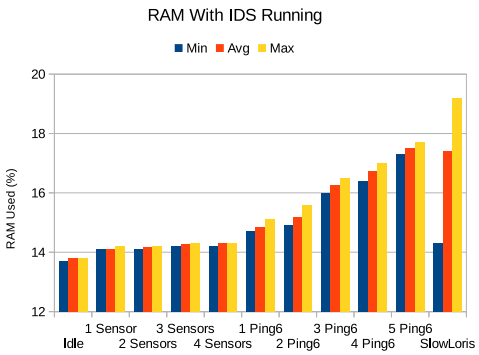


Figure 14. 6LBR RAM Usage With IDS Active.

RAM usage followed a similar trend to that of CPU load, with little variation when the IDS is not active. Fig. 13

shows that regardless of attack state, the percentage of RAM used by the entire system varies by less than 1% with the IDS deactivated. This variance increased significantly when the IDS became active, raising it to nearly 6% in some cases. Fig. 14 shows this variance and illustrates a clear progression of the percentage of RAM used as the number of detected attacks increased. A large variation between maximum and minimum percentage of RAM used in the case of the SlowLoris attack can be observed in Fig. 14. This is the result of the batch-like nature of the attack. The SlowLoris tool makes a large number of HTTP requests in a batch, with a small window between batched requests. The minimum value is a result of one of the windows, while the maximum value results from one of the batched requests.

For the purpose of this paper, we use CPU load and RAM usage metrics to determine the efficacy of running our IDS, and in both cases, these metrics stayed within normal operating limits for the duration of the evaluation. Both the ICMP type attacks and the SlowLoris web-server attacks had little effect on the resources of the Raspberry Pi, and although the IDS demanded significantly more resource use, it did not strain the system.

V. Conclusion

This project's approach to IoT and 6LoWPAN security is based on the consensus that a layered approach is needed in order to accommodate the resource-constrained nature of many IoT networks. The project leveraged a number of open-source tools to both construct and evaluate the system and its design. Through this work, the project establishes a solid foundation for a hybrid IDS that can be effectively run on many of today's inexpensive embedded systems (such as the Raspberry Pi).

Acknowledgment

The work presented in this paper is funded by National Science Foundation under Grant No. CNS 1637371.

References

[1] P. Kasinathan, C. Pastrone, M. A. Spirito, and M. Vinkovits, "Denial-of-service detection in 6lowpan based internet of things," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*. IEEE, 2013, pp. 600–607.

[2] A. Le, J. Loo, A. Lasebae, M. Aiash, and Y. Luo, "6lowpan: a study on qos security threats and countermeasures using intrusion detection system approach," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1189–1212, 2012.

[3] L. M. Oliveira, J. J. Rodrigues, A. F. de Sousa, and J. Lloret, "A network access control framework for 6lowpan networks," *Sensors*, vol. 13, no. 1, pp. 1210–1230, 2013.

[4] P. Kasinathan, G. Costamagna, H. Khaleel, C. Pastrone, and M. A. Spirito, "An ids framework for internet of things empowered by 6lowpan," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 1337–1340.

[5] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," *Ad hoc networks*, vol. 11, no. 8, pp. 2661–2674, 2013.

[6] J. Olsson, "6lowpan demystified," *Texas Instruments*, p. 13, 2014.

[7] N. Corbett. (2017, May) Understanding the aws iot security model. [Online]. Available: https://aws.amazon.com/blogs/iot/understanding-the-aws-iot-security-model/

[8] P. Pongle and G. Chavan, "A survey: Attacks on rpl and 6lowpan in iot," in *Pervasive Computing (ICPC), 2015 International Conference on*. IEEE, 2015, pp. 1–6.

[9] Simplelink multi-standard cc2650 sensortag kit reference design. [Online]. Available: http://www.ti.com/tool/TIDC-CC2650STK-SENSORTAG

[10] Simplelink sensortag debugger devpack. [Online]. Available: http://www.ti.com/tool/CC-DEVPACK-DEBUG

[11] (2012) 6lbr a deployment-ready 6lowpan border router solution based on contiki. [Online]. Available: http://cetic.github.io/6lbr/

[12] Mosquitto: An open source mqtt v3.1/v3.1.1 broker. [Online]. Available: https://mosquitto.org/

[13] Snort - network intrusion detection & prevention system. [Online]. Available: https://www.snort.org/

[14] Snort users manual 2.9.11: The snort project. [Online]. Available: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/

[15] S. Sen, "Performance characterization & improvement of snort as an ids," *Bell Labs Report*, 2006.

[16] M. Norton, "Optimizing pattern matching for intrusion detection," *Sourcefire, Inc., Columbia, MD*, 2004.

[17] Firnsy/barnyard2. [Online]. Available: https://github.com/firnsy/barnyard2

[18] Ogglas/orignal-slowloris-http-dos. [Online]. Available: https://github.com/Ogglas/Orignal-Slowloris-HTTP-DoS

[19] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *IEEE communications surveys & tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.