Ultra-Lightweight Deep Packet Anomaly Detection for Internet of Things Devices

Douglas H. Summerville, Kenneth M. Zach and Yu Chen
Department of Electrical and Computer Engineering
State University of New York at Binghamton
Binghamton, NY, USA
{dsummer,kzach1,ychen}@binghamton.edu

***Abstract*— As we race toward the Internet of Things (IoT), small embedded devices are increasingly becoming network-enabled. Often, these devices can't meet the computational requirements of current intrusion prevention mechanisms or designers prioritize additional features and services over security; as a result, many IoT devices are vulnerable to attack. We have developed an ultra-lightweight deep packet anomaly detection approach that is feasible to run on resource constrained IoT devices yet provides good discrimination between normal and abnormal payloads. Feature selection uses efficient bit-pattern matching, requiring only a bitwise AND operation followed by a conditional counter increment. The discrimination function is implemented as a lookup-table, allowing both fast evaluation and flexible feature space representation. Due to its simplicity, the approach can be efficiently implemented in either hardware or software and can be deployed in network appliances, interfaces, or in the protocol stack of a device. We demonstrate near perfect payload discrimination for data captured from off the shelf IoT devices.**

Keywords—Internet of Things; network anomaly detection;

I. INTRODUCTION

Despite advances in security technology, our current Internet remains vulnerable. Weakly protected resources are easily subverted and amassed into distributed collectives whose power can be turned against high-value targets. As we move toward an Internet of Things (IoT), threats multiply as the abundance of small unprotected devices can be aggregated and leveraged for wrongdoing. Privacy and physical safety are also at risk as many IoT devices provide a bridge from cyberspace to the physical world. To make matters worse, as more low-end devices become Internet-enabled it can be expected that common computational platforms will emerge, opening the door to zero-day attacks as large numbers of devices share common vulnerabilities.

Small, resource-constrained systems, however, often dedicate what little computing power they have to providing features or services. Design constraints, such as the need to increase performance or battery life, may restrict the ability of system designers to implement security effectively.

The threat from zero-day attacks and the lack of resources makes the use of signature-based detection approaches unsuitable in an IoT environment, given the need for a potentially large database of known attacks. Fortunately, small resource constrained devices execute fewer and potentially less complex network protocols than general purpose computing platforms. This results in less complex patterns of communication, making it easier to detect when such patterns have changed. Therefore, anomaly based detection methods, which attempt to identify deviations in measured statistics against a normal model of operation of a system, can be beneficial to use in resource constrained systems—if computational resource requirements can be kept low.

We have developed a high-performance ultra-lightweight deep-packet anomaly detection approach that is feasible to run on the smallest network-enabled embedded devices. Our approach was designed from the ground up to support low latency and high-throughput implementation in either hardware or software, while remaining competitive in terms of size and detection performance. It can be applied in either a stateless (packet-based) or stateful (connection-based) configurations, depending on the point of deployment. It can be implemented at the application level or in the network stack, network interface or within any network appliance. The approach is highly configurable, making it suitable for a wide range of anomaly detection tasks, yet well-defined, meaning that detectors may not have to be relearned at each point of deployment. Finally, protection can scale with available resources, allowing complex systems with more resources to implement the increased protection they require.

Our approach uses efficient bit-pattern matching, as in [1], to perform feature selection. Bit-patterns provide flexibility to match n-gram sequences for payload modeling and require little computational overhead. Using a windowing approach, counts of the sequences are obtained; these define the features of the detector. These extracted features are used to index a look-up table which stores a direct binary representation of the resulting discrete feature space. This efficient and simplistic detector allows the representation of arbitrary discrete shapes

This work was supported, in part, by the Cyber Research Institute, Rome NY, USA.

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

in the feature space, which provides excellent ability to discriminate normal payloads from anomalous ones.

The remainder of this paper is organized as follows. Section II provides an overview of related work. Section III describes our anomaly detection approach in detail. Section IV provides an analysis of the traffic from two IoT devices and an evaluation of the detection performance of our approach for common attack scenarios. Section V concludes the paper.

II. Related Work

Although much research has been done in designing systems for the detection of anomalous packets and intrusions, only a handful discuss the use of payload analysis to perform feature selection. To our knowledge, no specific approaches have been developed for IoT devices.

One early detection scheme, NATE [2], uses packet headers for feature selection. Another, NETAD [3], used the first 48 bytes of each packet. Neither are able to provide enough information to accurately characterize packet payload.

Reference [4] developed a course grained approach that uses byte frequency distribution over the ranges 0, 1-3, 4-6, 7-11, 12-15 and 16-155. Thus, creating a service specific intrusion detection system by combining type, length, and payload byte distribution as features for statistical modeling of normal traffic.

In contrast, a fine grained approach called PAYL was introduced in [5], which uses n-gram analysis to model full byte frequency distribution over different connection window sizes. In this approach, when n = 1, a 256-bin histogram of byte distribution is obtained. Although this histogram gives the byte frequency distribution, it does not take into account the relative position of these bytes, which leaves this approach vulnerable to an attacker adding padding bytes to the payload. If $n \geqslant 2$, one can better model the structure of the payload. However, the size of the feature space is 256^n, enormously increasing computational complexity even for small n.

A few attempts at improving PAYL have been proposed. Wang [6] developed a system called ANAGRAM which avoids some of the computational complexity of PAYL. ANAGRAM implements Bloom filters, one for normal traffic and one for known attacks. Then, a score is developed for unobserved n-grams in a payload and weighted by the number of malicious n-grams in that payload. Despite the improvement in complexity, effective Bloom filter implementations may not be possible on resource constrained embedded devices [9].

Additional approaches at improving computational complexity have been explored. McPAD [7] measures features using a sliding window to analyze pairs of bytes that are distance v apart. Performing this 2-v-gram analysis multiple times for different values of v, one creates a multiple classifier system. The resulting MCS approximates n-gram analysis, with $n>2$, and limits the dimensions of each classifier's feature space to 256^2. HMMPayl [8] also uses a sliding window approach to extract features from the payload. In this process, the window is moved only one byte at a time, but varies the size of the window from one implementation to another. The set of sequences obtained from this approach is then processed using a Hidden Markov Model. This approach reduces the computational complexity as compared to PAYL since larger window sizes negligibly increases complexity.

Although these improvements obtain solid detection results, neither are able to reduce computational complexity as well as our approach. In McPAD, the dimensionality of each classifier's feature space is 256^2, but adding more classifiers increases complexity. HMMPayl attempts to avoid this, but has a computational complexity of $O(n^4)$ for each classifier, where n is the length of the window chosen. However, our approach only increases linearly as we increase the size of our feature space.

The precursor to our proposed approach was lightweight stateless payload inspection, LiSP [1]. LiSP uses bit-patterns to generate n-gram features from packets, which are mapped to a two dimensional feature space called a bitmap. Our approach improves on LiSP in a number of significant ways that make it better suited for IoT devices. By using a sliding window the dimensionality of the feature space can be better controlled. In addition, the sliding window facilitates uniform detection within payloads. By supporting more than two dimensions and allowing different n-gram and grid sizes for each, the proposed approach achieves better detector discrimination and makes it harder for an attacker to evade detection. These improvements also allow our approach to operate in connection-oriented environments and reduce the complexity of the overall implementation.

III. Ultra-Lightweight Anomaly Detection

Network payloads are treated as a sequence of bytes, $(b_i)_{i \in N}$. Feature extraction operates on overlapping tuples of bytes, called n-grams. The value of n can be unique for each dimension d of the feature space. Fig. 1 illustrates the division of input bytes into n-grams and n-grams into windows. The n-gram starting at byte b_i, denoted b_i^n, is the n-tuple of bytes $(b_i b_{i+1}, \ldots, b_{i+n-1})$. Processing payloads as n-grams, rather than bytes, makes it more difficult for an attacker to evade detection by matching the statistical profile of normal payloads though padding or byte substitution. Allowing each dimension to process input using its own n-gram size raises the bar even further, since the attacker has to match the statistical profile of the payload for every n-gram size in use by the detector.

The detector operates on a windows of bytes with window size w and stride length s, as shown in Fig. 1. The j^{th} window of bytes, W_j, is the byte sequence $(b_{js}, b_{js+1}, \ldots, b_{js+w-1})$. For each dimension k of the detector, byte window W_j is processed

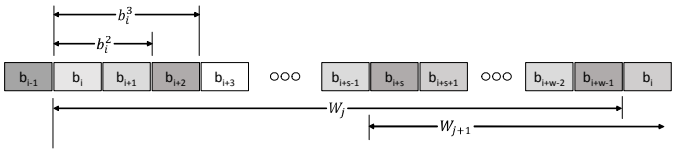


Fig. 1. Payload bytes are processed as n-grams, where each feature (dimension) can use a different value for n. Feature extraction uses a windowed approach with window size w bytes and stride length s bytes.

as the sequence of n_k-grams:

$$\left(b_{js+m}^{n_k}\right)_{m=0}^{w-n_k+1} \quad . \tag{1}$$

There can be a small difference in the number of n-grams processed for each dimension due to fringe effects at the end of the window. Our ultra-lightweight anomaly detection can be applied to either connection-oriented (stateful) or packet-based payload processing, depending on the operating environment at the point of deployment. In either case, at most a single window of payload is the only state required to be kept. Both window size and stride length can be adjusted to optimize detection based on the characteristics of the payload being monitored.

A d-element feature vector (c_0, c_1, ... c_{d-1}) is computed for each window of bytes. Feature c_k is a count of n_k-grams that match the bit-pattern associated with dimension k. Bit-patterns [1] are binary vectors containing don't care positions; alternatively, bit-patterns are ternary vectors of elements from the set *{0,1,X},* where X means match either a 0 or 1. A match between a bit-pattern and n-gram occurs when the corresponding bits match in all positions. For example, the bit-pattern 010XXXXX matches 1-grams (bytes) in the intervals [64,95], which for ASCII encoded values is the set containing predominantly the upper-case letters. Likewise, the bit-pattern 01XXXXXX001XXXXX matches 2-grams whose first byte is in [64,127] and whose second byte is in [32,63], which for ASCII encoded data would capture the set of 2-grams representing predominantly letters followed by numbers or punctuation. Bit-patterns are not specific to a particular data encoding (e.g. ASCII) and can be used to model any type of payload. Bit-patterns were chosen for feature extraction because they are extremely efficient to compute and intuitive to interpret.

Let P_k be the bit-pattern associated with feature k of the detector. Because standard digital hardware cannot store and process ternary values directly, a bit-pattern is stored as a mask-value pair (M_k,D_k), each of which is an $8 \cdot n_k$ bit binary value. M_k, the mask, encodes the bit positions in the n_k-gram that are to be matched with a 1; alternatively, it encodes the don't care positions as 0. D_k encodes the positions of 0's and 1's of the bit-pattern. A bit-pattern match operation on n_k-gram, therefore, is given by

$$match\left(P_k, b_i^{n_k}\right) = \begin{cases} 1 & ; if\, M_k \cdot D_k = M_k \cdot b_i^{n_k} \\ 0 & ; otherwise \end{cases} , \tag{2}$$

where $\cdot$ represents a bitwise-AND operation. Therefore, the bit-pattern match requires a single AND operation and a single equality comparison per n-gram. For the j^{th} window of payload bytes, element c_k of the feature vector is therefore given by

$$c_k = \sum_{i=js}^{js+w-n_k+1} match(P_k, b_i^{n_k}) \quad . \tag{3}$$

It is useful to note that the feature space is both discrete and bounded, with each feature c_k falling in the interval $[0,w]$. Furthermore, each of the finite number of points the discrete feature space is binary.

The feature extraction method described requires $O(d \cdot w)$ trivial operations (bitwise AND with equality comparison) per window of n-grams, all of which could potentially be done in parallel or pipelined. Thus, there is a great potential for optimization in either hardware or software. Furthermore, a new match operation can be performed as each payload byte arrives; thus, real-time implementations are feasible.

A conceptual illustration of a 3-dimensional feature space is shown in Fig 2. Since the binary feature space is discrete and finite, as described above, we chose to implement it with a simple lookup table (requiring at most w^d bits) rather than using a discrimination function that would require additional computation resources and time. As a result, run-time evaluation of a feature vector merely requires computing the index into the multidimensional array and subsequently reading the binary detection decision. In addition to computational efficiency, there are performance advantages to this approach as well. Unlike more complex approaches that use distance metrics which force the normal region(s) to be constrained to well-defined mathematical shapes, our approach can represent arbitrarily shaped regions because each point is individually mapped to a binary decision. This allows our approach to potentially represent nnormal regions with more accuracy, reducing false negatives.

In many cases, the full resolution of d bits per dimension in not needed. Thus, we allow the resolution in each dimension to be quantized by some chosen amount. For example, when using a window size of 256, one dimension could have the full range [0,255], while another could be reduced by half to [0,127] by dividing the feature count by 2. This both reduces storage requirements when a particular region of the space is unimportant and allows tradeoffs to be made between performance and storage requirements, if necessary. It is expected that in the embedded systems for which this approach is intended, most feature spaces will be low-dimensional and of relatively low resolution; the largest space we use in this study, for example, is 64^3 bits or 32 kilobytes.

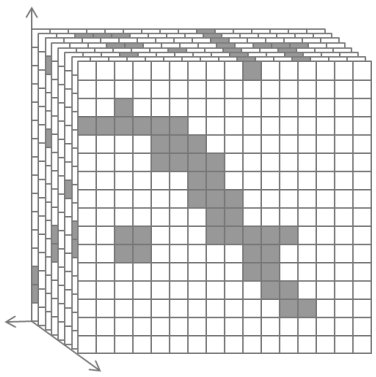


Fig 2. Conceptual view of a 3-dimensional feature space. The space can be represented by a table of binary values, allowing arbitrary shapes to be modelled.

IV. Experimental Evaluation

We have evaluated the proposed approach against traffic obtained from two Internet-enabled devices: a weather station and an interactive networked video camera. These were chosen, respectively, to represent an IoT device that primarily outputs data, which we call a sensor, and one that primarily receives control commands, which we call an actuator. Since the Internet of Things will bridge the cyber and physical worlds, these two types of devices, each representing one direction of that bridge, are a good starting point for analysis.

The network camera we tested is a Foscam FI8910W Pan & Tilt IP/Network Camera. The device is controlled through a series of HTTP GET requests containing the parameters necessary to manipulate the camera. While the device has two-way traffic, with the return traffic containing video and audio, we only evaluated the control traffic destined for the camera. This is the most likely protection scenario in a real-world environment. However, additional detectors could be defined to simultaneously monitor configuration traffic destined for the device, as well as for outgoing audio or video data.

The weather station, being representative of a network sensor, generates traffic primarily in the output direction. The device tested is an Ambient Weather WS-1001-WIFI Observer solar powered wireless remote monitoring station mounted in an outdoor location to obtain realistic readings. This device periodically reports current weather conditions to an Internet service through a series of HTTP GET requests.

We collected traffic from the two IoT devices, as well as generic HTTP traffic, to obtain three benchmark data sets: IoT control device, IoT sensor device, and generic HTTP. The latter data was collected for the purpose of evaluating the discrimination ability of the detector, both of which use HTTP as their underlying protocol. While the 1999 DARPA intrusion detection benchmarks [10] have been widely used to evaluate network anomaly detection systems, we chose not to evaluate the proposed approach against these for three reasons. First, a bit-pattern based anomaly detection approach has already been evaluated in [1] and was found to have very good performance. As the proposed approach represents a superset of that work we could have simply tuned a detector to match the given parameters reported and obtained similar results. Secondly, the anomaly detector proposed in this work is intended for protection of resource-constrained devices, while the DARPA benchmarks represent traffic from general-purpose hosts and servers. Finally, the DARPA benchmarks are over 15 years old and may not be truly representative of modern network traffic. The generic web data we collected is likely more representative or modern payload characteristics.

A. Traffic Analysis

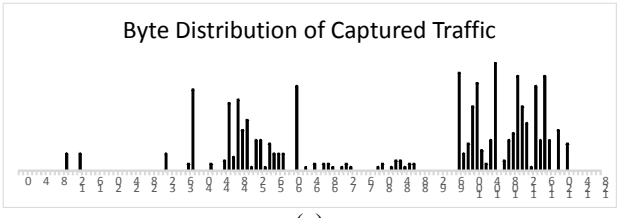
The weather station transmits sensor readings to a server encoded in HTTP GET request packets. Each packet contains a TCP payload of approximately 525 bytes encoding a single vector of readings. During traffic collection, TCP control packets with no payloads were filtered out of the stream and 18648 packets were collected to represent normal operation, with an additional 2141 for evaluation of false positives.

The camera receives control packets from user web browsers, also encoded within HTTP GET request. As before, TCP control packets with no payloads were filtered out of the incoming traffic and 4123 packets were collected to represent normal operation, with an additional 800 for testing of false positives.

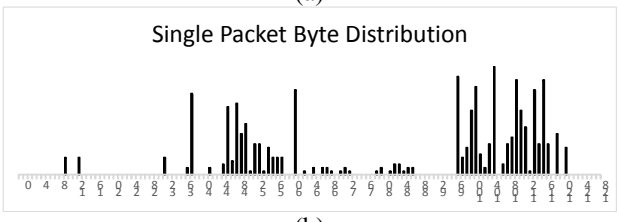
For comparison, generic HTTP payload data was collected on a network. Outgoing traffic with a destination TCP port of 80 and was collected until 5,000 packets with payloads were obtained. This traffic included requests to a wide range of sites and also included multimedia and other user data streams.

For illustration purposes, Fig 3 shows the distribution of byte values for the traffic collected from the control device, as well as the byte distribution for a randomly selected single packet. No byte values above 127 were observed, as the traffic represents standard 7-bit ASCII encoded characters. The figure shows that the distribution of bytes in a single sensor packet is similar to the aggregate distribution of the entire traffic set, suggesting that the distribution is similar among packets.

To evaluate this more precisely, the cosine similarity metric was computed for all pairs of packets in each of the three data sets. Fig 4 shows the distribution of inter-packet cosine similarity values for each of the data sets. Fig 4(a) shows that for the generic web traffic, the packets exhibit a wide-range of dissimilar distributions. By contrast, the traffic from each of the IoT devices exhibits a very high degree of similarity among packets (Fig 4(b) and Fig 4(c)). A summary of the minimum, maximum and mean cosine similarity values is provided in Table 1. The IoT sensor displayed the highest degree of similarity among packets; the IoT control device had slightly more variability in the types of messages used in its protocol exchanges and thus had a slightly higher dissimilarity among packets, though not nearly as much as the generic web traffic. This data suggests that, as presumed, these IoT devices use few protocols and that these protocols are not very complex, as complex protocols would exhibit more dissimilarity among packets.



(a)



(b)

Fig 3. (a) Distribution of payload bytes for camera control traffic; (b) distribution of payload bytes for a randomly selected packet. The single packet bears a remarkable similarity to the aggregate traffic.

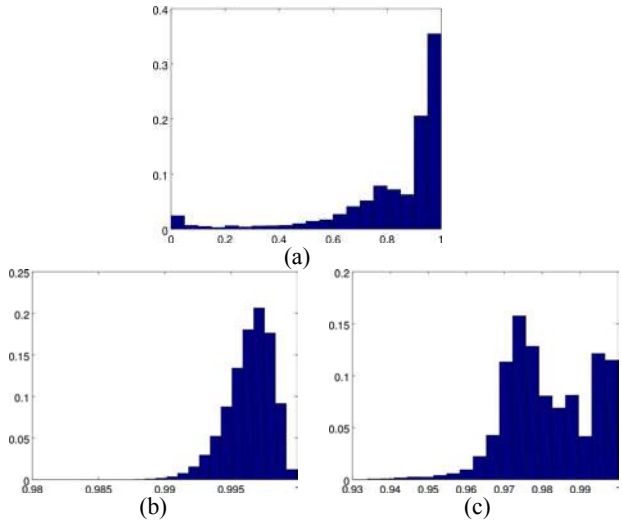


Fig 4. Distribution of inter-packet cosine similarity values for (a) generic web traffic; (b) IoT sensor traffic; and (c) IoT control traffic. Note the different horizontal axes. Packet byte distributions of IoT device traffic are remarkably uniform across packets. The IoT control traffic had a few outliers that were not visible in the graphs and were omitted.

Table 1. Summary of cosine similarity values among packets within each data set. The generic web traffic exhibits a wide range of packet byte distributions, while the IoT device packets show very high similarity.

Traffic Set	Minimum Similarity	Maximum Similarity	Mean Similarity
Generic Web	0	1.0	0.825
IoT Sensor	0.984	1.0	0.996
IoT Control	0.536	1.0	0.979

B. *Anomaly Detection Performance*

To assess the performance of the proposed ultra-lightweight deep packet anomaly detection approach on the IoT device traffic, we trained a detector for each device on its collected normal traffic and evaluated it against various attack scenarios, including worm propagation, tunneling, SQL code injection, and directory traversal attacks. The parameters of each detector were manually tuned based on the payload traffic patterns observed in the data set. Once bit-patterns, window size and stride were chosen, collected traffic was trained against the detector to identify the normal regions of the space. Subsequently, attacks were developed and evaluated against the trained detector.

Bit-pattern features were selected intuitively using the following heuristic. First, n-gram patterns in the payloads were identified by inspecting the histogram of 2-grams of each payload. To improve detection accuracy without over-learning normal payloads, bit-patterns were chosen to match commonly occurring sequences in the data without being overly specific. Therefore, don't cares were introduced into the patterns with the dual goals of retaining specificity and sensitivity of the detector. To achieve this we used the following heuristic: if introducing a don't care significantly changed the feature count over the more specific pattern, then its inclusion is rejected. The motive behind this heuristic was to be very specific on matching characteristics of normal traffic, but allowing the bit-patterns to match as broadly as possible on abnormal traffic. The latter should increase the likelihood that anomalous payloads would skew the feature vector out of the normal regions of the feature space. Future research will focus on automating learning algorithms.

1) *IoT Sensor Detection Performance*

From the traffic analysis in Fig 4 and Table 1, the weather station had the most uniform traffic of the devices tested. Upon inspection, all the packets were found to be approximately 525 bytes long; therefore, a window size of 256 bytes was chosen with a stride length of 64, allowing multiple windows of payload to be evaluated within each packet.

By studying the normal payload contents, two types of 2-gram patterns appeared. The first was letters followed by letters (ASCII codes in the range [65,122], followed by the same). For these 2-grams, we chose the bit-pattern 01XXXXXX 01XXXXXX, which matches pairs of bytes in the range [64,127]; however, because the packets contained no bytes in the range [128,255], we included and extra don't care in the most significant bit of the second octet of the 2-gram pattern, following our feature selection heuristic, to obtain 01XXXXXX X1XXXXXX.

The second pattern of 2-grams we observed were ASCII encoded "=" characters (code 61) followed a variety of letters or numbers. For this pattern, we therefore started with 00111101 0XXXXXXX ("=" followed by any value below 128). To reduce the overly specific first octet, we evaluated the pattern against the normal payloads and found that adding don't cares in bit positions 0, 1, 3 and 4 had little effect on the feature counts. Thus, our final bit-pattern was 001XX1XX 0XXXXXXX.

As a network sensor, the traffic being evaluated by the detector is that leaving the IoT device. Therefore, we simulated attacks representing the use case where a compromised sensor device is attempting to use the network to accomplish its intended misbehavior. We should note that we have no knowledge about the implementation of the IoT device or the server it communicates with; the intent was to simulate common attack scenarios, not ones that would necessarily succeed for this device. The attacks include attempting to establish an encrypted tunnel, an attempted code-injection attack against the upstream server receiving the sensor data, and a worm propagation. The tunnel attack was emulated with traffic captured from a secure shell connection. To emulate a code-injection attack, we replaced the *username="nobody"* string in one of the packets with the string *username= "1; DROP TABLE passwd"*, an attempt to have SQL code embedded in the HTTP GET request be interpreted by the host. It should be noted that we have no knowledge of the server's underlying system, including whether it uses SQL to store a database of usernames. To emulate a worm attempting to propagate to the device, we used the captured payload packets of the infamous Code Red II worm [11].

Fig 5 shows the feature space of the trained 64x64 detection grid for the selected bit-pattern pair. We started with a grid size of 64, with the intention of increasing or decreasing

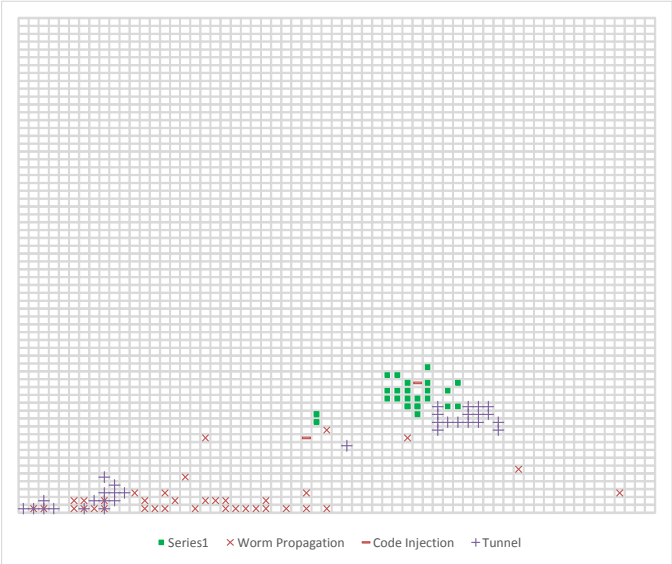


Fig 5. 64x64 feature space for the 2-gram bit-pattern pair 01XXXXXX01XXXXXX, 001XX1XX0XXXXXXXX. For clarity, false-negatives are not plotted.

it as necessary. The filled-in squares represent those regions of the space in which normal traffic occurred. Superimposed on the grid are symbols indicating the regions mapped to by three different simulated attacks: worm propagation, code injection, and tunneling attack. The figure illustrates that one or more windows of payload from each attack were detected by the trained detector. The code injection attack, being the most similar to the original traffic, occupied locations in the feature space that were close to the normal traffic; in fact, one packet occurs in an anomalous cell that falls within the normal cluster. This illustrates our claim about the improved discrimination that comes with using a lookup table detector; using a linear or elliptical discriminant function, for example, would not have detected this part of the attack.

The detection performance of the detector is summarized in Table 2, which shows per window, per packet, and per attack detection rates. While the main goal is to detect and stop the attack, the per-window detection rates are important because they indicate the ability of the detector to discriminate payloads. The per packet detection rates provide a measure of how many packets of the attack would be blocked by the detector. Although blocking only one may be necessary, it is desirable to block as many as possible. Included in the table is the normal test data used to assess false positives.

Table 2: Number of windows and packets detected for sensor attacks tested. In all cases, every packet of all the attacks was detected.

Attack	Number Windows	Windows Detected	Number Packets	Packets Detected	Attack Detected
Worm Propagation	51	51 (100%)	4	4 (100%)	Yes
Code Injection	6	2 (33.3%)	1	1 (100%)	Yes
Tunnel	66	64 (97.0%)	25	25 (100%)	Yes
Normal Testing	12846	1 (0.00%)	2141	1 (0.05%)	n/a

The data shows that not only was every attack detected, but every packet of each attack had at least one window that was detected as anomalous by our approach. Furthermore, the per-window detection accuracies were extremely high for the worm and tunneling attacks. The Code injection attack was expected to be the hardest to detect, given that it is implemented using a single packet that is slightly modified from a normal packet. Yet, despite this small change 33% of the data windows were affected enough by the change to be classified as anomalous. A single false positive occurred on the normal testing data.

The feature space shown in Fig 5 would require only 4096 bits to represent. A seen from the figure, however, the space is relatively sparse, particularly in the dimension represented on the vertical axis of the figure. This prompted us to try reducing that dimension of the detector from 64 to 32, which would cut the overall storage requirement in half. The reduced space is shown in Fig 6. This change caused 2 fewer windows of the tunnel attack to be detected, reducing the per-window detection accuracy to 94.0%; the per packet detection rate remained the same. No other value listed in Table 2 was affected. This illustrates the ability of our detector to balance performance and resources, as the change resulted in a small decrease in the number of windows of payload detected but results in a 50% reduction in detector storage.

It should be noted that when decreasing the size of the space, all normal regions in the larger feature space necessarily map to normal regions in the quantized space; therefore, the false positive rate can only decrease.

2) IoT Actuator Detection Evaluation

Analysis of IoT actuator traffic showed payload lengths between 264 and 500. A window size of 128 bytes was chosen, with a stride length of 32, to allow multiple windows to be processed for all packets. As a network actuator, the traffic being evaluated by the detector is that directed to the IoT device. Therefore, we simulated attacks representing the use case where the attacker is trying to gain access. The attacks include an attempted code injection attack, directory traversal attack, superfluous Unicode attack, worm propagation and an attempt to perform an unauthorized NVRAM restore. The code injection and worm propagation attacks were the

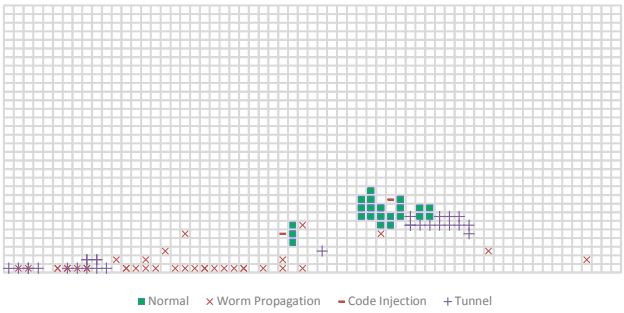


Fig 6. Feature Space from Fig 5, reduced to 64x32.

same as those used in the previous section. The directory traversal attack used a modified URL including the string *GET ../etc/passwd*, (instead of *GET /filename*) emulating an attempt to exploit a misconfigured system to gain access to the file system outside of the root directory of its http server. The superfluous Unicode attack is similar to the directory traversal but attempts to bypass filters looking for the "../" pattern by encoding those three characters in Unicode. The NVRAM restore attack contained an attempt to restore the device settings using a previously collected backup file.

By studying the payload contents, two types of 2-gram patterns appeared. The first was letters followed by letters (ASCII codes in the range [65,122]). For these two grams, we chose the bit-pattern 01XXXXXX 01XXXXXX, which matches pairs of bytes in the range [64,127]. Unlike the IoT sensor, including additional don't cares into this pattern did change the feature counts; therefore, we rejected them based on our heuristic. The second pattern of 2-grams we observed was the presence of a large number of ASCII encoded digits in the range [30,39] followed by a wide range of values. For this pattern, we selected 0011XXXX 0XXXXXXX. Following our heuristic, additional don't cares were not included, as they changed the feature counts dramatically.

Fig 7 shows the feature space of the trained 64x64 detection grid for the selected bit-pattern pair. We again started with a grid size of 64x64, with the intention of increasing or decreasing it as necessary. However, as shown in Table 4, the performance was very good on all attacks except for the superfluous Unicode attack, which was not detected. Increasing the grid size did not improve the performance, so we suspected that the issue was the higher dissimilarity among packets for this traffic (as reported in Section 4A). Therefore, we added a third dimension to the detector, using the bit-pattern 01XXXXXX, which counts 1-grams in the range [64,127].

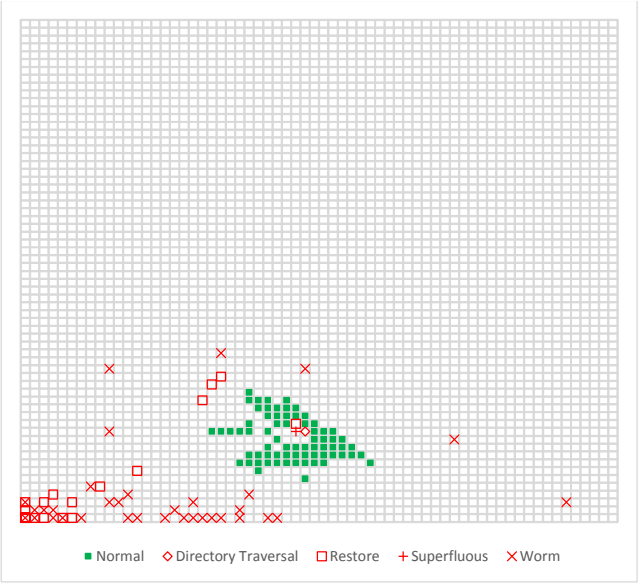


Fig 7. Two-dimensional64x64 feature space for IoT actuator using bit-pattern pair 01XXXXXX01XXXXXX , 0011XXXX0XXXXXXX.

Table 4: Number of windows and packets detected for actuator attacks using 64x64 detector in Fig 7 . One attack was not detected.

Attack	Number Windows	Windows Detected	Number Packets	Packets Detected	Attack Detected
Worm Propagation	111	111 (100%)	4	4 (100%)	Yes
Code Injection	11	2 (18.2%)	1	1 (100%)	Yes
Superfluous	7	0 (0%)	1	0 (0%)	No
Directory Traversal	7	1 (14.3%)	1	1 (100%)	Yes
Restore	173	144 (83.2%)	9	8 (88.9%)	Yes
Normal Testing	7290	0 (0.0%)	1215	0 (0.0%)	n/a

The results of this test are shown in Table 3. Of the four attacks not already detected perfectly by the two-dimensional detector, 3 showed improved performance using the three dimensional detector. Furthermore, all attacks were detected.

3) Discriminatory Ability of Detectors

We evaluated the ability of each detector to discriminate the traffic it was trained on from other traffic. The results are summarized in Table 5. The four detectors correspond to those used in the previous sections, and the three data sets are the collected normal data from the two IoT sensors and the generic web data. Recall that for the IoT sensor, two 2-dimensional detectors using the bit-patterns 01XXXXXX0XXXXXXX and 001XX1XX0XXXXXXX were evaluated. The difference between the detectors is only that one used a 64x64 feature space and the other a reduced 64x32 feature space. Further recall that for the IoT sensor, a 2-dimensional 64x64 feature space using the patterns 01XXXXXX01XXXXXX and 0011XXXX0XXXXXXX was evaluated. A 3-dimensional 64x64x64 space expanded on this by extending to a third dimension using the feature 01XXXXXX.

Table 3: Number of windows and packets detected for actuator attacks using 3-dimensional detector obtained by adding a dimension to the detector used in Table 4. Values showing improvement are in boldface.

Attack	Number Windows	Windows Detected	Number Packets	Packets Detected	Attack Detected
Worm Propagation	111	111 (100%)	4	4 (100%)	Yes
Code Injection	11	**5 (45.5%)**	1	1 (100%)	Yes
Superfluous	7	4 **(57.1%)**	1	**1 (100%)**	**Yes**
Directory Traversal	7	4 (57.1%)	1	1 (100%)	Yes
NVRAM Restore	173	**156 (90.2%)**	9	**9 (100%)**	Yes
Normal Testing	7290	0 (0.0%)	1215	0 (0.0%)	n/a

Table 5: Analysis of discriminatory ability of the four detectors on three data sets. Reported for each combination is the detection rate per window of payload, and per packet of payload.

Detector \ Data Set	IoT Sensor	IoT Control	Generic Web
IoT Sensor 64x64	n/a	Window: 99.9% Packet: 100%	Window: 96.2% Packet: 100%
IoT Sensor 64x32	n/a	Window: 99.9% Packet: 100%	Window: 93.5% Packet: 99.8%
IoT Actuator 64x64	Window: 92.9% Packet: 100%	n/a	Window: 47.7% Packet: 98.4%
IoT Actuator 64x64x64	Window: 92.9% Packet: 100%	n/a	Window: 100% Packet: 100%

The results indicate that both detectors developed for the IoT sensor performed were able to accurately discriminate against the IoT control traffic, detecting 99.9% of the payload windows and every packet as abnormal. The detectors for the IoT actuator, on the other hand, detected slightly fewer windows of the sensor payloads as being anomalous (92.9%), but still detected at least one window in each packet for a perfect packet (and therefore attack) detection rate.

The evaluation on the generic web data can reveal more information on the discriminatory ability of the detectors, given its highly dissimilar nature. Both detectors developed for the IoT sensor could detect 100% of the generic web packets as being anomalous, and both showed very high per-window detection accuracy. The results again show that the smaller 64x32 detector exhibited just slightly lower performance as a tradeoff for the reduction in size. The 2- and 3-dimensional IoT actuator detectors showed much more variability in the ability to discriminate the generic web payload windows, though both had high per packet detection accuracies. In moving from a 2-dimensional feature space to a 3-dimensional space, per-window detection accuracy improved from 47.7% to 100%; per-packet accuracies went from 98.4% to 100%. The 3-dimensional detector developed for the IoT actuator was able to perfectly discriminate the generic web traffic.

V. CONCLUSIONS

We have presented a high-performance ultra-lightweight deep-packet anomaly detection approach that is feasible to run on small IoT devices. The approach uses n-gram bit-patterns to efficiently and flexibly model payloads and allows the n-gram size to vary by dimension. By using a direct representation of the feature space for the discrimination function, the detector can make a fast packet classification decision. The approach can be implemented in hardware or software and has abundant parallelism to exploit for effective implementation. The approach can be deployed in an IoT device's network interface or protocol stack, or can be built into network appliances and firewalls. It can operate in a wide-range of network environments and is highly configurable and scalable.

The results presented have shown that small IoT devices use few and relatively simple protocols, leading to network payloads that are highly similar and therefore amenable to anomaly detection with an extremely low occurrence of false-positives. The detectors have been shown to be highly effective at identifying anomaly packets from a wide-range of attacks, and have an excellent ability to discriminate device-specific traffic from other types of Internet traffic.

REFERENCES

[1] N. Nwanze and D.H. Summerville "Detection of anomalous network packets using lightweight stateless payload inspection". The 33rd IEEE Conf. On Local Computer Networks, 2008.

[2] C. Taylor and J. Alves-Foss, NATE- Network Analysis of Anomalous Traffic Event, A Low Cost Approach, In Proceedings of the NSPW'01, Sept 10-13, 2001, pp. 89-96.

[3] M. Mahoney, Network Traffic Anomaly Detection Based on Packet Bytes, In Proceedings of the 18th ACM Symp. Applied Computing, pp. 346-350, 2003.

[4] C. Kruegel, T. Toth, and E. Kirda, Service Specific Anomaly Detection for Network Instrusion Detection, Symp. On Applied Computing (SAC). ACM Digital Library, Spain, Mar 2002.

[5] K. Wang and S.J. Stolfo. Anomalous Payload-based Network Intrusion Detection, in: Recent Advances in Intrusion Detection (RAID), 2004.

[6] K. Wang and S.J. Stolfo, Anagram, a content anomaly detector resistant to mimicry attack, in: Recent Advances in Intrusion Detection (RAID), 2006.

[7] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, W. Lee, McPAD: A multiple classifer system for accurate payload-based anomaly detection, Computer Networks 53, pp. 864-881, 2009.

[8] D. Ariu, R. Tronci, G. Giancinto, HMMPayl: An intrusion detection system based on Hidden Markov Models, Computer & Security 30, pp. 221-241, 2011.

[9] H. Chen, Y. Chen, and D. Summerville, "A Survey on the Application of FPGAs for Network Infrastructure Security," the IEEE Communications Surveys and Tutorial, Vol. 13, No. 4, 2011.

[10] Lippmann, R. and Haines, J.W. and Fried, D.J. and Korba, J. and Das, K., "The 1999 DARPA off-line intrusion detection evaluation", Computer Networks, vol. 34, no. 4, 2000.

[11] John C. Dolak, The Code Red Worm, Security Essentials Version 1.2e, 2001.