UIDS: Unikernel-based Intrusion Detection System for the Internet of Things

Vittorio Cozzolino, Nikolai Schwellnus and Jörg Ott
Technical University of Munich
cozzolin@in.tum.de, n.schwellnus@tum.de, ott@in.tum.de

Aaron Yi Ding
Delft University of Technology
aaron.ding@tudelft.nl

***Abstract*—The advent of the Internet of Things promises to interconnect all type of devices, including the most common electrical appliances such as ovens and light bulbs. One of the greatest risks of the uncontrolled proliferation of resource constrained devices are the security and privacy implications. Most manufacturers' top priority is getting their product into the market quickly, rather than taking the necessary steps to build security from the start, due to high competitiveness of the field. Moreover, standard security tools are tailored to server-class machines and not directly applicable in the IoT domain. To address these problems, we propose a lightweight, signature-based intrusion detection system for IoT to be able to run on resource-constrained devices. Our prototype is based on the IncludeOS unikernel, ensuring low resource utilization, high modularity, and a minimalist code surface. In particular, we evaluate the performance of our solution on x86 and ARM devices and compare it against Snort, a widely known network intrusion detection system. The experimental results show that our prototype effectively detects all attack patterns while using up to 2-3x less CPU and 8x less RAM than our baseline.**

I. Introduction

With the rising popularity of Internet of Things (IoT), an increasing number of devices are being connected to the Internet. Most of these devices are resource-constrained and security is often regarded as an afterthought. Mirai [3], Qbot [6], and Torii [20] are examples of large-scale network attacks enabled by the proliferation of vulnerable, badly configured smart devices. The distributed and hardly controlled nature of the IoT transformed it into what is today a powerful cyberattack platform. In fact, vulnerabilities have been found in all types of related devices, ranging from cars [24] to light bulbs [10]. Moreover, such devices are oftentimes connected to the network by people having little knowledge about security or privacy concerns. In 2010, a study [8] found that 13% ($\geq$580.000) of the discovered embedded devices still used factory default login credentials. In 2017, Positive Technologies found around 15% of devices with factory-default credentials, which proved to be an exacerbation of the problem [1].

IoT devices are too resource-constrained to employ traditional security tools (virus scanner, etc.), which renders both edge computing and enterprise networks far more exposed to attacks [11]. Additionally, botched or corrupted updates can leave the device in an unstable state, which may be hard to recover from due to the lack of user interfaces. What we need are security tools that are lightweight, modular, and easily deployable. Hence, we aim at laying the foundation of the concept of *composable security* through unikernels with the latter embedding self-contained security functionality quickly deployed on-demand. This work represents a first step in such direction with a signature-based, minimalistic ***U**nikernel **I**ntrusion **D**etection **S**ystem* (UIDS), which can to deliver the same detection capabilities of well-known IDSs (e.g., Snort) while using **2-3x** less CPU and **8x** less RAM. We make two contributions:

- A signature-based IDS, capable of detecting common Denial of Service (DoS), and port scan attacks, which can be deployed on resource-constrained devices with minimal overhead and memory footprint.
- A comprehensive evaluation of our solution with different hardware and datasets against a well-known IDS tool.

II. Related Work

Intrusion detection is a very mature field of research going back to Anderson's "Computer security threat monitoring and surveillance" [2]. The first prototype of a real-time IDS was developed between 1984 and 1988, called the intrusion detection expert system (IDES) [9]. Currently, the rising number of deployed embedded devices and sensors has motivated researchers to explore IDSs that can run on resource-constrained devices. As there are few specific solutions targeting IoT networks, part of the research is focused on adapting and profiling desktop-class tools, such as Snort[1], to be less resource-intensive. One interesting study is [15], where the authors investigated the performance of Snort and Bro[2] on wireless mesh networks (WMNs). It was found that these IDSs are unsuitable as a security solution for WMNs as they are too demanding. To address this problem, they posited a lightweight IDS for WMNs that decreases memory consumption and packet drop rates in such resource-constrained nodes. However, it could only detect a few types of attacks. Similarly, in [14], the authors also argued about the infeasibility of deploying Snort in WMNs and proposed a distributed solution called PRactical Intrusion DEtection in resource constrained wireless

Workshop on Decentralized IoT Systems and Security (DISS) 2020
23 February 2020, San Diego, CA, USA
ISBN 1-891562-64-9
https://dx.doi.org/10.14722/diss.2020.23008
www.ndss-symposium.org

[1] https://www.snort.org/
[2] Today known as Zeek — https://en.wikipedia.org/wiki/Zeek

mesh network (PRIDE). Kyaw et al. [17] compared Snort and Bro IDS running on a Raspberry Pi 2, and showed that a Raspberry Pi 2 has enough resources to run open-source IDSs such as Snort or Bro sufficiently fast to detect DoS attacks and port scans. In addition the authors concluded that Snort performed better than Bro on the Raspberry Pi. For the IoT edge network, researchers have also used Docker containers [12] to deploy cloud-assisted security functionalities, especially for D2D communication [13]. Finally, [21] focused on the present and in-deep analysis of the feasibility of deploying an IDS infrastructure based on Snort and Raspberry PIs. Based on their results, this is possible in small networks; however, and more experiments are needed to better grasp the true limits of the Raspberry Pi. Another example of IDSs for constrained devices is CEPIDS [5], which is a complex event processing (CEP)-based IDS for detecting DoS attacks and port scans. CEPIDS follows a similar architecture to Snort. It uses three components to collect, evaluate and potentially block malicious network traffic. The authors showed that their IDS performs better than Bro and is on-par with Snort. However, the methodology used to evaluate their solution is unclear, as they compare their results with those obtained from [17] by using a different dataset and device.

III. Background

In this section, we briefly introduce two main types of IDSs: signature-based and anomaly-based.

Signature-based IDSs use parameters of known attacks to detect them. Hence, one downside is that new attacks cannot be detected as long as their signatures are not yet known. As such, signature-based IDSs need to receive constant updates to be competitive. In addition, attacks that cannot be easily described with signatures are difficult to detect. On the other hand, a big advantage of signature-based IDSs is that known attacks can be detected fast, accurately, and with fewer *false positives*. This approach, however, depends on the accuracy and quality of the signatures. If the signatures are known, an attacker can craft traffic that is benign but triggers signature-based rules, classifying the traffic as malicious, and as such generates many false positives.

Anomaly-based IDSs require a *normal operation* model against for comparison to the current network traffic. The flowing traffic characteristics are then compared to this baseline, and if an anomaly is found, the IDS will generate an alert.

In the case of network-based IDSs, machine learning is oftentimes used to build a model trained on non-malicious traffic. Incoming packets not fitting the model are classified as abnormal and an alert is generated. An obvious problem with this approach is that no malicious traffic must be present during the model learning phase; otherwise, malicious traffic will be classified as normal and no alerts will be generated.

Anomaly-based approaches can detect attacks that are unknown at the time of deployment. However, they depend heavily on the accuracy of the baseline model and require a potentially long and tedious training process. Hence, one of the main challenges in highly heterogeneous network traffic is building such accurate models.

IV. UIDS

We design UIDS as a signature-based IDS capable of detecting common DoS attacks, such as TCP SYN flood, TCP ACK flood, and UDP flood. Moreover, our signature based approach can detect the most common port scans in three different variations: *one-to-one*, *distributed* and *decoy* scans. We base our prototype on the IncludeOS [4] unikernel which follows the *zero-overhead* principle and is written in C++. UIDS expands on the rudimentary connection tracking capabilities of IncludeOS to classify traffic as suspicious or benign and keeps additional state information regarding possible malicious packets.

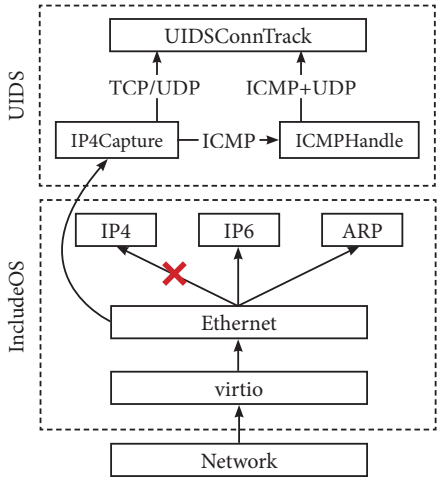

Fig. 1. UIDS packet reception path.

IncludeOS contains a few features that we found handy during the UIDS development. It offers state-keeping for network connections, UDP and ICMP, and a more sophisticated one for TCP. In addition, the IncludeOS modular network stack allows us to easily capture packets on the wire and redirect them to custom modules for additional processing. The network stack of IncludeOS comprises C++ classes for each module of the stack, such as IPv4, IPv6, and TCP which are connected together using delegates and can be rewired at runtime.

We extend IncludeOS as shown in Figure 1 to parse the captured packets and detect attacks. Packets received by the *virtio* device are passed up in the network's stack hierarchy of IncludeOS. After the Ethernet layer, we redirect packets to a custom capture module bypassing the standard one as shown in the figure with a red cross. Subsequently, we forward them to the core of our system: the *UIDSConnTrack* module. ICMP packets of type *destination unreachable* are parsed by *ICMPHandler* to extract the UDP packet that generated the ICMP error message. Afterwards, they are forwarded to the connection tracking module with the augmented data.

Port scans are classified as probes hitting different ports on the same host (vertical scans) or the same port on different hosts (horizontal scans). For this reason, we track suspicious packets on a per-host and per-port basis. UIDS classified packets as probes (suspicious) if at least one of the following criteria was met: *(i)* SYN packet to a closed, filtered or inactive port/host, *(ii)* ACK or FIN packet not belonging to an active connection, *(iii)* invalid packets (NULL/XMAS scan), *(iv)* partial three-way handshake, *(v)* UDP packets generating ICMP unreachable replies, and *(vi)* unanswered UDP packets.

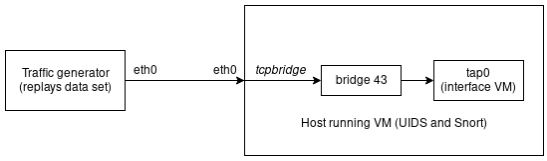

Fig. 2. Setup for traffic replay.

The *UIDSConnTrack* module stores information about packets that satisfied the above rules on a per-host, per-port basis using trackers. The latter are implemented as unordered maps, saving the address of the sending host, in addition to the scan type and time.
DoS detection is implemented similarly to port scan but with a different ruleset. In this case, trackers are simple packet counters, as common practice for such attacks, and store less information about the sender to save on resources.
The data structures tracking suspicious packets are analyzed periodically, and if an attack is recognized during a scan, the system generates an alert in the form of a JSON message. Subsequently, alerts are forwarded to an alert module, which either sends the data using UDP over a second network interface or flushes it to *stdout*.

V. Evaluation Setup

We benchmark UIDS attack detection capabilities against different datasets normally used to evaluate the effectiveness of IDSs. These are often developed to train and test anomaly-based IDSs using machine-learning but could also be used to evaluate signature-based IDS as well. In our evaluation, we use three publicly available datasets containing DoS attacks and port scans in a packet-based format. One of the most widely used dataset is the 1998 DARPA Intrusion Detection Evaluation Dataset[3]; however it is very old, and therefore did not contain traffic one would likely see today. In addition, several researchers have shown different flaws in this dataset [18], [19]. Hence, we instead used TRAbID [16] and CICIDS 2017 [22]. In addition to existing datasets, we use a small-scale testbed of our design to stress-test UIDS. Both port scans and DoS flooding attacks are used to stress-test our implementation and evaluate the accuracy of alerts raised by UIDS and Snort.

Traffic replay. UIDS and Snort run on KVM with QEMU using bridge networking to expose an interface to replay traffic to. IncludeOS comes with a deployment tool for KVM and QEMU, which allowed to configure this bridge networking. We replay the dataset network traffic through an Ethernet interface on the *traffic-generator* node and send it to the device hosting UIDS and Snort via the incoming network interface. On the receiving host, traffic is forwarded to the bridge interface using the tool *tcpbridge* included in the *tcpreplay* tool suit. Finally, the bridge interface (*bridge43*) is connected to the virtual machines (VMs). Figure 2 illustrates the complete network flow.
CICIDS2017 traffic is split into port scan and DoS to dramatically reduce the evaluation period from more than

[3]https://www.ll.mit.edu/r-d/datasets/1998-darpa-intrusion-detection-evaluation-dataset

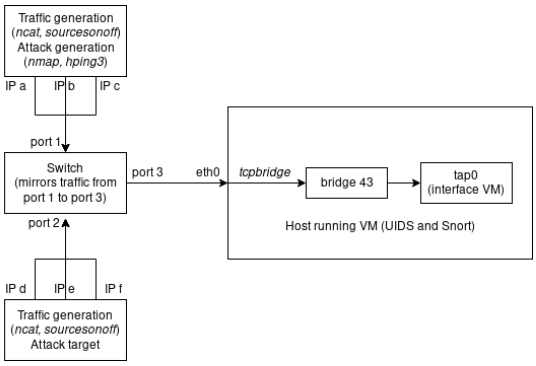

Fig. 3. Setup for live traffic.

9h to 1.5. However, such a procedure might introduce false-positives in the first few minutes of the new traffic since some connections might have been established directly before the split. Nevertheless, these false-positives were easily filtered out from the analysis (if they occurred).
The TRAbID dataset provides two traffic captures for port scan and DoS. We use the *probe_known_attacks* capture for the evaluation of the port scan detection accuracy. It contains seven port scans originating from different machines, but all directed to the same host. However, the amount of attack traffic is very small compared to the background traffic ($\leq$0.15%). Unfortunately, for this dataset, the DoS traffic capture is not currently publicly available, so we could not use it in our evaluation.

Live traffic. Besides testing our solution against existing datasets, we also generate our own network traffic traces. For this purpose, we connect two hosts with a switch supporting port-mirroring. A host running UIDS and/or Snort is connected to the mirrored port, to receive all traffic generated between the hosts. Figure 3 illustrates the described setup.
To generate the traces, we uses the tool *sourcesonoff*[4], which outputs realistic Internet-like traffic using statistical models, detailed in [23]. Moreover, we develop a simple script using *netcat* to simulate an arbitrary number of concurrent TCP connections. Finally, the attack traffic is injected in the testbed using *nmap* for the port scan and *hping* for DoS traffic.

VI. Results

In this section, we present the results obtained by testing UIDS against the datasets described before while using Snort as the baseline. We focus primarily on CPU and RAM utilization to evaluate the compatibility of UIDS with resource-constrained devices. In terms of the memory footprint, UIDS weights only $\approx$2.3MB and boots in $\approx$200 ms on a non-optimized version of KVM (we did not use Solo5[5]). We run our tests on two different hardware platforms: a laptop equipped with an Intel i7-4710@2.5GHz (LAP) and a Raspberry Pi 3B+ with an A53 ARMv8@1.4GHz (RPI). On the former, both IDSs run virtualized on top of KVM. On the latter, due to the lack of support for ARM, IncludeOS can not be directly virtualized using KVM. Instead, we emulate the x86

[4]http://www.recherche.enac.fr/~avaret/sourcesonoff
[5]https://github.com/Solo5/solo5

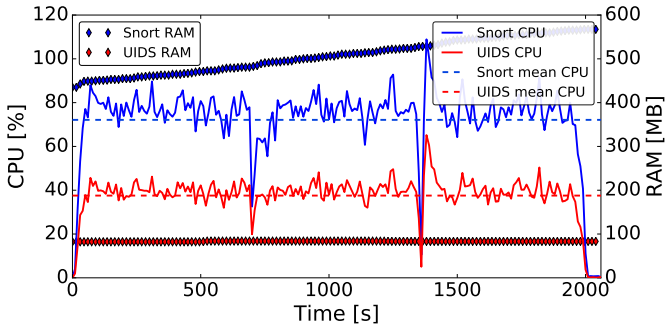

Fig. 4. UIDS vs. Snort — Port scan (TRAbID, LAP).

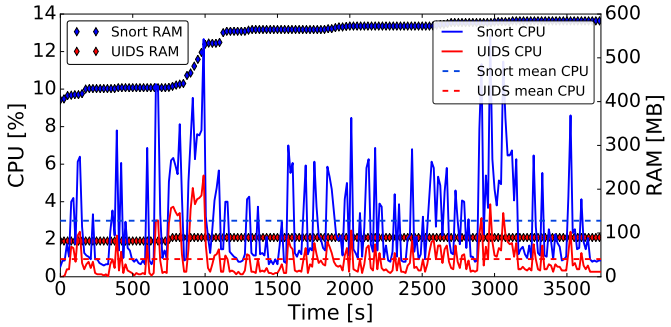

Fig. 5. UIDS vs. Snort — Port scans (CICIDS2017, LAP).

architecture on top of ARM using QEMU. Therefore, UIDS suffers a considerable performance penalty due to the additional emulation overhead which affected the results as well. Conversely, Snort runs baremetal, which gave it a considerable advantage in terms of performance. Hence, on the RPI we test UIDS in what could be seen as the worst case scenario.

A. TRAbID

Traffic from the TRAbID dataset is replayed over ≈33 min with an average speed of 33.68Mbps. Figure 4 shows the resource consumption on our laptop, which are correlated to the number of packets sent. Two dips in the graph at ≈600 and ≈1350 s are caused by a sharp drop in the number of packets generated by the dataset, moving from an average of ≥14000 to ≤1500 packets per second (pps).

The port scan traffic is not visible in the graph as it only represented a very small part of the overall traffic. Both UIDS and Snort equivalently detect 85% of all port scans contained in the dataset covering UDP, SYN, NULL, TCP connect, FIN, and XMAS scans. The ACK scan is not detected because it is not supported in UIDS and no rules to detect it are added to Snort. On the laptop, memory consumption for UIDS is just under 100 MB while it hovers around 400-600 MB for Snort. Additionally, UIDS CPU utilization is ≈40% while it is up to ≈80% for Snort. On the Raspberry Pi, both UIDS and Snort use, on average, the same amount of CPU. Regarding the memory allocation, UIDS is less demanding than Snort, requiring ≈50MB of RAM. Figure 7 shows the results in detail. Our solution is definitely penalized with this setup, as it is running on a twice virtualized stack, which clearly affects the performance.

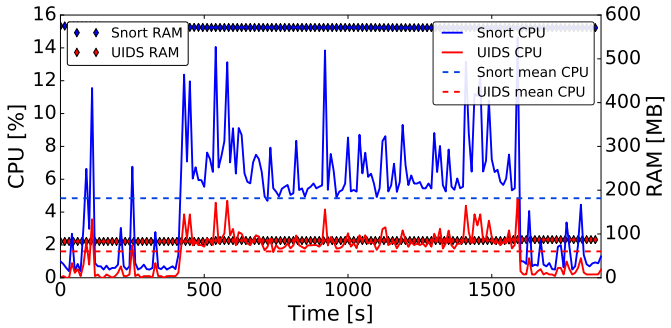

Fig. 6. UIDS vs. Snort — DoS (CICIDS2017, LAP).

TABLE I. CICIDS2017 — PORT SCANS AND RELATIVE DETECTION.

Scan	From (min)	Until (min)	UIDS	Snort
TCP SYN scan	11	13	yes	yes
TCP Connect scan	14	16	yes	yes
TCP FIN scan	17	19	no	no
TCP XMAS scan	20	22	no	no
TCP NULL scan	23	25	no	no
ICMP Ping scan	26	27	no	yes
TCP version scan	28	30	yes	yes
UDP scan	31	32	yes	yes
IP-protocol scan	33	35	no	no
TCP ACK scan	36	38	no	no
TCP window scan	39	41	yes	yes

B. CICIDS2017

The results for this dataset are divided for port scan and DoS attacks, and are described as follows.

Port scan. Port scans are executed during specific time windows, as specified in Table I. We notice that both UIDS and Snort detect most TCP/UDP-based scans contained in the dataset. As a side note, the CICIDS dataset supposedly contained FIN-, NULL-, and XMAS-scans, but could not find any evidence of such scans in the downloaded dataset. In fact, no packet without TCP flags set (NULL scan) or with URG, PSH and FIN flags set (XMAS scan) could be found using various tools. Therefore, neither UIDS nor Snort raise any alert regarding such attacks. The only difference in port scan detection between Snort and UIDS is the ICMP ping scan, which is not currently implemented, and therefore, is not detected by UIDS. The TCP version and window scans have similar characteristics as TCP SYN or connect scans, and are detected by both IDSs but classified as TCP SYN scans.

Snort and UIDS generated false positive alerts for FIN scans in the first 120 seconds of the dataset. As mentioned in Section V, this is due to the splitting of the dataset which led both systems to see finalization packets that belonged to connections lost during the splitting.

Figure 5 shows the resource utilization. Overall, the CPU usage is low for both IDSs because the packet rates in the CICIDS dataset are smaller than those in TRAbID, averaging around ≈330 *pps* and approximately 1 Mbps. Memory consumption is definitely higher for Snort, with 400-600MB, against UIDS, with less than 100MB (**4-6x** lower). A spike in memory usage can be observed after the first port scan is executed at the 11-12 min mark. Interestingly, this spike is modest for UIDS with a variation of ≤10MB but substantial for Snort with ≥130MB.

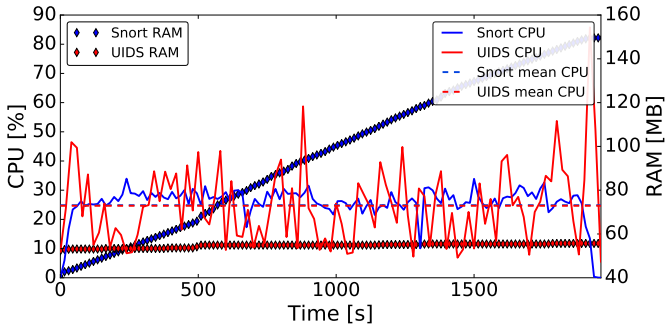

Fig. 7. UIDS vs. Snort — Port scan (TRAbID, RPI).

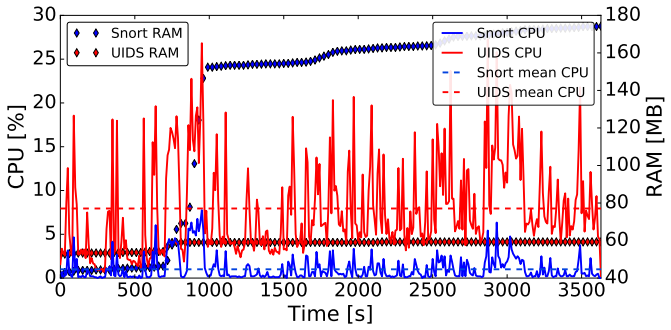

Fig. 8. UIDS vs. Snort — Port scan (CICIDS2017, RPI).

Flood-based DoS. Traffic from the CICIDS2017 dataset contains a DoS attack generated with the open-source tool Low Orbit Ion Cannon (LOIT)[6]. This attack is active from 15:56 until 16:16 and is contained in the second traffic slice generated by splitting the dataset. Both Snort and UIDS emit alerts correctly during the active phase of the attack. Figure 6 clearly shows the beginning and end of the DoS attack in relation to the number of CPU resources used. As foreseen, memory consumption remain stable and marginal during the attack since the very little state information needed storing to detect flood-based DoS attacks. Also, for this benchmark, UIDS proves to be extremely lightweight compared to Snort. In fact, it allocates ≈**4x** less memory than Snort during the attack peak and on average **3x** less CPU.

Figures 8 and 9 show CPU and RAM usage for the Raspberry Pi for CICIDS2017 port scan and DoS traffic, respectively. While UIDS CPU usage is **5-6x** higher compared with Snort, memory allocation is surprisingly reduced. Hence, considering that we are running in an emulated x86 environment, UIDS can handle moderately fast traffic (up to ≈34Mbps) for the CICIDS2017 dataset and reliably detect the same attacks as that in the case of running on more powerful hardware.

C. Results with custom testbed

We use our own testbed to evaluate the performance of UIDS under heavy loads, as described in section V. To simulate a fully utilized link, we open multiple TCP sessions with *netcat* and transmit random traffic as fast as possible through all concurrent connections. As the TCP protocol performs load

[6]https://sourceforge.net/projects/loic/

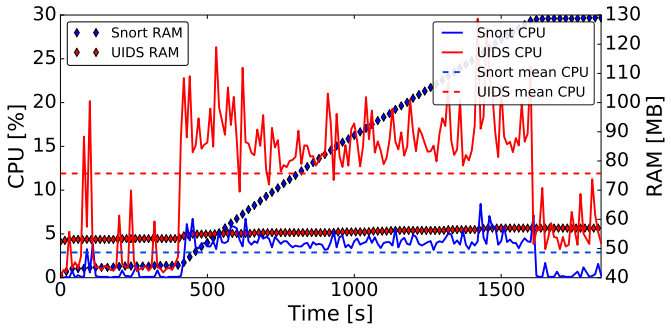

Fig. 9. UIDS vs. Snort — DoS (CICIDS2017, RPI).

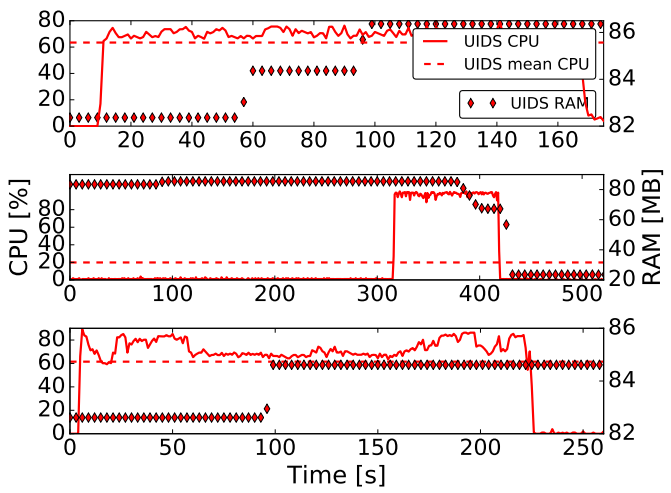

Fig. 10. UIDS stress-test (LAP) — Saturated 1Gbps link (top). Background traffic plus attacks (middle). 1Gbps, 5000 concurrent TCP connections (bottom).

balancing for us, we do not need any additional configuration steps. Figure 10(top) shows the resource consumption of UIDS dealing with a saturated 1Gbps link with an increasing number of concurrent connections, specifically, 1000 connections until 60 s, 5000 until 90 s and 10000 until 165 s. A step-up in memory consumption can be observed when the number of concurrent connections increased. This is expected, as UIDS needs to keep track of these extra connections.
Because the link capacity is fully utilized, some traffic is lost, and therefore, not available for our connection tracking algorithm. This is problematic because the port scan detector relied on accurate connection tracking. Moreover, we observe several false positives for TCP no-reply (i.e., lost answers to SYN packets) as well those for FIN scans.

In a second experiment, we evaluate whether UIDS could detect attacks in a realistic background traffic and if UIDS can cope with a large-scale DoS attack using 1Gbps traffic. Figure 10(mid) shows resource consumption during the background traffic, port scans, and a large DoS attack using the ICMP flood. The CPU usage is very small when handling traffic generated by the tool *sourcesonoff*, while memory usage is comparable to the results obtained for the other datasets.
Four different port scans are executed during the first 300 s of the second experiment: TCP SYN, TCP XMAS, TCP NULL and a UDP scan. All four scans are detected and no false alerts

are generated.
UIDS behaves as expected during the ICMP flood between 310 and 420 s, as shown in Figure 10(mid). However, the ICMP flood with ≈120,000 *pps* and close to 1Gbps rate is strong enough to effectively disable our traffic-generating host. Consequently, we see a reduction in the allocated memory as many connections timed out and no new ones are generated; thus, fewer connections need tracking.

Finally, we conduct a third experiment to evaluate the port scan detection capability under high load, as shown in Figure 10(bottom). We use the same four scans as in the previous experiment and a saturated 1Gbps link with 5000 concurrent TCP connections. In this case, all types of scans are detected including TCP SYN which got linked to false-positive alerts, and as such, are not accurate.

VII. Conclusions and Future Work

This paper presents UIDS: our first prototype of signature-based unikernel IDS for the IoT. UIDS is implemented from scratch in C++ and is based on IncludeOS. We evaluated our solution on both a mid-range laptop and a Raspberry PI and compared the results against Snort on two datasets. From our experiments, UIDS required **2-3x** fewer CPU resource and up to **8x** times less memory than Snort without penalizing any detection capability. We consider these results very promising as our main goal was to build a lightweight modular solution, with reduced hardware resource demand.
Despite this being our first prototype, UIDS showed great potential by delivering better resource efficiency, isolation, and a small memory footprint without sacrificing on the security aspects. Its modularity enabled easier code updates and opened the door to composable, on-demand security with unikernels. In our future work, we plan on exploring the tradeoffs of extending UIDS to anomaly-based detectors and simplifying its setup by using the IncludeOS configuration language NaCl and integrating UIDS with our edge-cloud deployment and chaining framework [7] to orchestrate a network of UIDS and fully exploit its modularity for service composition.

References

[1] "Positive technologies - learn and secure : Practical ways to misuse a router," http://blog.ptsecurity.com/2017/06/practical-ways-to-misuse-router.html, accessed: 2019-07-17.

[2] J. P. Anderson, "Computer security threat monitoring and surveillance," 1980.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 1093–1110.

[4] A. Bratterud, A. Walla, H. Haugerud, P. E. Engelstad, and K. Begnum, "Includeos: A minimal, resource efficient unikernel for cloud services," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov 2015, pp. 250–257.

[5] A. Cardoso, R. Fernandes Lopes, A. Teles, and F. Benedito Veras Magalhaes, "Poster abstract: Real-time DDoS detection based on complex event processing for IoT," 04 2018, pp. 273–274.

[6] G. Cluley, "Mutating Qbot worm infects over 54,000 PCs at organizations worldwide," *Tripwire, Tripwire*, 2016.

[7] V. Cozzolino, A. Y. Ding, and J. Ott, "Edge chaining framework for black ice road fingerprinting," in *Proceedings of the 2Nd International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys '19. New York, NY, USA: ACM, 2019, pp. 42–47. [Online]. Available: http://doi.acm.org/10.1145/3301418.3313944

[8] A. Cui and S. J. Stolfo, "A quantitative analysis of the insecurity of embedded network devices: Results of a wide-area scan," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 97–106. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920276

[9] A. K. D and S. Venugopalan, "Intrusion detection systems: A review," *International Journal of Advanced Research in Computer Science*, vol. 8, 10 2017.

[10] N. Dhanjani, "Hacking lightbulbs: Security evaluation of the philips hue personal wireless lighting system," *Internet of Things Security Evaluation Series*, 2013.

[11] B. Duncan, A. Happe, and A. Bratterud, "Enterprise IoT security and scalability: how unikernels can improve the status Quo," in *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2016, pp. 292–297.

[12] I. Hafeez, A. Y. Ding, L. Suomalainen, A. Kirichenko, and S. Tarkoma, "Securebox: Toward safer and smarter iot networks," in *Proceedings of ACM Workshop on Cloud-Assisted Networking*, ser. CoNEXT CAN '16. ACM, 2016.

[13] I. Hafeez, A. Y. Ding, and S. Tarkoma, "Ioturva: Securing device-to-device (d2d) communication in iot networks," in *Proceedings of the 12th ACM MobiCom Workshop on Challenged Networks*, ser. MobiCom CHANTS '17. New York, NY, USA: ACM, 2017, pp. 1–6.

[14] A. Hassanzadeh, Z. Xu, R. Stoleru, G. Gu, and M. Polychronakis, "Pride: Practical intrusion detection in resource constrained wireless mesh networks," in *International Conference on Information and Communications Security*. Springer, 2013, pp. 213–228.

[15] F. Hugelshofer, P. Smith, D. Hutchison, and N. Race, "Openlids: a lightweight intrusion detection system for Wireless Mesh Networks," 01 2009, pp. 309–320.

[16] E. K. Viegas, A. Santin, and L. Soares de Oliveira, "Toward a reliable anomaly-based intrusion detection in real-world environments," *Computer Networks*, vol. 127, 08 2017.

[17] A. K. Kyaw, Yuzhu Chen, and J. Joseph, "Pi-ids: evaluation of open-source intrusion detection systems on Raspberry Pi 2," in *2015 Second International Conference on Information Security and Cyber Forensics (InfoSec)*, Nov 2015, pp. 165–170.

[18] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," in *Recent Advances in Intrusion Detection*, G. Vigna, C. Kruegel, and E. Jonsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 220–237.

[19] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, Nov. 2000. [Online]. Available: http://doi.acm.org.eaccess.ub.tum.de/10.1145/382912.382923

[20] C. Osborne, "Meet Torii, a new IoT botnet far more sophisticated than Mirai variants," 2018.

[21] A. Sforzin, F. G. Mármol, M. Conti, and J.-M. Bohli, "Rpids: Raspberry Pi IDS — a fruitful intrusion detection system for IoT," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. IEEE, 2016, pp. 440–448.

[22] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," 01 2018, pp. 108–116.

[23] A. Varet and N. Larrieu, "How to generate realistic network traffic?" in *2014 IEEE 38th Annual Computer Software and Applications Conference*, July 2014, pp. 299–304.

[24] A. Wright, "Hacking cars," *Commun. ACM*, vol. 54, no. 11, pp. 18–19, Nov. 2011. [Online]. Available: http://doi.acm.org/10.1145/2018396.2018403