DeadBolt: Securing IoT Deployments

Ronny Ko
Harvard University
hrko@g.harvard.edu

James Mickens
Harvard University
mickens@g.harvard.edu

ABSTRACT

In this paper, we introduce DeadBolt, a new security framework for managing IoT network access. DeadBolt hides all of the devices in an IoT deployment behind an access point that implements deny-by-default policies for both incoming and outgoing traffic. The DeadBolt AP also forces high-end IoT devices to use remote attestation to gain network access; attestation allows the devices to prove that they run up-to-date, trusted software. For lightweight IoT devices which lack the ability to attest, the DeadBolt AP uses virtual drivers (essentially, security-focused virtual network functions) to protect lightweight device traffic. For example, a virtual driver might provide network intrusion detection, or encrypt device traffic that is natively cleartext. Using these techniques, and several others, DeadBolt can prevent realistic attacks while imposing only modest performance costs.

CCS CONCEPTS

• **Security and privacy → Systems security**; • **Computer systems organization → Embedded and cyber-physical systems**; • **Networks** → *Mobile and wireless security*;

KEYWORDS

Internet of Things, IoT, security, remote attestation

ACM Reference format:
Ronny Ko and James Mickens. 2018. DeadBolt: Securing IoT Deployments. In *Proceedings of Applied Networking Research Workshop, Montreal, QC, Canada, July 16, 2018 (ANRW '18),* 8 pages.
https://doi.org/10.1145/3232755.3232774

ANRW '18, July 16, 2018, Montreal, QC, Canada

ACM ISBN 978-1-4503-5585-8/18/07…$15.00
https://doi.org/10.1145/3232755.3232774

1 INTRODUCTION

The Internet of Things (IoT) enables sensing and actuating in a variety of new scenarios. Inside a home, appliances like refrigerators, and regulation devices like thermostats, can be manipulated via explicit network protocols [5, 35, 49]. In industrial settings, heavy machinery can send status reports to a centralized controller [29]; remote sensors can monitor the status of potentially dangerous chemical reactions [42].

Unfortunately, IoT deployments create security risks, since network-facing IoT devices are exposed to potentially malicious traffic. Remote exploits are an old threat; traditional servers have faced these problems for decades. However, the nascent IoT ecosystem has prioritized functionality over security, and ignored many of the hard-won insights from traditional network security. The result has been a variety of well-publicized security failures. Some of these failures have been small in scope, like the "smart light bulbs" which used an insecure wireless protocol, allowing an eavesdropper to extract Wi-Fi passwords [19]. Other security problems have been much more devastating. For example, in September 2016, hundreds of thousands of hacked cameras and digital video recorders joined the Mirai botnet, participating in a denial-of-service attack that generated a record-breaking 1.1 terabits per second of traffic [20].

In this paper, we introduce DeadBolt, a new security framework for deploying IoT devices. DeadBolt has two goals. First, DeadBolt tries to minimize the threat surface that IoT devices expose to a remote attacker. Second, DeadBolt tries to stop a potentially subverted device from subverting other devices in the IoT deployment. To achieve these goals, DeadBolt uses three techniques.

- First, DeadBolt forces all device traffic to go through a trusted gateway which implements virtual device drivers; these drivers allow extra security measures to be "bolted onto" devices whose vendors no longer exist, or do not provide frequent security patches.
- Second, by forcing devices to remotely attest their software, DeadBolt implements device quarantine: once the DeadBolt gateway detects that a device runs insecure software, that device loses network access, and cannot regain it until the device willingly patches its software, or the user installs a virtual driver on the gateway which enables the device to be used securely.

- Finally, DeadBolt forces each device to protect against remote exploits. In particular, devices must continually re-randomize device-side code, and apply security patches as soon as possible to a hot backup of the device's software stack that can be brought online immediately.

The primary contribution of this paper is a demonstration that IoT deployments are not intrinsically doomed to insecurity. Some of DeadBolt's security techniques (e.g., remote attestation) are well-known, but conventional wisdom has suggested that these techniques are too expensive to run in IoT scenarios. We demonstrate that this conventional wisdom is incorrect; furthermore, we introduce new security techniques like stackable virtual drivers. Using experiments with real IoT devices, we show that DeadBolt can prevent real exploits with minimal decreases in application-level performance. Given these promising results, we hope that IoT vendors will implement DeadBolt-style techniques to protect IoT deployments.

2 BACKGROUND

As shown in Table 1, IoT devices possess a wide range of computational abilities. This heterogeneity is a major reason why securing IoT deployments is hard: many devices are not configurable, or lack the computational resources needed to implement state-of-the-art security techniques. In this paper, we make a fundamental claim: at a minimum, a secure IoT deployment requires a network gateway that is powerful enough to act as a security monitor. Even if all other devices are computationally weak, a security monitor can use virtual drivers (§4) and firewall rules to protect the overall deployment. At first glance, adding such a security monitor might seem financially prohibitive. However, we believe that adding a security monitor using (say) a $90 Minnowboard is financially prudent, given the costs of suffering from an exploited IoT deployment.

We define a heavyweight device as one that possesses a TPM chip, has updatable firmware and software, and has enough computational resources to implement schemes that enforce control flow integrity (§4). All other devices are lightweight.

3 THREAT MODEL

DeadBolt considers an IoT device to be trusted if (1) its software is up-to-date, protects against control flow exploits, and uses TLS to exchange network data, or (2) the device's network interactions are mediated by software that is up-to-date, protects against control flow exploits, and uses TLS to exchange network data. DeadBolt ensures that, for each device in an IoT deployment, the device can only use the network if condition (1) or (2) holds. By ensuring this, DeadBolt protects IoT deployments from network-based adversaries.

In general, DeadBolt cannot protect against vulnerabilities that arise from software misconfiguration or poor software design. For example, DeadBolt cannot prevent a user from willingly exposing private sensor data to an untrustworthy (but TLS-authenticated) endpoint. However, for a TPM-enabled device, each attestation message contains the hashes of local configuration files; in some cases, a DeadBolt AP can use those hashes to detect insecure device configurations.

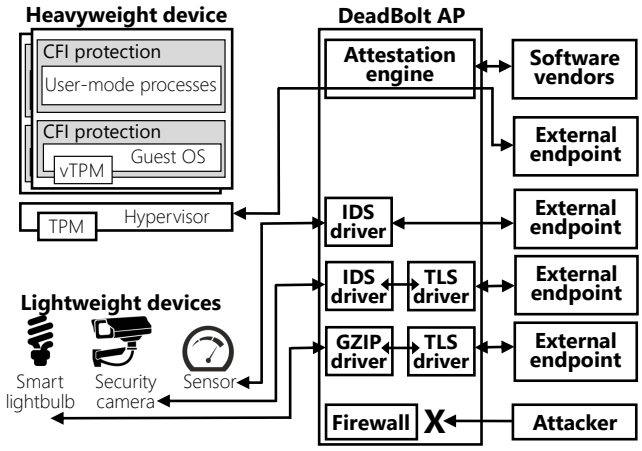

Figure 1: The DeadBolt architecture.

For TPM-enabled devices, DeadBolt trusts the TPM hardware. On the AP, DeadBolt trusts all hardware, the OS, and all of the DeadBolt software. The AP software validates client-provided attestations; the software also manipulates packet forwarding rules to quarantine insecure devices, and redirect traffic to virtual drivers. The AP's attestation daemon communicates with software vendors or third-party update services using TLS; DeadBolt trusts the TLS certificate infrastructure.

4 DESIGN

Figure 1 shows DeadBolt's architecture. Individual devices connect to the outside world via the DeadBolt AP. The AP's firewall is deny-by-default. Arbitrary incoming connections are blocked, and traffic from internal IoT devices is blocked unless the AP knows the devices to be safe; a device is safe if it can remotely attest a known-safe software stack, or if the AP possesses virtual drivers which can secure the device's otherwise-insecure traffic. Optionally, the AP can use manufacturer usage descriptions [28] to further restrict the external hosts that a device is allowed to contact.

Virtual drivers: An individual driver performs a single function, like scanning traffic for malicious packets, or implementing a TLS [12] tunnel. The drivers for a particular device can stack, e.g, to create a virtual IoT device that both rejects malicious packets and encrypts end-to-end communication with TLS. Virtual drivers (which can be written by third parties) provide a safe, easy way to expose otherwise-insecure lightweight devices to the external world.

Name	CPU	RAM	Onboard TPM?	Price
Conga IA4	4x1.04 GHz x86 (64 bit)	4 GB	Yes	$455
Intel STK2m364CC	2x900 MHz x86 (64 bit)	4 GB	Yes	$259
Minnowboard Turbot	2x1.46 GHz x86 (64 bit)	2 GB	Yes (fTPM)	$90
Raspberry Pi 3	4x1.2 GHz ARMv8-A (64 bit)	1 GB	No	$35
C.H.I.P.	1 GHz ARMv7-R (32 bit)	512 MB RAM	No	$9
LinkIt One	260 MHz ARM7EJ-S (32 bit)	4 MB RAM	No	$59
Flora	16 MHz Atmel AVR (8 bit)	2.5K RAM	No	$15
Arduino Uno	20 MHz Atmel AVR (8 bit)	2 KB RAM	No	$25
Flir FX camera	N/A	N/A	No	$140
Garadget garage door opener	N/A	N/A	No	$89
Monnit temperature sensor	N/A	N/A	No	$49
Sabre motion sensor	N/A	N/A	No	$30

Table 1: The computational resources available to various IoT devices. The first set of devices are powerful enough to be heavyweight DeadBolt participants. DeadBolt hides the latter two sets of devices behind virtual drivers.

Virtual drivers are useful for heavyweight devices as well, since those devices may also be afflicted by a lack of vendor-supplied firmware or software updates [38]. Also note that virtual drivers can implement non-security functionality like data compression.

A virtual driver may change device traffic in ways that require a remote endpoint to adjust its behavior. For example, DeadBolt can use a TLS driver to encrypt the traffic generated by a device that natively speaks in cleartext; however, the remote endpoint of the connection must also speak TLS to usefully interact with the virtualized device. Fortunately, updating server-side software is easy, at least in principle. In contrast, directly updating device-side software is often hard, e.g., because the device implements all functionality in hardware, or because the original software vendor no longer supports the device. Virtual drivers ease the burden of securing these kinds of devices.

Remote attestation with device quarantine: When a heavyweight device attests to a DeadBolt AP, the device includes a list of all the device-side software components. The AP periodically contacts the vendors (or a third-party security service akin to Google's Safe Browsing API [21]), checking for patches or deprecation warnings for the device's software. For example, if the device runs a Linux OS like Debian, the AP can use standard `apt-get` interfaces to determine the trustworthiness of a device's attested stack; `apt-get` interfaces use TLS and package signing [27] to allow the AP to verify the authenticity of purported software updates. If the AP determines that a device's stack has become insecure, the AP notifies the device. If the device does not update itself during a grace period, the AP revokes the device's network access. To regain access, the device must query the AP to determine which blacklisted software should be deleted, or which required software should be installed, or which preexisting software should be updated. After deleting the blacklisted items, the device essentially treats the AP as a local software repository, downloading new or updated software from it. Afterwards, the device must reboot and re-attest to gain network access.

Protecting dynamic program state: Buffer overflows [8] and ROP attacks [6, 36] are common exploit vectors for IoT devices. The DeadBolt AP rejects the remote attestation of a heavyweight device if the device does not use defensive techniques against these exploits.

- To avoid traditional buffer overflow attacks, each user-level device process must employ stack canaries [11] and no-execute bits for non-code pages [16]. These protections are standard on desktop and server platforms, and should be standard on IoT devices too.
- Network-facing user-level processes must also protect against advanced control flow attacks like ROP. In our DeadBolt prototype, an AP requires heavyweight devices to use Shuffler [46], an execution platform that continually randomizes the locations of code pages during runtime. Frequent shuffling makes ROP attacks hard—if gadget discovery and exploitation do not finish before the next reshuffle, the attacker loses all work, and must restart the attack from scratch.

Protection of dynamic program state compliments the protection of static code that is provided by attestation.

Fast patching: A device which reboots frequently can aggressively discard subverted dynamic state. Rebooting is also necessary after the installation of many OS patches (and some kinds of user-mode patches). However, rebooting leads to application downtime, making frequent reboots and aggressive patching unattractive.

To minimize the application-level downtime that is incurred by reboots, a heavyweight DeadBolt device uses VMs to store the majority of the device's code and data. In the steady state, a device's hypervisor only executes one VM. However, when that VM's state needs to be refreshed (e.g.,

to install a patch in the guest OS), the hypervisor launches a new VM in the background. As the foreground VM continues to execute and interact with external network clients, the background VM updates itself, e.g., by calling `apt-get update` to install new patches. Once the background VM has finished its boot, the foreground VM uses CRIU [9] to snapshot user-level applications like web servers. The foreground VM writes the snapshots to a disk partition that is shared with the hypervisor. The hypervisor then kills the foreground VM, and allows the new VM to read the CRIU snapshots and resurrect the associated user-level applications.

DeadBolt binds the new VM to the old VM's IP address; CRIU also uses Linux's `TCP_REPAIR` facility [24] to snapshot and restore TCP send/receive queues and other connection state. Thus, DeadBolt allows network-facing applications to preserve open connections across a VM switch. Generally, a device will not snapshot all user-level processes, but only a select few that are network-facing. Other processes, like logging daemons, will restart naturally as the new VM boots.

When a device uses VM-based patching, the device must use a modified attestation protocol. After a reboot of the physical device (but before any VMs have launched), the hypervisor must attest to the DeadBolt AP to establish the trustworthiness of the low-level device software. Later, after a new VM has launched, the new VM must attest to the DeadBolt AP as well. The resulting attestation protocol is similar to the classic vTPM protocol [2]. The basic idea is that the hypervisor extends the virtual TPM's public key into the hypervisor's physical `PCR[10]` register. In this fashion, the hypervisor vouches for the authenticity of the key, allowing the DeadBolt AP to construct a chain of trust that starts in the device's physical hardware, goes up through the hypervisor's software, and continues through the VM's virtual TPM and the VM's software stack.

Note that, after a hypervisor has rebooted the physical hardware and attested, subsequent VM-level attestations are not in the critical path for VM switching. The reason is that the vTPM protocol allows a VM that is booting in the background to attest before it moves to the foreground and starts handling live traffic.

The CRIU snapshot for a shuffled process contains the randomized code layout in virtual memory. At first glance, this might seem to prevent a VM from correctly attesting the associated binary—an attested binary is represented by the hash of its code, but Shuffler randomizes code locations at the granularity of functions. However, when CRIU restores a snapshotted process, CRIU must read the original, unshuffled binary to extract various pieces of ELF metadata. The read of the original binary is captured by the attestation log, allowing a DeadBolt AP to verify the identity of the shuffled process. Note that, on the device-side, a mismatch between the code in the snapshotted process and the original binary will cause the CRIU resurrection to fail.

Stage	Hypervisor	VM
WPA2 connection	1878 ms	2051 ms
IP address registration	172 ms	197 ms
Remote attestation	854 ms	641 ms
Total	**2904 ms**	**2889 ms**

Table 2: Network access latencies. The WPA2 connection used EAP-TTLS.

5 PROTOTYPE IMPLEMENTATION

We used a Minnowboard Turbot [31] as the DeadBolt AP. The AP ran Xen 4.10-unstable [47], patched to support vTPMs; inside a VM, the guest OS was Ubuntu Server 17.04. We used HostAP [30], WPA Supplicant [30], and FreeRadius [14] to allow the Minnowboard to act as a WPA2 access point.

Both sides of DeadBolt's remote attestation protocol were built using the IBM TSS library [17] and the IBM ACS library [18]. On the AP, we had to modify 10 lines of DNSMasq [25] source code to integrate remote attestation with standard AP functionality involving DHCP and DNS. Heavyweight IoT devices used TPM-enabled GRUB2 [15] to measure the kernel image and the boot-time ramdisk. Afterwards, devices used Linux's Integrity Measurement Architecture [37] kernel module to extend `PCR[10]`. Devices used Shuffler [46] to periodically rerandomize a process's code offsets, and CRIU [9] to snapshot in-memory state.

6 EVALUATION

In this section, we demonstrate that DeadBolt is a practical, efficient framework for increasing the security of IoT deployments. All experiments used a Minnowboard Turbot as the DeadBolt AP. The AP connected to IoT devices using a wireless connection, and to external test machines using a wired Ethernet connection; the AP and the external test machines were on the same LAN.

6.1 Performance

Network access latencies: To communicate with the outside Internet, a DeadBolt VM must associate with the AP, receive an IP address, and then attest its software stack to the AP. Table 2 depicts the costs for these operations when the VM runs atop a Minnowboard Turbot that uses Xen 4.10. VM-based rebooting requires both a hypervisor and a VM to attest to the AP, but as discussed in Section 4, the bulk of a new VM's interactions with the AP can be conducted in the background, as the old VM (and its applications) continue to interact with the outside world. So, the latencies in Table 2 are mostly hidden from applications, except for immediately after a reboot of the physical device; in this scenario, applications cannot access the network until the completion of a synchronous hypervisor association+attestation and a synchronous VM association+attestation.

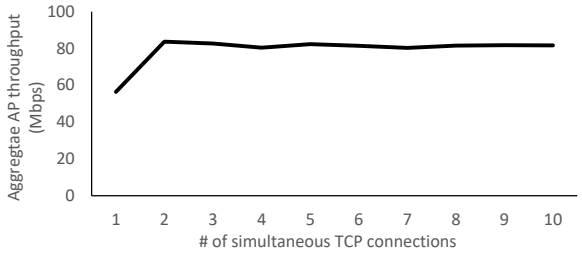

Figure 2: DeadBolt's virtual driver overhead as a function of the number of concurrent `iperf` flows.

Shuffling overheads: When running the CPU 2006 benchmarks [43] on a Minnowboard, the computational slowdowns range from 2.3% to 31.5%. However, many IoT programs are IO-bound, not CPU-bound, or have a mixture of IO bursts and CPU bursts. For these applications, shuffling overheads are much smaller. For example, we connected a Parrot AR 2.0 drone [34] to a Conga IA4 board which ran the (shuffled) drone controller software. The Conga used one Wi-Fi dongle to connect to the drone, and another to connect to a DeadBolt AP running on a Minnowboard; via the AP, the drone connected to a remote client which streamed video from the drone. The shuffling of the drone's controller software did not introduce statistically-significant degradations to the video latency or streaming rate.

Virtual driver overheads: To test the end-to-end overheads of virtual drivers, we ran `iperf` [23] on a Raspberry Pi 3 that connected through a Deadbolt AP to a remote client that was connected via wired Ethernet to the AP. We examined the stream latency and throughput when the AP associated:

- no virtual drivers with the stream, or
- two null (i.e., pass-through) drivers, or
- a TLS driver, or
- a compression driver and a TLS driver, or
- an IDS driver,[1] a compression driver, and a TLS driver.

The median throughput difference between running no drivers and running three drivers was 1%; the median latency difference was 2%. The impact on video streaming from the drone was similarly negligible.

Figure 2 shows the scalability of a Minnowboard AP as the number of concurrent `iperf` flows increases. To stress-test the AP, all `iperf` flows were generated by a laptop, with the sink being a desktop machine. The AP assigned a Snort driver, a compression driver, and a TLS driver to each flow. As demonstrated by Figure 2, the AP's maximum forwarding rate was approximately 83 Mbps, and the AP was able to maintain this rate once enough clients arrived to fully load the AP.

[1]The IDS driver used Snort [7] to scan packets for malicious data.

NGINX Throughput (Two worker processes)	
Steady state	**As a background VM loads**
1843 requests/sec	1614 requests/sec
sysbench: CPU (Two worker processes)	
Steady state	**As a background VM loads**
107.6 secs	135.0 secs
sysbench: Random I/O (Two worker processes)	
Steady state	**As a background VM loads**
Reads: 117.2 KB/s	Reads: 73.7 KB/s
Writes: 78.1 KB/s	Writes: 52.5 KB/s

Table 3: Performance slowdowns for a foreground VM as a background VM is launched on a Minnowboard Turbot. We used two application processes to ensure that both cores of the dual-core Minnowboard were contended by the foreground and background VMs.

Stage	**Delay**
VM_1: Snapshot the NGINX server	0.37 s
VM_1: Disable the NIC	0.12 s
VM_2: Enable the NIC	0.39 s
VM_2: Restore the NGINX snapshot	0.10 s
Total	**0.98 s**

Table 4: Visible downtime using VM-based patching.

VM-based patching: In general, a heavyweight device has only one VM running. However, Table 3 shows the performance degradation of applications in the foreground VM when a background VM is loading; the IoT device was a Minnowboard. To test the impact on CPU-bound and disk-bound applications, we used the `sysbench` tool [26]. To examine the impact on a network-bound workload, we ran NGINX, using the Apache Benchmark tool [1] to issue 100,000 requests, each of which were for 512 byte objects; the number of concurrent requests at any given moment was 100.

As shown in Table 3, the impact on the disk-bound workload was highest; the reason was that the loading of the background VM was also disk-bound. The Minnowboard required approximately 52 seconds to load the VM (i.e., to boot the VM to the point where it could be switched to the foreground). Table 4 illustrates the switching costs, showing that application-level downtime was only 0.98 seconds.

6.2 Security

We empirically tested DeadBolt's ability to prevent various kinds of real attacks. For example:

- IoT devices often run local web servers, to enable interactions with the outside world via HTTP. Using virtual TLS drivers, a DeadBolt AP prevents sniffing attacks on device traffic that would otherwise be exposed via cleartext HTTP. Furthermore, code shuffling prevents remote

adversaries from launching ROP attacks. For example, our Minnowboard ran a version of NGINX [32] that contained known ROP exploits [45]; however, due to code shuffling, the NGINX server resisted attacks from a Metasploit module that tried to leverage the ROP vulnerability [10].

- Drop-by-default firewall rules inside the DeadBolt AP can reject malicous traffic before it can interact with device software. For example, Shodan [39] is an IoT search engine which scans IP addresses, categorizing the discovered devices by device type; each type can then be mapped to a list of vulnerable software on that device. A DeadBolt AP automatically drops Shodan packets, hiding the local IoT devices from external scanning.

We believe that DeadBolt can stop additional IoT attacks that have been seen in the wild, but which we cannot directly replicate due to our lack of the relevant devices. For example, BASHLITE malware [44] exploits a parsing bug in older versions of the Bash shell. DeadBolt's quarantine protocol can isolate heavyweight devices with outdated shells; to protect lightweight devices which lack the ability to update, DeadBolt can use a virtual IDS driver to inspect incoming HTTP traffic and drop requests that have malicious BASHLITE HTTP headers.

7 RELATED WORK

The IoT industry has slowly realized the importance of device security. For example, the Industrial Internet Consortium, whose members include Intel, GE, and Huawei, recently published a white paper that describes several threats to IoT deployments [22]. Microsoft has also produced an overview of IoT security challenges [3]. However, these efforts have not resulted in the creation of a *concrete*, *application-agnostic*, *open* framework for securing IoT deployments. Thus, IoT developers are forced to secure their applications using vendor-specific, proprietary solutions. DeadBolt provides a generic security platform which can be leveraged by arbitrary types of IoT devices and applications.

Several research proposals have described the benefits of running security analyses in an IoT gateway [40, 48]; these proposals lack full designs and implementations, but share DeadBolt's high-level goal of using an IoT gateway to protect vulnerable devices from external threats. For example, IoTSec [48] envisions a security gateway that uses crowdsourced attack signatures to detect malicious IoT traffic. Gateways run lightweight VMs to act as middleboxes that manage device traffic. We believe that the virtual driver abstraction is more natural than a middlebox abstraction, since virtual drivers directly represent the encapsulation of individual devices within one or more layers of abstraction. The IoTSec design also lacks concrete mechanisms for ensuring that device software is patched and safe to expose to remote clients.

Like a DeadBolt AP, a HomeOS [13] gateway uses the driver abstraction to interact with individual IoT devices. For each low-level device protocol like Z-Wave or UPnP, HomeOS defines a connectivity driver that implements a standard interface for device discovery and device communication. Connectivity drivers are used by functionality drivers; each functionality driver implements the services provided by a broad class of device, e.g., a video camera or a temperature sensor. Other IoT frameworks use similar driver abstractions [41]. DeadBolt is compatible with HomeOS-style driver decompositions. However, HomeOS and related frameworks lack most of DeadBolt's security mechanisms. For example, a HomeOS gateway forces a device to register before using the network, but HomeOS does not use attestation with a hardware root of trust to validate the device's code. HomeOS does not force devices to protect dynamic state using techniques like forced reboots. HomeOS also does not support virtual drivers to safely expose devices that would otherwise be vulnerable.

In DeadBolt, heavyweight IoT devices use TPM chips to perform remote attestation. TyTAN [4] and Sancus [33] allow a low-end device to use a minimal amount of new hardware to support remote attestation without having to implement the entire TPM specification. DeadBolt is compatible with such devices. However, neither TyTAN nor Sancus deal with the practical issues that are addressed by DeadBolt's virtual drivers, device quarantine, and CFI enforcement. Also note that Sancus and TyTAN use a symmetric cryptographic scheme in which each device shares a unique secret key with a trusted IoT administrator. Thus, a Sancus or TyTAN device can only attest to the trusted administrator (or to parties with whom the administrator has shared the relevant secrets).

8 CONCLUSION

DeadBolt is a new security framework for IoT deployments. A DeadBolt AP quarantines devices that run untrusted or out-of-date software, and uses traditional firewall techniques to prevent external attackers from probing vulnerable IoT devices. Virtual drivers allow an AP to safely permit a lightweight, insecure device to communicate with other parties. The AP forces heavyweight devices to remotely attest their software stacks and periodically randomize code locations to thwart control flow attacks. To reduce the application downtime that is required to patch a heavyweight device, DeadBolt uses a VM swapping mechanism to overlap patching activity with normal application execution. Our evaluation shows that a DeadBolt AP can be efficiently implemented on a $90 Minnowboard; furthermore, device-side techniques like code shuffling and VM-based patching impose modest performance overheads. Thus, we believe that DeadBolt is a practical approach for securing IoT deployments.

REFERENCES

[1] Apache Software Foundation. Apache Benchmark, 2018. https://httpd.apache.org/docs/2.4/programs/ab.html.

[2] Berger, S., Cáceres, R., Goldman, K. A., Perez, R., Sailer, R., and van Doorn, L. vTPM: Virtualizing the Trusted Platform Module. In *Proceedings of USENIX Security* (2006), pp. 305–320.

[3] Betts, D., and Lamos, B. Internet of Things security architecture, June 14, 2018. Microsoft Azure documentation. https://azure.microsoft.com/en-us/documentation/articles/iot-security-architecture/.

[4] Brasser, F., Mahjoub, B. E., Sadeghi, A. R., Wachsmann, C., and Koeberl, P. TyTAN: Tiny trust anchor for tiny devices. In *Proceedings of ACM/EDAC/IEEE Design Automation Conference* (2015), pp. 1–6.

[5] Bregman, D. Smart home intelligence: The eHome that learns. *International Jounrnal of Smart Home*, 4 (October 2010), 35–46.

[6] Checkoway, S., Davi, L., Dmitrienko, A., Sadeghi, A.-R., Shacham, H., and Winandy, M. Return-oriented Programming Without Returns. In *Proceedings of CCS* (2010), pp. 559–572.

[7] Cisco. Snort, 2018. https://www.snort.org/.

[8] Cowan, C., Wagle, F., Pu, C., Beattie, S., and Walpole, J. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. In *Proceedings of the DARPA Information Survivability Conference and Exposition* (2000), vol. 2, pp. 119–129.

[9] CRIU. A project to implement checkpoint/restore functionality for Linux, 2018. http://criu.org.

[10] Dang, H.-V. Analysis of CVE-2013-2028, May 23, 2013. https://github.com/danghvu/nginx-1.4.0.

[11] Dang, T. H., Maniatis, P., and Wagner, D. The Performance Cost of Shadow Stacks and Stack Canaries. In *Proceedings of ASIA CCS* (2015), pp. 555–566.

[12] Dierks, T. and Rescorla, E. The Transport Layer Security (TLS) Protocol Version 1.2, August 2008. https://tools.ietf.org/html/rfc5246/.

[13] Dixon, C., Mahajan, R., Agarwal, S., Brush, A. J., Lee, B., Saroiu, S., and Bahl, P. An Operating System for the Home. In *Proceedings of NSDI* (2012), pp. 337–352.

[14] FreeRADIUS Server Project. FreeRADIUS, 2018. https://freeradius.org/.

[15] Garret, M. GRUB2 with TPM2 support, March 23, 2016. https://github.com/mjg59/grub.

[16] Gisbert, H. M., and Ripoll, I. On the Effectiveness of NX, SSP, RenewSSP, and ASLR against Stack Buffer Overflows. In *Proceedings of the IEEE International Symposium on Network Computing and Applications* (2014), pp. 145–152.

[17] Goldman, K. IBM's TPM 2.0 TSS, May 29, 2018. https://sourceforge.net/projects/ibmtpm20tss/.

[18] Goldman, K. IBM's TPM Attestation Client and Server, June 15, 2018. https://sourceforge.net/projects/ibmtpm20acs/.

[19] Goodin, D. Crypto weakness in smart LED lightbulbs exposes Wi-Fi passwords. *Ars Technica* (July 7, 2014). http://arstechnica.com/security/2014/07/crypto-weakness-in-smart-led-lightbulbs-exposes-wi-fi-passwords/.

[20] Goodin, D. Record-breaking DDoS reportedly delivered by >145k hacked cameras. *Ars Technica* (September 28, 2016). http://arstechnica.com/security/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever/.

[21] Google. Google Safe Browsing APIs, 2018. https://developers.google.com/safe-browsing/.

[22] Industrial Internet Consortium (IIC). Industrial Internet of Things Volume G4: Security Framework, 2016. https://www.iiconsortium.org/pdf/IIC$_{P}$UB$_{G}$4$_{V}$1.00$_{P}$B.pdf.

[23] iPerf. iPerf: The ultimate speed test tool for TCP, UDP and SCTP, 2018. https://iperf.fr/.

[24] Jonathan Corbet. TCP Connection Repair, May 1, 2012. https://lwn.net/Articles/495304/.

[25] Kelley, S. DNSMasq: Network Services for Small Network, March 18, 2018. http://www.thekelleys.org.uk/dnsmasq/doc.html.

[26] Kopytov, A. sysbench: Scriptable database and system performance benchmark, May 3, 2018. https://github.com/akopytov/sysbench.

[27] Kuppusamy, T. K., Torres-Arias, S., Diaz, V., and Cappos, J. Diplomat: Using Delegations to Protect Community Repositories. In *Proceedings of NSDI* (2016), pp. 567–581.

[28] Lear, E., Droms, R., and Romascanu, D. Manufacturer Usage Description Specification, June 4, 2018. IETF Draft. https://datatracker.ietf.org/doc/draft-ietf-opsawg-mud/.

[29] Lee, J. Smart factory systems. *Informatik-Spektrum 38* (2015), 230–235.

[30] Malinen, J. hostapd and wpa_supplicant, January 12, 2013. https://w1.fi/.

[31] MinnowBoard.org Foundation. Minnowboard, 2018. https://minnowboard.org/.

[32] NGINX Inc. Welcome to NGINX Wiki!, 2017. https://www.nginx.com/resources/wiki/.

[33] Noorman, J., Agten, P., Daniels, W., Strackx, R., Herrewege, A. V., Huygens, C., Preneel, B., Verbauwhede, I., and Piessens, F. Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base. In *Proceedings of USENIX Security* (2013), pp. 479–498.

[34] Parrot SA. Parrot AR Drone 2.0 Elite Edition, 2018. https://www.parrot.com/global/drones/parrot-ardrone-20-elite-edition.

[35] Robles, J. R., and Kim, T.-H. Review: Context Aware Tools for Smart Home Development. *International Journal of Smart Home*, 1 (2010), 1–11.

[36] Roemer, R., Buchanan, E., Shacham, H., and Savage, S. Return-oriented Programming: Systems, Languages, and Applications. *ACM Transactions on Information and System Security (TISSEC) 15*, 1 (March 2012).

[37] Safford, D., Kasatkin, D., et al. Integrity Measurement Architecture (IMA), 2018. https://sourceforge.net/p/linux-ima/wiki/Home/.

[38] Schneier, B. Security Risks of Embedded Systems, January 9, 2014. Schneier on Security blog. https://www.schneier.com/blog/archives/2014/01/security$_{r}$isks$_{9}$.html.

[39] Shodan. Shodan IoT Search Engine, 2018. https://www.shodan.io/.

[40] Simpson, A. K., Roesner, F., and Kohno, T. Securing vulnerable home IoT devices with an in-hub security manager, January 2017. University of Washington. Technical Report UW-CSE-17-01-01.

[41] SmartThings. SmartThings Developer Documentation: Overview of Device Handlers, 2018. http://docs.smartthings.com/en/latest/device-type-developers-guide/overview.html.

[42] Spec Sensors. Gas Sensors for the Internet of Things, 2018. https://www.spec-sensors.com/.

[43] Standard Performance Evaluation Corporation. SPEC CPU 2006, January 9, 2018. https://www.spec.org/cpu2006/.

[44] Trend Micro. Bash Vulnerability (Shellshock) Exploit Emerges in the Wild, Leads to BASHLITE Malware, September 25, 2014. https://blog.trendmicro.com/trendlabs-security-intelligence/bash-vulnerability-shellshock-exploit-emerges-in-the-wild-leads-to-flooder/.

[45] VN Security. Analysis of NGINX 1.3.9/1.4.0 Stack Buffer Overflow and x64 Exploitation, May 21, 2013. http://www.vnsecurity.net/research/2013/05/21/analysis-of-nginx-cve-2013-2028.html.

[46] Williams-King, D., Gobieski, G., Williams-King, K., Blake, J. P., Yuan, X., Colp, P., Zheng, M., Kemerlis, V. P., Yang, J., and Aiello, W. Shuffler: Fast and deployable continuous code re-randomization. In *Proceedings of OSDI* (2016), pp. 367–382.

[47] XEN. Unstable development version, June 9, 2018. http://xenbits.xensource.com/xen-unstable.hg.

[48] Yu, T., Sekar, V., Seshan, S., Agarwal, Y., and Xu, C. Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things. In *Proceedings of HotNets* (2015).

[49] Zamora-Izquierdo, M. A., Santa, J., and Gomez-Skarmeta, A. F. An Integral and Networked Home Automation Solution for Indoor Ambient Intelligence. *IEEE Pervasive Computing*, 4 (2010), 66–77.