RIGADO

Whitepaper

IoT Security at Scale

Managing end-to-end security for commercial IoT gateways

Introduction

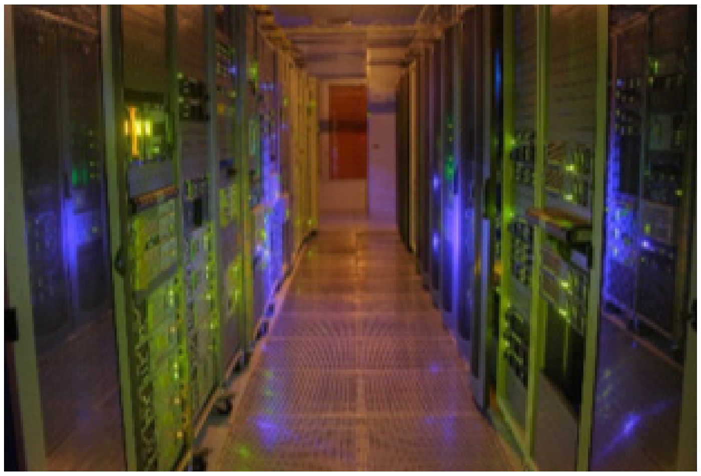
This white paper discusses IoT security at scale and shares some of the experiences Rigado has had working with over 300 clients and 5 million connected devices helping them manage end-to-end security of IoT gateways in commercial IoT applications.

Almost all IoT project teams follow similar DevOps practices for cloud-based continuous integration and deployment. However, when developers are focusing on building their application that gathers, for example, the guest experience data for their IoT application, they are not spending enough time dealing with ongoing security patches. This creates a huge security problem for these teams and the solutions they build.

Fortunately, there are mitigation techniques and solutions in the form of a platform they can build on that includes regular patches and automatically updated security infrastructure. Before settling on a specific solution, developers need to know as much as possible about the threat models associated with IoT Gateways and IoT devices, and some of the methods used to prevent or mitigate those threats.

Threat Models & Mitigation

Let's begin by looking at something we are possibly more familiar with, data centers and public cloud. In the data centers and when using public clouds they typically have enough resources to use VMs or containers to separate workloads and reduce risk. They usually have excellent physical security such as keycards, which limit the personnel who can access the data center. Network security is also very good; software-defined networking provides limited access to different hosts or ports. Also, there is a wide variety of performance monitoring anomaly detection tools available. Overall, when working in a data center and/or a public cloud, developers look to the cloud provider to solve most of these security problems. Many of the developers moved to the cloud for that reason, assuming that adequate security measures were already in place.



By contrast in IoT, the various use cases and environments demand their own specific solutions. For example, consider the security requirements of a chain of hotels. Deploying IoT devices into hundreds or thousands of rooms drives down the budget available for hardware resources like processor and RAM that can be deployed on these devices, which are used to support multiple different simultaneous applications like guest experiences, monitoring the minibar, integrating with HVAC, or controlling the lighting. In the hotel room there is no physical access control other than the gateway is hidden in the ceiling, behind a wall, or inside a container of some sort.

The environment of a public data center is very different from that of the hotel rooms and warehouses, conference rooms and commercial buildings, where IoT devices are typically deployed. Each has its own set of threats that must be dealt with based on system defenses that can be constrained by small amounts of RAM or slow processor speeds.

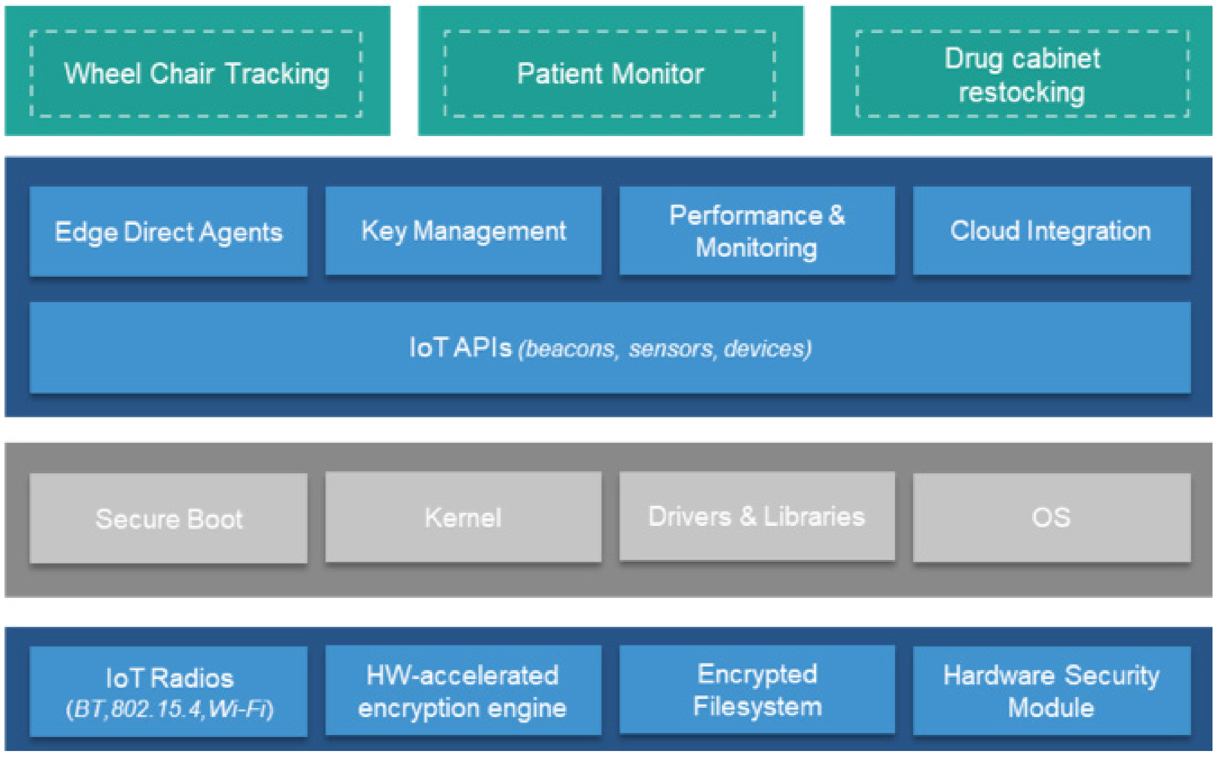
IoT Edge Device Security Threats

There are many different threats to IoT edge devices. For example, there is the problem of the “nosy neighbor” which is a play on the common multi-tenant problem of “noisy neighbor” only is more concerned with security than resource hogging. In the cloud there are always many different applications running together on a single piece of hardware. We need to provide the same security solution in IoT wherever we’re multitenant – however we have limited resources, so we need a low-cost solution that will be effective for creating similar virtual environments, confinement, and security.

We also have to be cognizant of physical attacks since that gateway and/or the IoT device might be accessible. A malefactor could tamper with these controls to change the behavior of the application. They could crack open the device and actually steal some of the information stored inside, such as API keys, or other IP. Like many cloud applications, we worry about someone exploiting the software bugs that are on these devices to create some sort of remote execution. There are a lot of stories about botnets and other IoT devices being used as a vantage point for attackers to launch denial of service attacks, mine crypto currency, or other clever ways that attackers can devise to take advantage of IoT devices.

Confinement

In multi-tenancy situations, the threat is that there are other unauthorized software and applications on the IoT device. These problems have been solved in the cloud and in VMs with containers. But on small, resource-limited IoT devices the overhead of running a VM is untenable. Even using containers like LXD drives up memory and CPU usage, while adding to networking complexity with elements like bridges or VLANs. Meanwhile the IoT applications need to interact with non-network interfaces like BLE or serial and require highly granular permissions to function properly.



At Rigado, we've picked Snaps for our application confinement to solve these problems because it limits app permissions using AppArmor Seccomp, cgroups and namespaces. It behaves as if you are running an application on the familiar Ubuntu Linux system. Confinement keeps the OS files safe from the application. Even though it runs the daemons as root, it doesn't give access to modify elements like the /etc/ directory or any of the kernel or other files systems that are outside the application. It defaults to limiting access. For example, you can allow access to the dbus interface to talk to Bluez for Bluetooth, but at the same time disallow access to manage Wi-Fi configuration over this same dbus interface. Snaps also keep everything needed to run the application together. The libraries and the run times are all together, which makes those stacks easier to manage and update.

With the confinement limiting access, capturing and reporting security events are critical for security at scale. Imagine for whatever reason that app is compromised, reporting that it is trying to execute, even if it's failing – it may be trying to access privileged paths or resources on the IoT device. Receiving notification of that policy violation will essentially allow you to do early detection and remediation of any issues –- you can only do that if you have strong confinement along with strong access controls and granular permissions on these devices. The reasons for confinement really aren't different than the reasons for containers in the cloud – you have a predictable, repeatable and immutable infrastructure. However, on these edge devices where resources are limited, we need a slightly different mechanism that has much lower overhead.

Tampering by Physical Attacks

Another class of threats that are important to consider when designing an IoT device are the physical attacks can actually be made on that device. An important one is tampering with the device to change its behavior. A malefactor could modify the code that's running on a physical device so that it might act as a "man in the middle." Imagine a system that's checking for usage and providing that information for billing purposes. If an attacker can access and modify the code, he could change the amount that is reported. Or he might decide to siphon off personal data and hold that for ransom. Attackers might also be able to mine crypto currency using the resources that you have placed in these IoT devices.

```
function checkLevel
{
 if (x > y)
   reportUsage(x/2);
   skimInfo(x);
   mineCoin();
}
```

When the attacker takes apart one of these IoT devices, he should not be able to plug serial cable into a header on the PCB and have root access. We have seen this configuration in several production commercial devices and it is always amazing – a prime example of worst practices. So, your first check box should be to make sure there is no open debug console on the device. This is very, very important. If there is a terminal, often there's a good reason to have a serial interface on the device for activities like debugging.

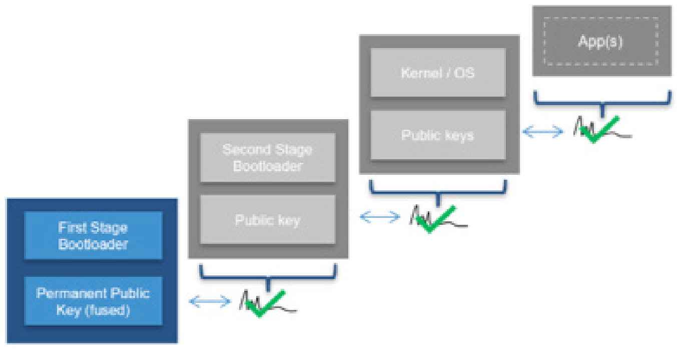
Don't employ a username and password combination on the device that uses defaults. This practice is probably one of the biggest contributors to the proliferation IoT botnets that are out there with user devices. It's better to have a managed system delivering configurations remotely than having any kind of a default user name and password that users should change when they install. Our experience is that this hardly ever happens.

If the purpose of these physical attacks is to actually change the behavior of an application, there are remedies. One way to combat the attack is to make sure that the app can't be changed when we use snaps. These are essentially read only file systems that prevent someone from tampering with the code bits on a device and instead would entail replacing the entire snap. Security is built in layers. Even if we don't allow console access, we still want to prevent something like an attack designed to replace an authentic snap with one that's loaded with malware.

Fortunately, Ubuntu Core has a strong trust model that uses SHA384 signatures to validate OS and applications as they're installed. Verifying the application boils down to having a signature that the system can verify. It does that by holding a copy of the public key and then making sure those match. And there is a whole chain of verification that that public key has in fact been signed. If we follow that signature path all the way back eventually we end up at the kernel. We want to make sure that the kernel has been verified before it's run – the mechanism that we use on our IoT gateways to accomplish this is a secure boot.

Secure Boot

How does a secure boot work? Deep in the processor there is a first stage bootloader with a set of fuses that are permanently burned with a public key. The key provides public use for verifying the second stage bootloader. The first stage loads the second stage from the external flash and computes its signature; it then compares the results with the public key before starting the execution. If that succeeds, the second stage will perform the same loading of the kernel from external flash and continue loading all the way up to the apps. Suppose, for example, someone takes the Cascade Gateway off the wall and tries to tamper with the files in the flash chip in order to introduce new behavior. The secure boot process simply prevents the system from booting if those changes were made all the way down to these layers of the OS.



There are a lot of keys here and so it's important to note that their management is fairly complex. Fortunately, Canonical (the company behind Ubuntu Core) handles the kernel signing keys through a workflow and a process that Canonical maintains.

Rigado takes that public key and makes it into our second stage bootloader. Workflows are used to sign that second stage and protect our keys. At that point, only assets that are actually transferred to the factory where everything is loaded contain public information. Since setting up those workflows is non-trivial, many well-intentioned IoT projects skip over setting them up correctly. This jeopardizes the entire security of an IoT process because a secure boot is the root of trust for any commercial IoT security at scale. Because that is so important Rigado has decided to do that for every one of the gateways that we deploy by configuring them in the factory with the secure boot process.

> **Secure boot is the root of trust for any commercial IoT security at scale.**

Protecting Intellectual Property

Another physical attack threat to IoT devices requires protecting the IP – the intellectual property – and any of the assets like API keys or configurations that actually live on that device at the edge. Gateways and applications need those credentials to start conversations up to the cloud and down to the devices. But if intruder can, for example, hack a Raspberry Pi by pulling out a micro SD card and put it into a computer, they can browse through the files on that filesystem as easily as looking at the pictures from their summer vacation taken by a digital camera. When the application that is being written is in something like Python, NodeJS or Java, or interpreted languages that are all written in plain text, your precious IP there at the edge is very vulnerable.

Secrets Management

When delivering an application, often it is tempting to put something like an access token directly in the code. But to maintain IoT security at the edge, we definitely need to consider not putting those credentials directly in the app code. It's best to use a service or a secrets management tool of some kind to deliver those secrets to the device when it's running instead of passing it through an entire CICD pipeline where there's a lot more exposure.

Some security do's and don'ts:

- Don't put credentials into the app code
- Credentials should be unique for every device
- There should be no default usernames and passwords for the device itself. Even at the application level it is a security best practice to keep each one of the devices credentials unique.
- In order to keep those credentials unique, assuming there's there is a mechanism like a secrets management service that can deliver that secret to the device, it still probably needs to be stored on the disk.
- When it's on the disk there is still the risk that an attacker will be able to read it out of the flash filesystem. So, we've considered several different mechanisms for encrypting.

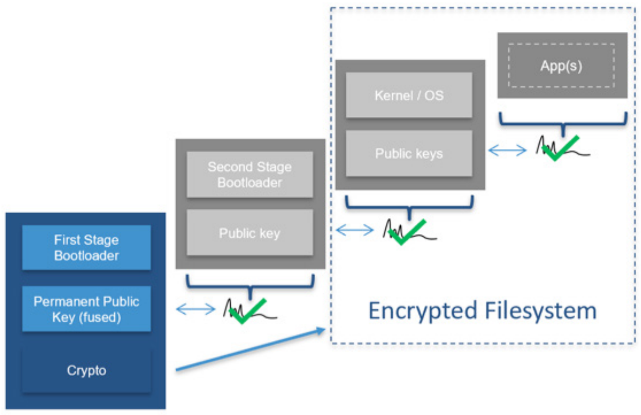
Encryption

One of the strategies for IOT security we considered was the provision of a set of encryption libraries. The chipsets inside the gateway provide encryption allowing us to deliver a set of libraries that each application can use to protect sensitive information.

But we ultimately decided not to do that for two reasons. The first is that this approach might protect elements like a credential or a configuration, but it typically doesn't protect the entire application – for example the entire Python application or NodeJS application. And the second reason is that introducing something like a library or utility for encryption would require that every application be modified.

That kind of friction is the enemy of security when we're talking about building IoT at scale. We want to make it extremely easy for every one of our customers to migrate their applications to Cascade. This means that if they started doing development on a laptop or a Raspberry Pi, we don't want to make them stop and use a special configuration utility to be able to take advantage of encryption on these devices.

The way to solve this problem is to provide encryption for the entire file system so that it encrypts and protects the entire app, including: all of the code; all the configure information; and all the credentials that might be stored alongside them. This is accomplished using that secure boot process mentioned above.



This approach ensures that we can trust the software we are running. This trust allows us to provide access using a secret key that's buried deep inside the processor. So again, when the second stage bootloader is authenticated, there is a crypto key that can be passed into the kernel so that entire encrypted file system can be unlocked and used.

This is a little different than the usual approach. Encrypted file systems in Ubuntu have been available for a while. However, a lot of those systems depend on a human entering a pass phrase during boot or plugging a USB stick into the device during the boot process. Because we can take advantage of a secure boot, we can allow this mechanism to be used in the field without any human interaction. This solves tampering or changing application issues and solves security for the encryption of configuration credentials. This solution is very important for devices that have physical access exposure, which allows an attacker to get their hands on them.

But these are not the only threats that IoT devices have to withstand. Most of the previous threats focused on motivated attackers with physical access. But the scariest attack is the one in which the attacker can remotely turn an IoT device into a botnet because of a software vulnerability.

High Profile Vulnerabilities

There have been numerous high-profile vulnerabilities over the last few years – from Meltdown and Spectre, which may not be as appropriate or applicable to gateway devices – to vulnerabilities like Heartbleed, BlueBorne attacking the Bluetooth connectivity, or Krack which attacked the WPA2 Wi-Fi encryption. Huge numbers of vulnerabilities happen every month – for example, something like 1800 new CVEs (Common Vulnerabilities and Exposures) were filed in June 2018.

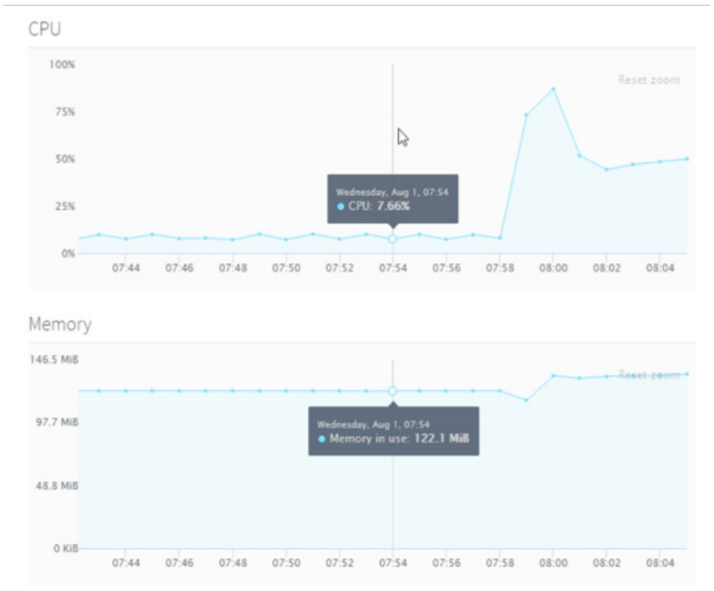
Given this high level of activity, the most important factor in IoT security at scale is the automatic application of frequent security patches. That's difficult to do when the system does not have a clean separation. Fortunately, with Canonical's Ubuntu Core the snaps provide a good clean separation of the OS, the system and the applications, allowing them to be updated independently. For example, Rigado and Canonical are providing kernel updates to the Cascade system on a cadence of about one update every three to four weeks. Critical patches for high-profile vulnerabilities can be delivered faster, as necessary.

It is extremely important to implement automatic updating that is independent of the app feature releases.

Think about a team that is working very hard on a regular basis having to cope with a feature release or a product team that is pushing to layer in new features also having to do hot fixes or implement special releases for security issues. These are not practices that most teams take time for. Therefore, finding a mechanism at scale that automatically handles these special security releases and OS updates is a priority item.

Monitoring is Key

No security is perfect. If you have a vulnerability that is not patched, a malefactor could potentially take advantage of the system or an application remotely. Preventive measures are not enough – monitoring is also important.



At scale, centralized application performance monitoring can identify critical issues.

For example, take a situation where CPU performance was nominal – 10% or less – and then it spiked to 80-90% percent for some reason. These sorts of anomalies need to be flagged so that an attack underway can be discovered quickly and a remediation plan developed immediately. There is also a need for some sort of centralized logging so that when you see a big spike in CPU, memory or network bandwidth, or you start having anomalies regarding which hosts the device is talking to, logging can help you go back, look through what's going on, understand where an issue might have been introduced and understand the scope of that breach. Because security isn't perfect, it's good to have the tools on hand that can help you deal with issues when they come up.

Bringing It All Together, Edge Protect

The issues discussed above have had a major impact on how Rigado developed one of the core pillars of its Cascade product, Edge Protect. It includes mechanisms from the hardware and manufacturing process with secure boot and provides you with all the security features enabled on our gateways. The issues discussed also influenced our decisions to use snaps and collaborate with Canonical to provide confinement that works on our resource limited systems. The partnership also gives us continuous security updates. These important factors play an essential role in helping us mitigate the threats to IoT security at scale.

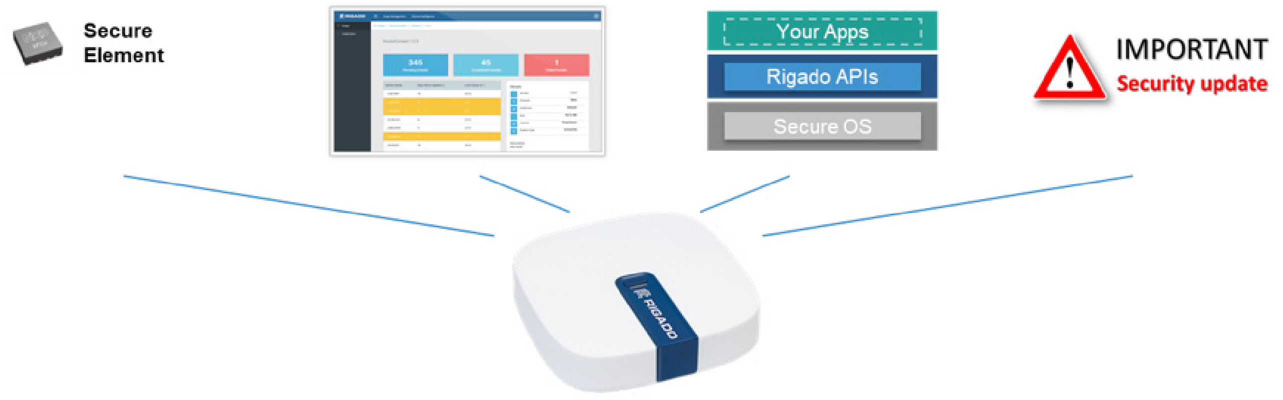
End-to-End Security for your for IoT Edge Infrastructure

Gateways are created with a secure ID and encrypted key at the point of manufacture.

Initial programming & all future updates are signed & verified for run-time protection.

All application run in secure containers, on an encrypted OS & filesystem.

Regular security patches are tested & published by Rigado as new risks emerge.



Conclusion

IoT security at scale is not layered in as an afterthought. It is a foundation built on a root of trust that uses confinement, signing, and encryption. It's an ongoing process with continuous security updates and active monitoring.

Ubuntu Core takes a security-first approach, which is why we chose it for the foundation of our Cascade platform and the background of our Edge Protect feature on Cascade. Edge Protect provides end-to-end security for IoT edge infrastructures. It's an essential part of Rigado's mission to provide our customers with advanced technology that allows them to quickly develop and deploy reliable, secure IoT solutions.