SORINT lab

SYNwall

A zero-configuration (IoT) firewall

About us

Cesare Pizzi

https://github.com/cecio

SORINT.lab

About us

Miso Mijatovic

SORINT.lab

SYNwall – Safer IoT? How?

There are several levels we can work on for IoT security. Software, hardware, monitoring tools are all good places to start from.

But usually IoT devices are out of a central control, with low profile hardware, tough environmental conditions and...we have no time to dedicate to maintain the security.

SYNwall – Alternatives?

So, may be we can not patch our IoT infrastructure and it will be very hard to maintain a "firewall-like" access control…

we have to think to a different way to do it

VPNs could be a valid answer to create “enclaves”, but they could be hardware demanding, difficult to setup and maintain.

SYNwall – The uW battle

The real IoT battlefield is not in CPU power, but in power drain.

Does really make sense to duplicate CPU duties (and CPU power consumption) by applying two times the encryption (SSH/HTTPS and VPNs)?

SYNwall – The uW battle

Also, in this battle, we may face trade-off between optimal security and speed, performance and energy consumption.

We expect to have to adjust this trade-off in the real world usage.

SYNwall – Can we make IoT "invisble"?

May be...sort of:

security through obscurity is not by any mean a good way to approach it, but: what if this is just an additional layer added with low effort?

Can we try to “hide” the IoT devices to other without getting crazy with configurations?

SYNwall – A code to rule them all 1/2

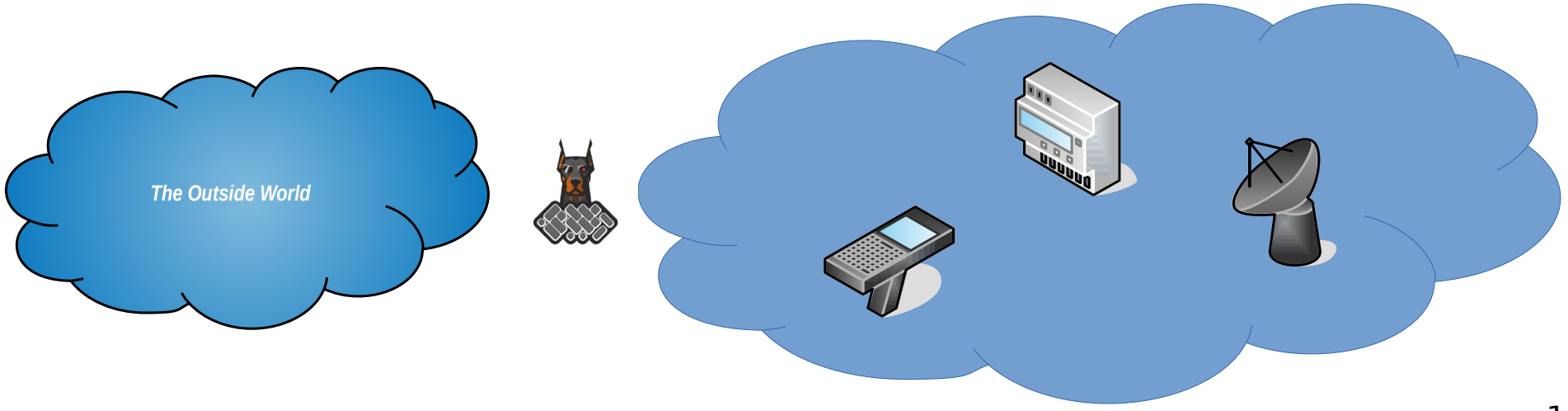
Here the idea:

creating a de-centralized one-way One Time Password code to enable the NETWORK access to the device.

No prior knowledge about who need to access is required, we just need a Pre-Shared Key to deploy.

SYNwall – A code to rule them all 2/2

Then, after this has been deployed, only the devices with the proper PSK will be able to interact: every other packet will be discarded, making the hardware unaccessible to the others.



SYNwall – Where the code will be placed?

The OTP will be in the payload of the SYN TCP packet, the very first TCP(*) packet sent to the device.

If the OTP is not recognized, the packet will be discarded. Otherwise the communication proceed.

(*) Not only TCP actually

SYNwall – BUT...payload in SYN is not allowed 1/2

Only partially true:

RFC allows payload in SYN packet (https://tools.ietf.org/html/rfc793#section-3.4), even if it shouldn't be sent to the application:

> Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake

SYNwall – BUT...payload in SYN is not allowed 2/2

Also someone else had the same idea, but with contrary purpose (TCP Fast Open, https://tools.ietf.org/html/rfc7413#section-2).

2. Data in SYN

Standard TCP already allows data to be carried in SYN packets ([RFC793], Section 3.4) but forbids the receiver from delivering it to the application until the 3WHS is completed. This is because TCP's initial handshake serves to capture old or duplicate SYNs.

To enable applications to exchange data in a TCP handshake, TFO removes the constraint and allows data in SYN packets to be delivered to the application. This change to TCP semantic raises two issues (discussed in the following subsections) that make TFO unsuitable for certain applications.

Most of the firewalls (Checkpoint, Palo Alto) have options to allow the payload.

SYNwall – Features 1/2

- zero (prior) knowledge of who is accessing required
- zero maintenance required (once deployed) (no firewall configs or iptables maintenance)
- protect in case of unpatched/unknown/0days remote exploitable vulns
- low HW profile and multiplatform (Intel, ARM, AARCH64, MIPS)
- works on different kernels (5.x, 4.x, 3.x)
- works with existing protocols (SSH, HTTP, MQTT)

SYNwall – Features 2/2

- with the proper toolchain, is virtually portable to any platform
- but…does not work for **UDP**? Yes, UDP works as well
- may help in be compliant with Industry Sec Standard
- SOCKS server provided to manage centralized access with the SYN Payload management
- fully Open Source solution
- ANSIBLE deploying script available

SYNwall – The gory details

Initial implemenation as a Linux kernel device driver.

This is its first form, we'd like to port this at SoC level (es. ESP8266).

SYNwall – The gory details

The implementation uses the lightweight hashing algorithm **Quark** (designed by Jean-Philippe Aumasson, Luca Henzen, Willi Meier and María Naya-Plasencia) to create the OTP:

https://en.wikipedia.org/wiki/Quark_(hash_function)

This choice allow to cross compile the driver on several architecture (x86, ARM, MIPS, ... virtually every architecture supported by Linux).

SYNwall – The gory details

We leveraged the implementation done by Jean-Philippe Aumasson

https://github.com/veorq/Quark/

with some minor mods (not at the algo ;-)), to make it kernel space compliant.

The default implements c-Quark (at least 160-bit security), but it‘s possible to downgrade if lower footprint is needed.

SYNwall – The importance of "One-Way"

The authentication is One-Way, and it is a must: this because one of the basic feature is to be as much silent as possible with not known devices (“don't talk to strangers”).

So every information needed for the authentication must be in the packet. The only way the destination have to validate the OTP is to recompute the hash with the proper information.

SYNwall – The OTP

OTP is basically built on a two values binary blob

Size of the blob depends on Quark algotihm and PSK length

SYNwall – The Hash

The hash is computed on the following values

Quark Hash

PSK + Time + Random Buffer + Dest IP

This makes the HASH unique and not replayable.

SYNwall – PSK

PSK is valid from 32 to 1024 bytes.

Quark Hash

PSK + Time + Random Buffer + Dest IP

SYNwall – Time

Time is split in intervals (“precision” from now on)

Quark Hash

PSK + **Time** + Random Buffer + Dest IP

The time used for the HASH computation will be rounded to a given precision. This allow to manage time skew between the devices.

To improve performance (avoid modulo operations) and overcome some limitations in some IoT processors we decided for a "power of 2" precision, which allow us to round a number with a simple bitwise AND operation.

SYNwall – Random Buffer

Random buffer of the same length of the PSK

Quark Hash

PSK + Time + **Random Buffer** + Dest IP

We wait for random buffer to be initialized before injecting the module, to be on the safe side in protecting the Hash and PSK (more on this later).

SYNwall – Destination IP

To be fully safe against replay attack, this must be part of the Hash.

Quark Hash

PSK + Time + Random Buffer + **Dest IP**

If NATs are present in the communication, these may break this verification. So, the presence of Dest IP in Hash is controlled by an option.

SYNwall – The XORed value

XOR between the PSK and a random buffer of the same lenght

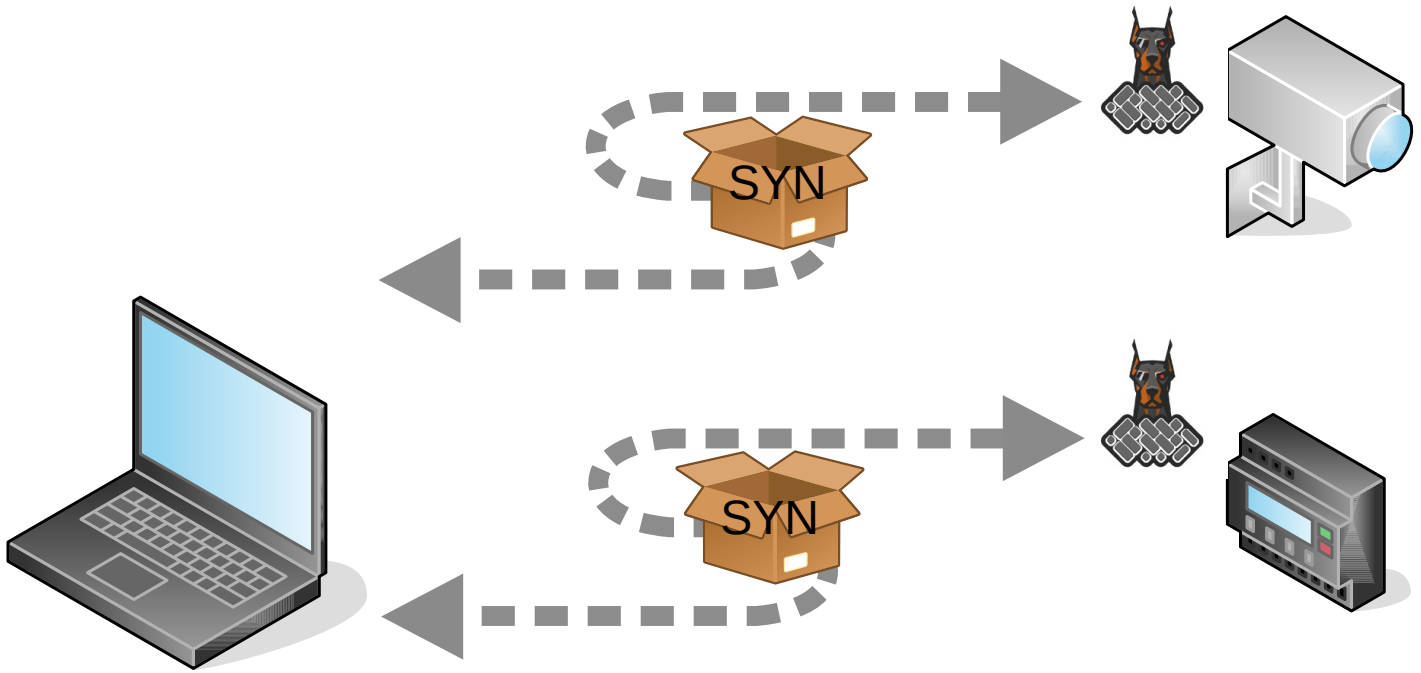
XORed value

PSK xor Random Buffer

The random buffer will be the same hashed in the first part.

SYNwall – Verification Process

Knowing the PSK, the client will be able to retrieve the Random Buffer and then use it to rebuild the Hash. If the Hashes match, the connection is allowed, otherwise the packet is dropped.



SYNwall – Anti Replay

All the used Hashes will be trashed by the receiving device, so that if one of them is replayed, no auth will be given.

It is necessary to trash all the received Hashes in a time precision slot.

SYNwall – TCP & UDP

The project was initially thought for TCP (as per name).

Then, during the development we decided to include UDP as well, directly in the driver itself.

SYNwall – TCP

The HASH is inserted into the payload of the SYN packet. During our tests we didn't noticed any strange behaviour once the packet reached the destination, due to the presence of an unexpected payload.

Right now the payload is delivered as it is. No removal performed.

SYNwall – UDP

We need to face some issues here:

- connection tracking
- payload removal
- common UDP protocols (DNS, NTP)

For the connection tracking we rely on the netfilter conntrack module, and we use this one.

The usage of UDP protection is optional: the module must be present on **both** the device, because the payload **must be removed** before forwarding the data to the application.

SYNwall – ICMP

We block ECHO requests...but not all the ICMP protocol.

If the device is routing traffic, blocking all the ICMP messages could be disrupting. This is open to discussion anyway. And can be modified too :-) (the beauty of OpenSource).

SYNwall – and for non-Linux platform?

The whole idea is built on the TCP/UDP stack, so it can be ported to other architecture as well.

One of next steps planned is the port for “System on Chip world”, starting from ESP8266.

SYNwall – Performances

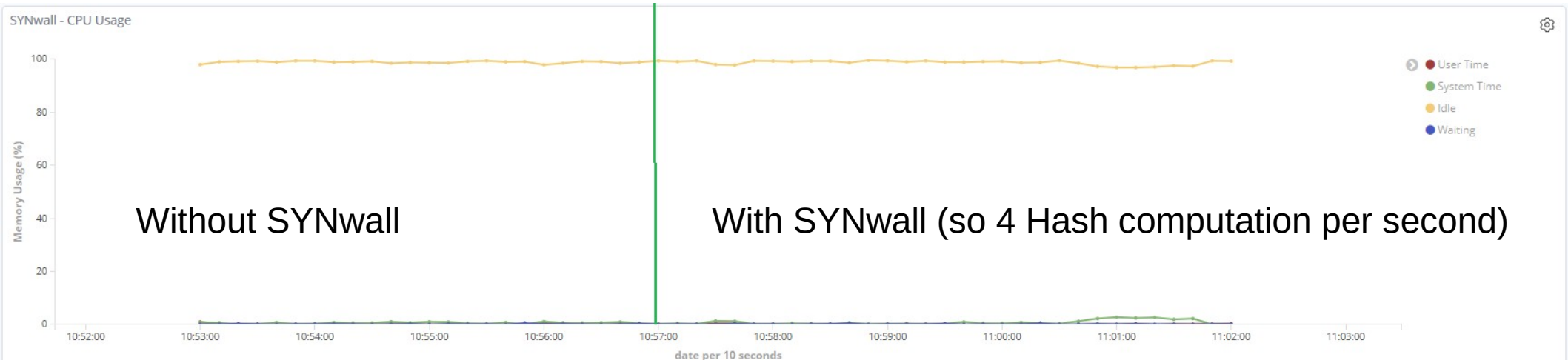
How much performances are affected?

- Quark is a very lightweight hashing algorithm
- We inspect just one packet per connection (SYN for TCP, first packet for UDP)

The impact is very low

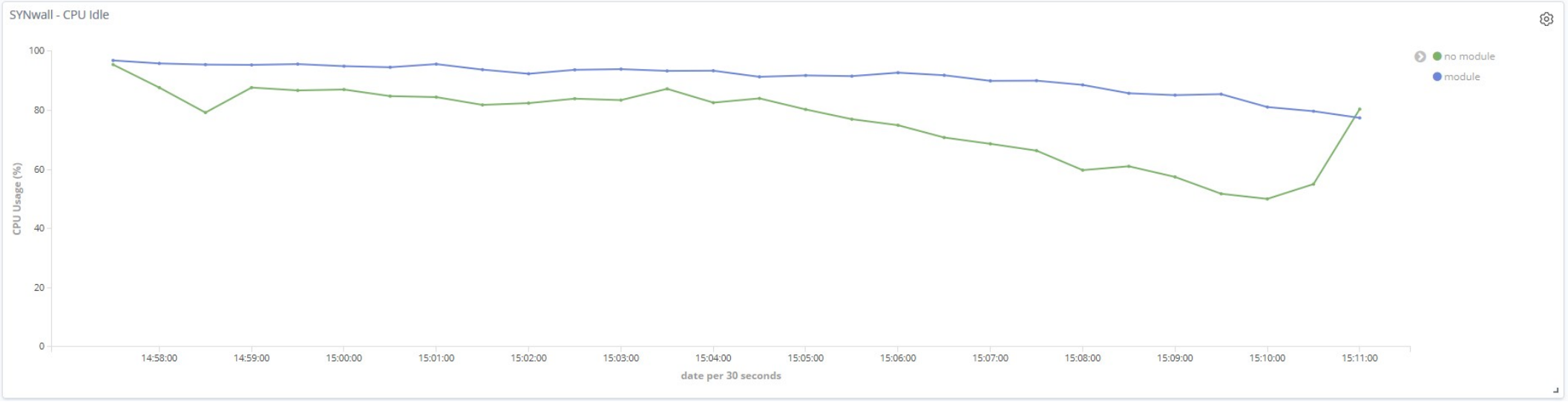
SYNwall – Performances

4 SYN pkts/sec on a Raspi Zero (ARM)



SYNwall – Performances

Heavy Load on a Raspi Zero (ARM), from 100 to 1000 pkts/second



SYNwall – Performances

But we have a drawback here: if an attack is done with the known OTP length, the device can be forced in a massive Hash computation.

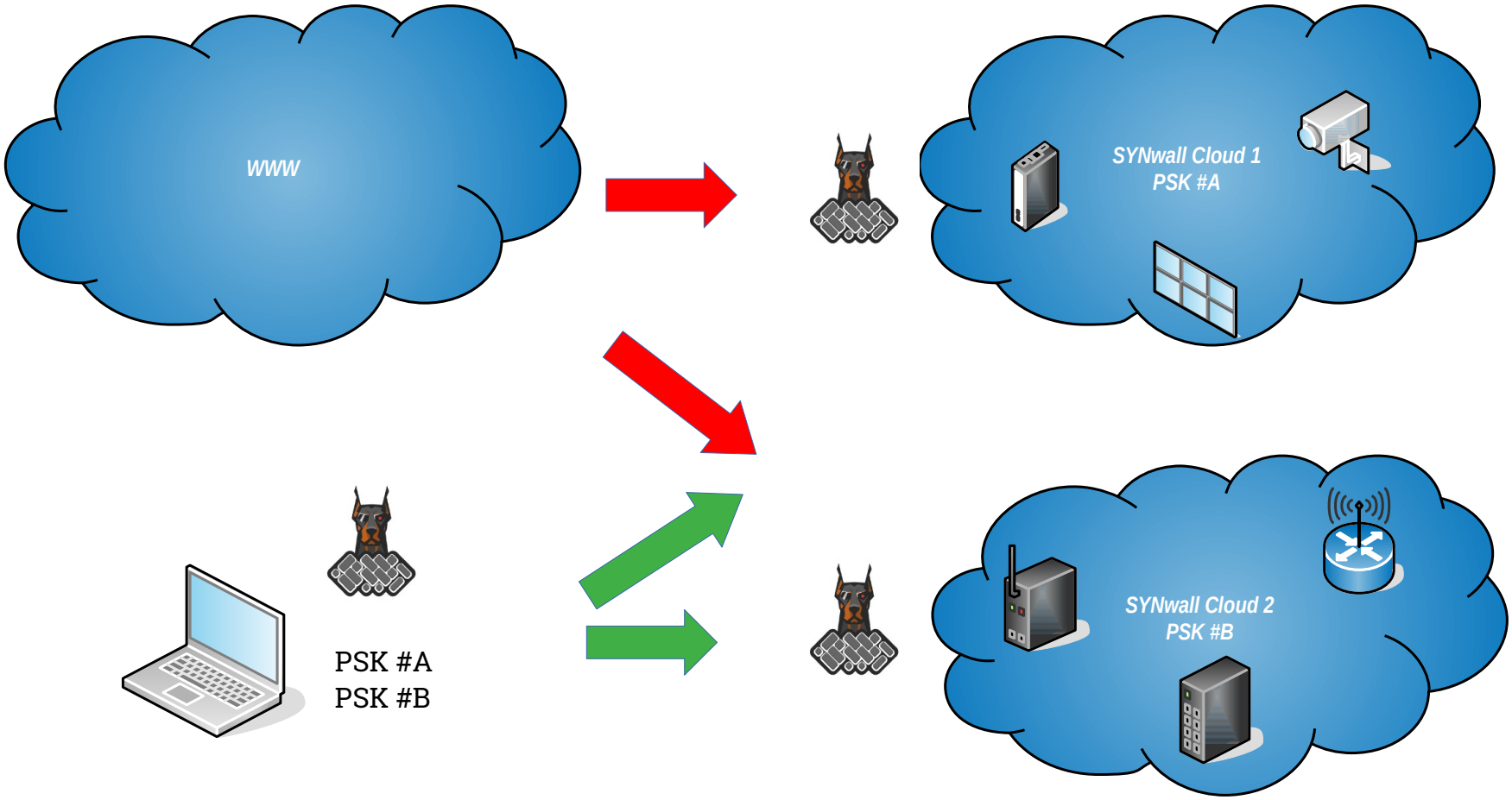
For this reason we introduced a DOS protection (optional) that can limit the hash computation per second, to protect device stability.

SYNwall – Manage the devices

A special module (SYNgate) is available, to manage the devices from a central point. It can be loaded on a SOCKS server to be used centrally. The PSK will be choosen on the destination IP basis.

We provide an OVA with a sample installation (with the well known “danted” SOCKS Sever) to try this out.

SYNwall – Manage the devices



No configuration required, just PSK

SYNwall – Companion Tools

In addition to the SOCKS-ready module, we provide an ANSIBLE script to automate the installation.

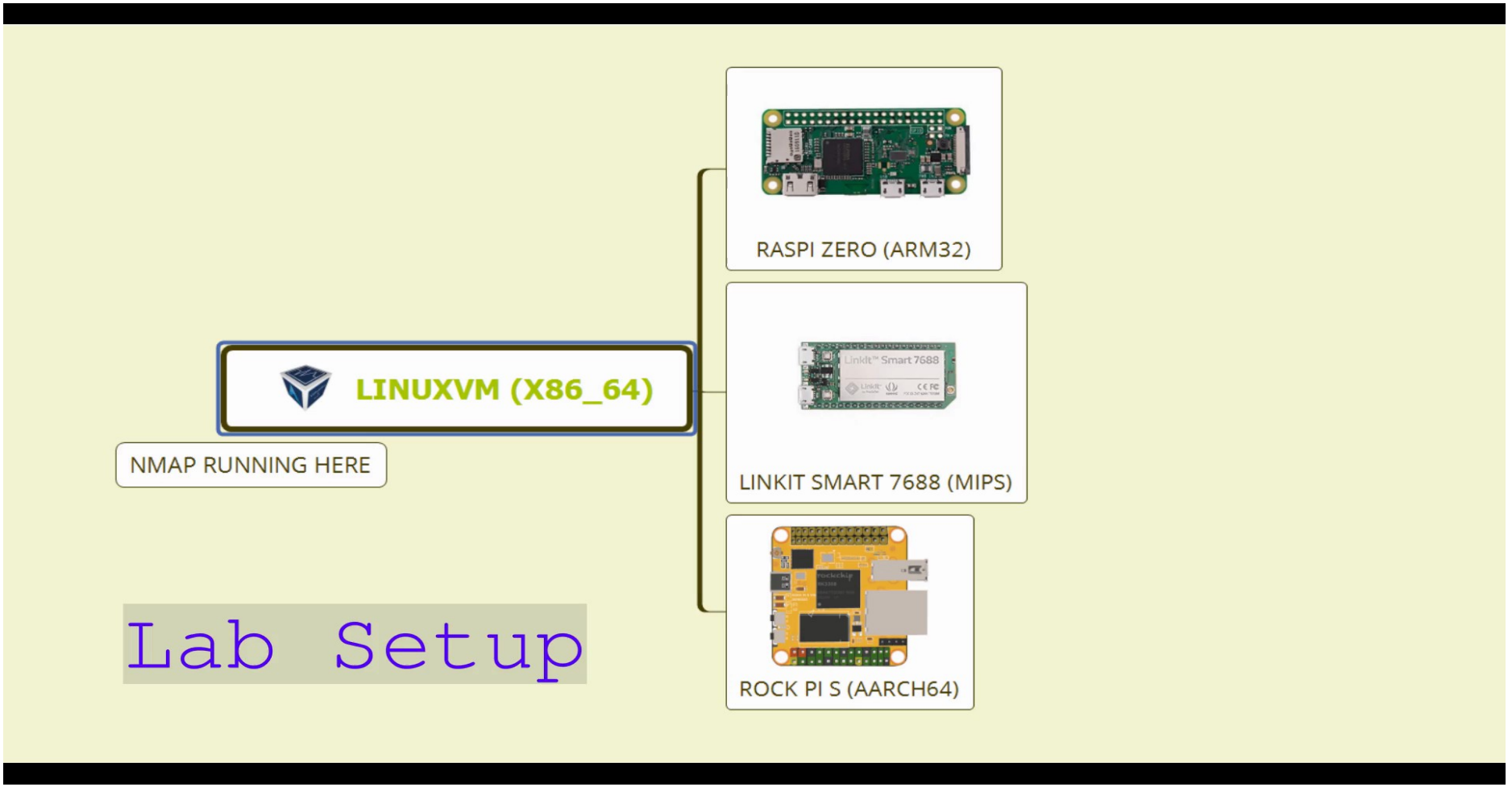
The script is highly configurable, allowing you to choose different installation options (where to get the software, how to install it, etc)

SYNwall – Failsafe mechanism

We know that thins could go wrong…and so we added some options:

- safe-delay at reboot: you have configurable number seconds after the module reload to get in the device
- one-time port-knocking code to temporary disable the module from remote
- anti-DOS mechanism for Hash computing

SYNwall – DEMO



SYNwall – Known Issues

Payload in SYN, even if not prohibited, may cause issue. TCP-FastOpen already tried to address this. Some firewall are dropping these packets, but most have specific options to allow them.

For directly exposed device, it shouldn't be an issue.

From our tests we saw few cases where the packet itself was cleaned up (VBox NAT) or dropped

SYNwall – Open Point

Securing storage of PSK on device to be safe from Phyisical Access Attack.

Right now the best approach looks like using some secure EEPROM for storing (like ATAES132A). The module supports the fileless on-the-fly PSK injection. We are in the process to do a PoC with an EEPROM chip and an ARM platform.

SYNwall – Known Limits

Possibility of MITM, due to the one-way scheme.

Consideration: SYNwall is not replacement of any authentication method, it's just an additional layer to make device as much invisible as possible and as much as network resilient as possible.

SYNwall – Known Limits

Possible entropy issue for random buffers.

Consideration: entropy for random numbers could be an issue on low-end devices. For the time being we are relying on the “wait_for_random_bytes()”, blocking the module activities until the pool is initialized. In this way we try to protect the PSK.

SYNwall – Known Limits

Brute forcing attack on XORed value.

Consideration: this must be evaluated and, recalling the trade off slide, carefully adjusted. We need your help here, also looking at the “Too Much Crypto” considerations (stay here for the next talk!! :-)).

SYNwall – Next steps

- community feedbacks…we need you to improve it!
- porting to other platform and environments (ESP8266, Arduino, Yocto Linux)
- IPV6 version
- Secure Storage readiness

SYNwall – References (thanks!)

Quark Hashing:

https://en.wikipedia.org/wiki/Quark_(hash_function)

Quark Hashing implementation (by JP Aumasson):

https://github.com/veorq/Quark

Survey on Prominent RFID Authentication Protocols for Passive Tags (by Rania Baashirah & Abdelshakour Abuzneid):

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6210508/

Untraceability Analysis of Two RFID Authentication Protocols (by CHEN Xiuqing, CAOTianjie & ZHAI Jingxuan):

https://www.researchgate.net/publication/308125441_Untraceability_Analysis_of_Two_RFID_Authentication_Protocols

SYNwall – Thank you

https://github.com/SYNwall

SORINT lab