Securing IoT Microservices with Certificates

Marc-Oliver Pahl
Technical University of Munich
pahl@net.in.tum.de

Lorenzo Donini
Technical University of Munich
lorenzo.donini@tum.de

***Abstract*—The Internet of Things (IoT) consists of distributed computing nodes. With increasing processor power such nodes can be used as hosts for microservices. IoT services routinely processes security critical data that affects the privacy, safety, and security of users. However, suitable security mechanisms remain missing. Fundamental open challenges are the authentication of services, securing the metadata of services, and validating the correct functioning of security mechanisms on distributed entities under different authorities. In this paper we present a certificate-based methodology for authenticating services, securely adding information to their executables, and validating the correct functioning of distributed entities of our design. We add X.509 certificates with extended attributes to the service executables. By introducing different trust anchors, services and their metadata are protected through their entire life cycle from developers to the computing nodes running them. Our solution enables distributed nodes to verify the security properties locally. It enables reliably changing certificate properties across the distributed IoT nodes. It features autonomous certificate management. We evaluate the traffic caused by our autonomous certificate management process quantitatively. The presented solution is churn tolerant and applicable to diverse distributed systems.**

***Index Terms*—IoT, security, certificates, x.509, microservices, autonomous certificate management, unattended nodes**

I. Introduction

The Internet of Things (IoT) consists of distributed computing nodes. A goal of the IoT is providing personalized services. To reach this goal, IoT *services* inherently process private data including sensor data and actuator data. Providing adequate *security* is therefore critical for IoT components. However, implementing security is still not a standard feature when developing IoT components. The Mirai botnet exemplified this in 2016 [1].

Three fundamental challenges for implementing *IoT service security* are *authentication*, *accountability*, and *integrity* [2]–[4]. The term authentication describes the verification of the identity of a service. In our context this especially includes authenticating that a service comes from a certain developer, store, or belongs to an IoT site. The term accountability describes in our context that actions can be tracked down to specific responsible entities. The term integrity describes the assertion that data is accurate and consistent.

In this paper we provide a certificate-based solution for achieving *authentication*, *accountability*, and *integrity* for IoT (μ-) services running on distributed computing nodes [5], [6].

This research has been supported by the German Federal Ministry of Education and Research (BMBF) in the project DecADe (16KIS0538).

As topology we assume a setting similar to software distribution systems for PCs or smartphones. Developers upload executables to a store that distributes them to the systems where they get installed. Different from classic software distribution systems, we assume that an *IoT site consists of distributed nodes that can run services*. This makes providing security on-site significantly more complex than securing a single trusted machine. To enable site-internal service management we assume the presence of a hierarchical service management infrastructure.

The following Sec. II introduces the setting more in detail. This section also details the research questions. Sec. III introduces the proposed security architecture. Sec. IV evaluates the traffic generated-by the proposed automated certificate renewals depending on changing amounts of services. This section also proposes a tool to enable real world use of the proposed solution even in scenarios with lots of running μ-services. Sec. V puts our work into the context of the state of the art.

II. Setting

The IoT consists of distributed computing nodes. An increase of the computing power computing power [7] enables running services on regular IoT nodes. We assume the presence of a service management infrastructure within an IoT smart space.

On the right of Fig. 1 a generic hierarchical service management architecture is shown. It consists of a Site-Local Service Manager (SLSM) that does the overall management of installed services. On each computing node a Node-Local Service Manager (NLSM) manages the locally running services. The SLSM manages the NLSMs that do the local management autonomously. The SLSM also communicates with the Store (center) that is used for service distribution.

For our solution we consider microservices (μS) to run on the distributed *unattended* IoT nodes. The μS setting implies that individual computing nodes do not require strong resources to run services. At the same time it implies that *many μS* are deployed within an IoT site, making it necessary to *handle the emerging security management complexity*.

The top of Fig. 1 shows the life cycle of a service. The goal of our security mechanism is to protect a service from the *development* over the *distribution* to an IoT space where it is *configured*, *deployed*, and *updated*. For our setting we assume many independent developers, one or multiple Stores,

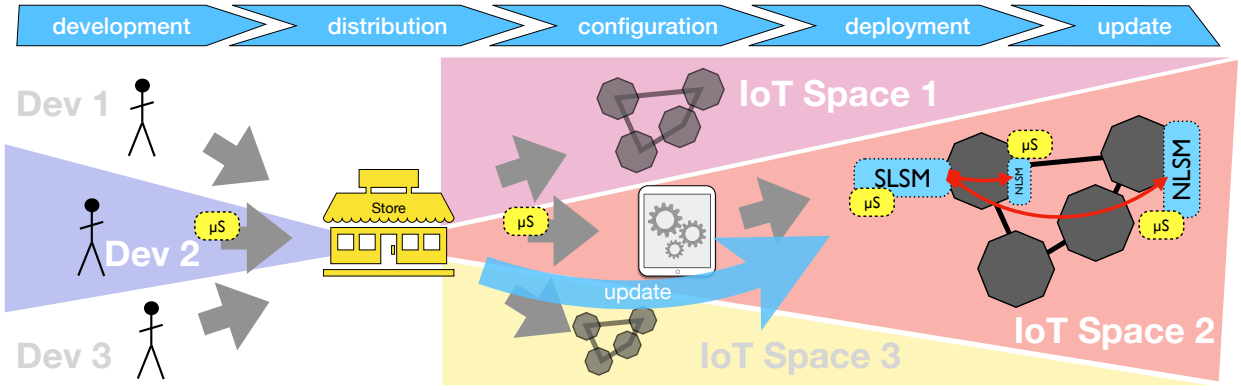

Fig. 1. Assumed IoT Service Ecosystem.

and multiple IoT sites. See Fig. 1 with three developers on the left and three IoT smart spaces on the right.

In contrast to many existing works [2] we assume that the developer sites, the App store, and the IoT spaces are *under different administration*. We assume that the IoT computing nodes and the developers behave well and are not malicious. We assume that the developers and the local sites have trust relationships to the store. Still we want to provide means to *control its correct operation when distributing services*.

A developer uploads a service to the store. The store offers it for distribution. Different IoT smart space sites exist. IoT site-locally a service is installed within a group of IoT devices under the management of the SLSM and the NLSMs.

The architecture in Fig. 1 is similar to other application distribution systems such as the smartphone App economy in Mobile Computing [8]. A major difference between smartphones or PCs and IoT systems is the distribution of components. The IoT is a *distributed system*. IoT components cannot be considered as trusted as parts inside a smartphone or a PC.

Our goal is protecting a service and its meta data over their entire life cycle. This includes the executable and related information such as configuration or security parameters.

Our work targets the following challenges for providing authentication, accountability, and integrity:

- Enabling distributed IoT nodes to identify if a service belongs to the IoT smart space site (authentication).
- Enabling distributed SLSMs to verify the correct operation of the Store (accountability).
- Enabling distributed NLSMs to verify the correct operation of the SLSM (and the Store) (accountability).
- Enabling distributed IoT nodes to check the integrity of the service executable and its metadata attributes (integrity).
- Creating a fully distributed solution to provide the scalability and flexibility that the IoT requires.
- Creating a self-managing solution that does not require user or developer interaction.
- Enabling a guaranteed update of the secured information that is distributed over the loosely coupled IoT nodes within an IoT smart space site.

The scalability and autonomy properties of the solution are especially relevant when it comes to IoT settings with μS where many services are running [5], [6].

III. Service Security Architecture

We introduce certificates to protect a) the executable of a service and b) additional metadata. The certificates are signed by different entities over the lifetime of a service. Site-locally certificates with a short lifetime are issued to implement a distributed guaranteed update scheme. As certificate lifetimes are used, this revocation scheme does not need Certificate Revocation Lists and can therefore be executed fully locally. Fig. 2 shows our security methodology applied to the setting from Fig. 1.

A. Service Certificates

For protecting the executable, and for enabling the secure adding of properties to it we introduce X.509 version 3 [9] certificates to the setting. We introduce the extended attribute *executableHash* for storing a cryptographic hash over the executable. Further properties can be stored either directly in the certificate or in a metadata file that is protected by a hash that is also added to the certificate.

The integrity of all data in the extended attributes is secured by the certificate. By adding the executable hash, the certificate gets *pinned* to the executable. Via the use of a cryptographic hash the integrity of the executable is protected. Hashes over other files can also be added as extended attributes, e.g. for securing content of a metadata file. Without restricting the generality we assume in the following that only the executable hash is part of the certificate and no other files are required to run the service executable.

B. Certificate Signing and Verification

For signing the introduced service certificate, we introduce different keypairs throughout the lifecycle of a service. Developers register at the Store and upload their public keys. When delivering a service to the store, a developer creates the executable, hashes it, adds the hash to a locally generated certificate, and signs the certificate with her private key. The developer certificate Cert_{dev} and the executable are uploaded to the Store.

The store checks the integrity of Cert_{dev} with the public developer key. It also checks the integrity of the executable.

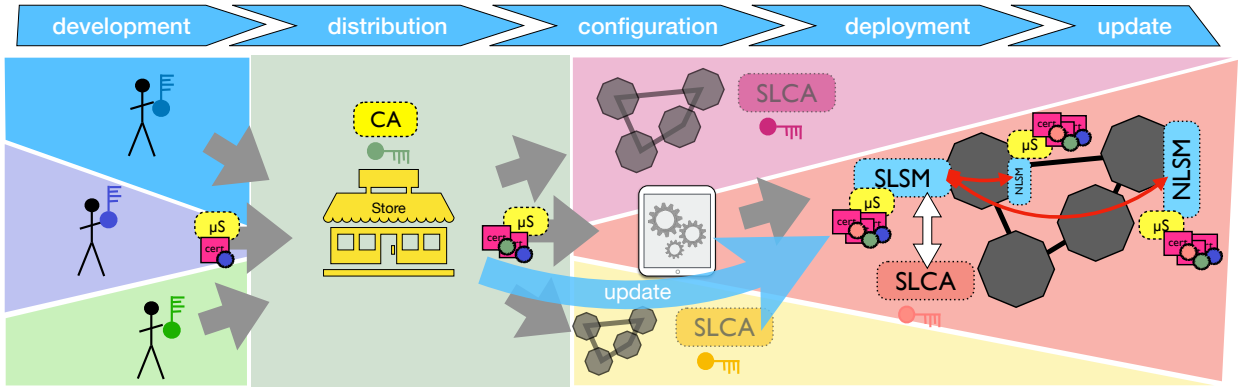

Fig. 2. Distributed Certificate-Based IoT Service Security.

Then it copies the data from the developer certificate and signs this Store certificate $\text{Cert}_{\text{store}}$ with its private key.

When a service is installed to an IoT site, the SLSM obtains the package $\langle \text{executable}, \text{Cert}_{\text{dev}}, \text{Cert}_{\text{store}} \rangle$. As in other software distribution scenarios, the SLSM trusts the Store and has its public key stored locally. It uses it to verify the $\text{Cert}_{\text{store}}$ and the executable.

Analog to the Store, the SLSM copies the values of the $\text{Cert}_{\text{store}}$ into a new certificate $\text{Cert}_{\text{SLSM}}$. For various purposes the SLSM may add or change data in the certificate. An example is adding or changing access policies and storing them in the certificate. After having made all necessary changes, the SLSM signs its certificate with the site's public key. For that purpose a Site-Local Certificate Authority (SLCA) is introduced: $\text{Cert}_{\text{SLSM}}$.

When starting a service, the service package is transferred to an NLSM. The NLSMs have the store key and the IoT-site-local SLSM key pre-configured. They verify the integrity of all data and execute the service. By using site-local certificates, an NLSM can *fully locally* determine if a service belongs to the site.

C. Verifying the Store and the SLSM

To verify if the Store did not manipulate the content of the Cert_{dev}, the SLSM can compare the content of the Cert_{dev} with $\text{Cert}_{\text{store}}$. To verify the integrity of Cert_{dev} it requires the public key of the developer.

The NLSMs can verify if the SLSM did not tamper the $\text{Cert}_{\text{store}}$ data by verifying that certificate with the pre-configured key of the store, and comparing the result to $\text{Cert}_{\text{SLSM}}$.

Carrying the three certificates to the NLSM enables a verification of the full certificate set locally. Like the SLSM, the NLSM requires the developer public key to verify if the store tampered the data originally provided by the developer. The crosscheck is especially relevant for ensuring that the executable was not modified and the hash matches in all three certificates.

D. Modification and Revocation via short Certificate Lifetimes

The site-local certificates enable a fully distributed verification of certificates when knowing the site's public key. As the nodes are unattended and we do not want to bother the user, the NLSMs automatically renew service certificates via the SLSM and the SLCA before they expire.

If the user *modifies* service metadata, such as access policies that are protected by the $\text{Cert}_{\text{SLSM}}$ the SLSM asks the NLSM to immediately renew the service certificate. If the NLSM is currently unreachable due to churn, the security critical properties will remain valid latest until the expiry time of the service certificate. A disconnected NLSM will automatically renew all expired certificates once it reconnects to the SLSM.

To enable a fully distributed certificate revocation it is desired to have short certificate lifetimes. For lowering the network traffic, CPU load, and to enable a longer operation of services on disconnected nodes, a longer certificate lifetime is desirable.

Under the assumption that the time is synchronized between all distributed IoT nodes, this solution implements a fully distributed certificate modification and revocation scheme that is enforced after a maximum time of $\text{Cert}_{\text{lifetime}}$.

As we use standard X.509 certificates, those can directly be used for establishing transport security via TLS, DTLS, or WTLS [3]. Via our extended attributes, diverse access control schemes such as RBAC, PBAC, or ABAC can be implemented on top of our solution.

IV. Evaluation

To assess the practical applicability of the automated certificate renewals we assess the caused traffic in a simulation. The IoT is highly heterogeneous in devices and resources. Therefore we measure the impact of our solution on theoretically unlimited, simulated resources to enable the reader to match the results with concrete IoT device characteristics.

Our simulated IoT site consists of 15 computing nodes running 30 μ-services each. As *simulation run time* we chose about 24h to capture the starting and stable cases of the system. We implement a node *churn rate* of 0,001%. After a drop out, a node stays *randomly between 1s and 2500s offline*. The outliers Fig. 3 are caused by reconnecting IoT nodes that refresh all local service certificates at once.

Our *site-certificate key length* is 2048 bit. The *additional payload* of a certificate for additional SLSM information, e.g. for access control, varies between 0 and 200 Bytes per

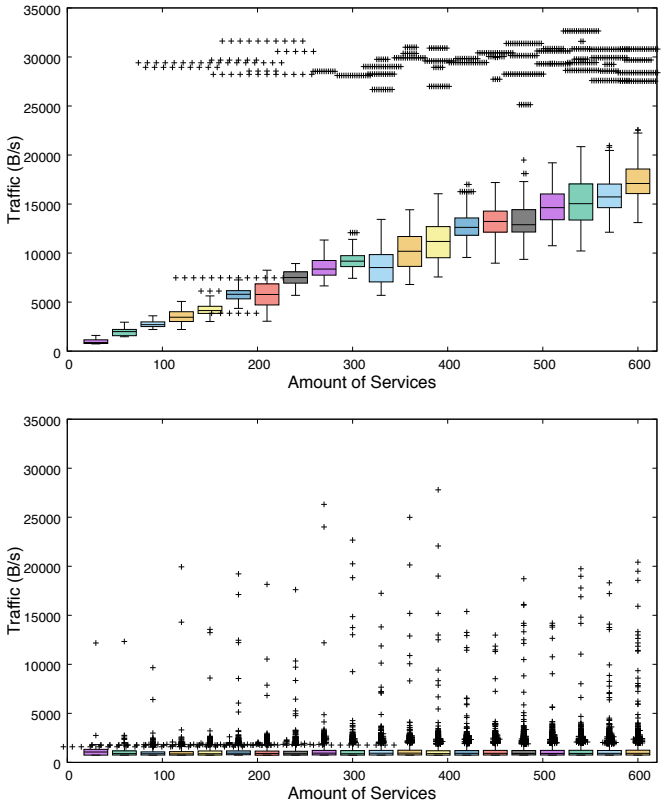

Fig. 3. Peak traffic generated by certificate renewal depending on the total amount of services without (top) and with a random backoff (bottom).

service. The *certificate lifetime* is set to 5000s since this value experimentally showed a good trade off between desired short life times and low traffic.

In our simulation we assume that all services are roughly started at the same time. This is a realistic assumption since IoT spaces are constantly running, and a power outage may cause all local IoT nodes to restart at the same time. Having the same start time and certificate lifetime, all renewals happen periodically at the same time resulting in traffic peaks. Fig. 3 visualizes this at the top. The boxplots show the peak traffics per second, omitting seconds where no communication is caused by our solution. This is relevant to understand the network dimensioning our solution requires to work.

To mitigate from unwanted renewal periodicity, we introduce a *certificate renewal backoff*. With the backoff mechanism enabled, services do not renew a certificate at the very moment when it expires, but use a random backoff. For the 5000s renewal interval we set the random backoff to 1250 seconds (25%). A renewal happens randomly between 1250 and 1s before a certificate expires. The effect of the random backoff becomes well visible when comparing the traffic produced by different amounts of services in Fig. 3.

The two box plots in Fig.3 aggregate the traffic per second over 20 simulation runs of 100000s each within one boxplot for each amount of nodes. When the certificates get renewed at fixed times, during these times lots of traffic is generated, while no traffic at all is generated throughout the rest of the simulation, except for the traffic caused by node churn. With the amount of services the generated traffic increases linearly. This is visible by looking at the quartile boundaries displayed in the boxes.

As opposed to this, in the simulations with random backoff in the certificate renewal times the generated traffic only has occasional peaks. The quartile boundaries remain below the 2,5 KB/s line even with 600 active services. The traffic still linearly increases, mainly due to outliers, but as it is well distributed over time. The coefficient is very low leading to significantly lower average traffic.

Fig. 3 shows that our proposed solution with the random backoff requires low bandwidth even for larger amounts of services.

V. Related Work

Most of the existing work focuses either on transport security or on authorization schemes such as OAuth [2]. Those works are complementary to ours and should be combined to achieve more security properties for IoT systems.

The authors of [3] look at standardization efforts for the IoT. The focus with such activities is on transport security. The authors come to the conclusion that providing our security properties for services is a highly relevant and complex task that must be addressed.

In mobile computing, Apple and Google secure their Apps with certificates similar to our approach [10]. As discussed in Sec. II the approaches are relevant and applicable to the IoT but not sufficient as they do not consider distributed execution environments.

Rivest discusses already in 1998 [11] that short certificate lifetimes could be an option for getting rid of Certificate Revocation Lists. However he sees the effort of reissuing the certificates. We solve this problem by introducing our automated certificate renewal process.

VI. Conclusion

Our solution provides IoT services with authentication, accountability and integrity from the developer to the runtime environment. It enables all participating entities to verify the correct operation of the preceding entities in the processing chain. The site-local autonomous certificate management enables the changing of secured attributes on all participating IoT nodes within guaranteed time limits. By enabling node-local verification of the secured data our solution scales well. This is especially relevant in the IoT where the connections between nodes are highly heterogeneous. Our solution is churn tolerant, which is also relevant for the IoT.

IoT architectures where third party developers provide services will be reality soon. Security has to be included in any IoT design from the beginning. We hope to raise awareness on this topic, and to contribute to a more secure future IoT.

REFERENCES

[1] C. Kolias, G. Kambourakis, A. Stavrou, and J. M. Voas, "DDoS in the IoT - Mirai and Other Botnets." *IEEE Computer*, 2017.

[2] A. Ouaddah, H. Mousannif, A. A. El Kalam, and A. A. Ouahman, "Access control in the Internet of Things - Big challenges and new opportunities." *Computer Networks*, 2017.

[3] S. L. Keoh, S. S. Kumar, and H. Tschofenig, "Securing the internet of things: A standardization perspective," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 265–275, 2014.

[4] C. R. P. dos Santos, J. Famaey, J. Schonwalder, L. Z. Granville, A. Pras, and F. De Turck, "Taxonomy for the Network and Service Management Research Field," *Journal of Network and Systems Management*, Jan. 2016.

[5] M.-O. Pahl, G. Carle, and G. Klinker, "Distributed Smart Space Orchestration," in *Network Operations and Management Symposium 2016 (NOMS 2016) - Dissertation Digest*, 2016.

[6] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–6.

[7] R. Want, "When Cell Phones Become Computers," *Pervasive Computing, IEEE*, vol. 8, no. 2, pp. 2–5, 2009.

[8] Apple Incorporated, "App Distribution Guide," Apple Inc., Oct. 2013.

[9] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280 (Proposed Standard), Internet Engineering Task Force, May 2008, updated by RFC 6818. [Online]. Available: http://www.ietf.org/rfc/rfc5280.txt

[10] Apple Incorporated, "IOS Security Guide," Tech. Rep., May 2016.

[11] R. L. Rivest, "Can We Eliminate Certificate Revocations Lists?" *Financial Cryptography*, 1998.