Authentication and Authorization for IoT Devices in Disadvantaged Environments

Sebastián Echeverría, Grace A. Lewis
Software Engineering Institute — TTG
Carnegie Mellon University
Pittsburgh, PA USA
{secheverria, glewis}@sei.cmu.edu

Dan Klinedinst
Software Engineering Institute — CERT
Carnegie Mellon University
Pittsburgh, PA USA
dklinedinst@cert.org

Ludwig Seitz
RISE Research Institutes of Sweden AB
Lund, Sweden
ludwig.seitz@ri.se

***Abstract*—Internet of Things (IoT) devices are increasingly being used to support operations in disadvantaged environments, such as those experienced by first responders, military, medics, and other field personnel. However, current IoT security efforts are mainly targeted at stable and connected environments such as home and industry. This paper presents an implementation for authentication and authorization of IoT devices in disadvantaged environments, based on an IETF proposal for authentication and authorization in resource-constrained environments (ACE). It includes capabilities for bootstrapping of credentials and token revocation to account for high-priority threats of node impersonation and theft, as well as limited connectivity.**

***Index Terms*—IoT, security, standards, disadvantaged environments**

I. Introduction

Internet of Things (IoT) devices are increasingly being used to support operations in disadvantaged environments, such as those experienced by first responders, military, medics, and other field personnel [1]. In addition to disconnected, intermittent, and limited (DIL) network connectivity, threats in these environments often include sabotage, capture, and impersonation, of both IoT devices and their clients [1] [2]. Therefore, strong yet decentralized authentication and authorization mechanisms are necessary to mitigate these threats.

A starting point for this work is Authentication and Authorization for Constrained Environments (ACE), an active working group in the Internet Engineering Task Force (IETF).[1] The charter for this group is to develop a standardized solution for authentication and authorization of IoT devices by extending OAuth 2.0 [3]. Two problems with the use of OAuth in resource-constrained environments are: (1) the protocol requires user intervention and/or online access to an authorization server, and (2) implementation technologies require resources (e.g., memory, storage, bandwidth) greater than what is available in these environments.

The basic protocol flow for ACE is shown in Figure 1. ACE adopts the OAuth client credentials grant flow (see 4.4 in [3]) and works as follows: A client *C* (e.g., mobile device) that wishes to access a resource at a Resource Server *RS* (e.g., IoT device) first contacts the Authorization Server *AS* to request an access token (1). The AS determines whether C has the right to the requested token, and if that is the case, creates and returns a token to C along with access information (2). The access information enables C to set up a secure connection with RS and to prove that it is the rightful owner of the access token (via a Proof-of-Possession (PoP) key). C then sends a resource request and the access token to the RS (3). After verifying that the token grants access to the requested resource and that C is the rightful owner of the token, RS responds to the request (4). The proposed ACE solution includes:

- Constrained Application Protocol (CoAP) as the communication protocol instead of HTTP. CoAP runs on top of UDP, and uses a compact message format that reduces overhead and message exchanges [4].
- Concise Binary Object Representation (CBOR) [5] for token encoding. This is a binary encoding designed for extremely small code size and fairly small message size.
- Application layer security using CBOR Object Signing and Encryption (COSE) [6], which adds object security to CBOR-encoded data for token protection.

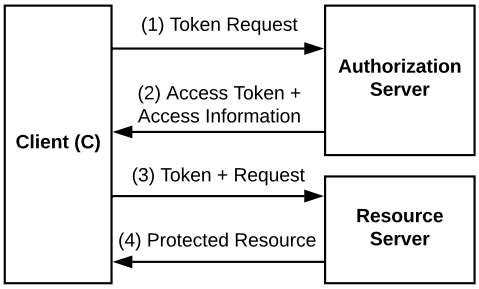

Fig. 1. ACE Basic Protocol Flow

In this work we evaluate, adapt, and implement a system based on ACE such that it is resilient to the characteristics and threats of disadvantaged environments, which are currently not addressed in ACE. Section II presents related work. In Section III we identify gaps in ACE for use in disadvantaged environments. Section IV presents the threat model that guided the architecture and implementation of the system described in Section V. Section VI presents the evaluation of the system. Finally, Section VII presents conclusions and next steps.

[1] https://datatracker.ietf.org/group/ace

978-1-5386-4980-0/19/$31.00 © 2019 IEEE

II. Related Work

Authentication and authorization in IoT environments have received a lot of attention in recent years, however many of these approaches are not aimed at disadvantaged environments. The Open Mobile Alliance has a standard for machine to machine (M2M) device management called OMA Lightweight M2M [7] that uses CoAP and DTLS (Datagram Transport Layer Security). Security is based on Access Control Lists (ACLs) with a relatively static structure of access rights that does not scale well with increasing numbers of users having different access rights. Vucnic et al. designed OSCAR [8], an approach for access control in the IoT using object security, based on secret keys for authorization to access resources. However, fine-grained access rights require the management of an increasing number of secret keys, which is difficult in disadvantaged environments. Neto et al. propose AoT [9], an authentication and access control scheme for the IoT device life cycle. AoT uses Identity- and Attribute-Based Cryptography with Attribute-Based Access Control. Their analytical results on the communication overhead reveal that their approach is not well suited for constrained networks with severely limited bandwidth and small packet sizes at the physical layer. Finally, Sciancalepore et al. [10] propose an adaptation of ACE called *OAuth-IoT*. They motivate their design based on the flawed assumption that the approach used by ACE would not work if the client is located outside the IoT network. In addition, their design makes the Gateway a trusted man-in-the middle between the client and the IoT device, thus creating a brittle security architecture in which the Gateway is a single point of failure and a high-value target for attacks.

III. Using ACE in Disadvantaged Environments

A. Bootstrapping of Credentials/Keys

Bootstrapping of credentials/keys is out of scope for ACE. ACE assumes that clients and RSs have previously securely exchanged credentials with the AS. This is not an oversight in ACE, but rather a consequence of the heterogeneity of IoT devices and pairing mechanisms. While acceptable in the context of home and potentially industrial environments, not including bootstrapping as an integral part of the process is a risk in disadvantaged environments given that node capture and impersonation are high-impact, high-probability threats [2]. We define bootstrapping as (1) granting clients and RSs access to the network, and (2) exchanging credentials to set up a secure channel.

In general, credentials can be delivered at different stages of the manufacturing/commissioning process, as shown in Table I. All these options assume that (1) a form of credential/key is printed on a QR code associated to the IoT device, and (2) the AS will be deployed in the field and potentially disconnected from the enterprise. We selected to implement the *During Deployment (Temporary PSK)* option described in Section V-A because it provides the flexibility and security needed in disadvantaged environments. We did not consider solutions based on raw public keys (RPKs) or certificates because these would increase message size.

B. Token Revocation

Due to the often chaotic nature of disadvantaged environments, it is important for clients and IoT devices to know if active tokens in their possession belong to compromised IoT devices and clients, respectively [2]. Token revocation, other than time-based via expiration times associated to tokens, is not implemented in ACE. The problem is that (1) compromised clients will have access to resources until token expiration time, and (2) clients will have access to compromised resources until token expiration time. To address this problem we developed a mechanism for periodic introspection between IoT devices and the AS, and between clients and the AS, as described in Section V-B.

IV. Threat Model

The threat model for the system is based on the actors defined by ACE: Client, AS, and RS. Using Microsoft Threat Modeling Tool [11], we developed separate diagrams for the two steps that are necessary for the secure interaction between ACE actors: *Pairing* and *Resource Access*. The tool applies the STRIDE threat model [12] to each element in the diagram (process, data flow, and data store) and provides a guided analysis of threats and mitigations. Pairing takes place between the Client and the AS, and between the RS and the AS. The goal of this step is to (1) grant the Client and RS access to the network that the AS is on, and (2) exchange credentials between the Client and the AS, and between the RS and the AS. Resource Access corresponds to the basic protocol established by ACE, in which the Client obtains tokens from the AS, that are then used to obtain data from the RS.

For Pairing, the tool identified 32 threats, of which 17 were considered not applicable. The remaining 15 threats were grouped and prioritized as shown in the top part of Table II. For Resource Access, the tool identified 46 threats, of which 19 were considered not applicable. The remaining 29 threats were grouped and prioritized as shown in the bottom part of Table II. Threat priority is based on characteristics of disadvantaged environments in which (1) node impersonation/capture is likely, (2) communications eavesdropping and tampering is likely, and (3) RS can manage sensitive data. All the threats were examined by a threat modeling expert on our team, evaluated for their applicability to the system, and implemented in the system described in Section V.

V. Architecture, Design, and Implementation

In this section we describe the architecture, design, and implementation of a system for authentication and authorization of IoT devices in disadvantaged environments, based on ACE, that addresses the gaps identified in Section III and the threats presented in Section IV.

A. Pairing Procedure

As stated in Section III-A, to support bootstrapping of credentials we selected the *During Deployment (Temporary PSK)* option. A device will always start up in ”pairing mode” and go back to this mode after a reboot. After executing the

TABLE I
CREDENTIAL DELIVERY OPTIONS

Option	Description	Advantages	Disadvantages
At Manufacturing Time	Hardware-based credentials (e.g., TPM) are embedded in the IoT device. Public-key-equivalent portion of the key is exposed on the QR code associated to the device. AS reads the QR code (trust via physical proximity) and adds the device credentials to its list of paired devices.	Hardware-based credentials are a strong method for addressing the threat of device impersonation.	Requires IoT devices that support hardware-based credentials.
Before Deployment	Device credentials are exchanged before deployment such that the AS is pre-paired to all its IoT devices. AS reads QR code for all devices and adds them to its list of paired devices. QR code does not have to "accompany" the device into the field.	Out-of-the box deployment in the field.	Additional devices cannot be added in the field (loss of flexibility).
During Deployment (Fixed PSK)	AS reads the QR code for the device and adds it to its list of paired devices. QR code associated to the device is the pre-shared key (PSK) that is used in ACE exchanges.	Pairing can be done in the field (flexibility).	(1) PSK used in ACE exchanges is available to anyone who has access to the QR code. (2) PSK remains constant throughout all deployments.
During Deployment (Temporary PSK)	QR code associated to the device is a randomly-generated PSK that is used for initial encryption of communication between the device and the AS. AS reads the QR code and generates a second PSK that is sent to the device for ACE exchanges. AS then adds the device to its list of paired devices.	(1) Pairing can be done in the field (flexibility). (2) Key used for ACE exchanges is generated during deployment only for that deployment. (3) New credentials can be generated for every new deployment.	IoT device would have to support the transfer and storage of new credentials.

pairing procedure, it will ignore further pairing requests until it is rebooted again. This is done for security and to avoid rogue ASs pairing to the device after deployment. The reboot needs to be secured in a physical manner to avoid that anyone that comes close to the device can perform a reboot. The pairing procedure consists of two main steps:

1. Channel Setup:
   - AS reads the QR code on the IoT device that contains the RS, using a built-in or attached camera.
   - AS decodes the QR code to obtain the 128-bit key.
   - AS establishes a secure channel using the key encoded in the QR code, using CoAP over IPv6 with the DTLS profile, with the QR code key as the DTLS PSK.
2. Credential Exchange:
   - AS generates a new 128-bit PSK for the RS.
   - AS sends the PSK to the RS over the secure channel, along with its ID.
   - RS stores the PSK, and replies with its ID and a list of the ACE scopes it provides. Scopes in ACE refer to the parameter that indicates what an access token authorizes, such as "read", "write", "open", or "close." We use the operation and the name of the resource provided. For example "r_temperature" indicates that the token can "read temperature."
   - AS adds the RS, the PSK, and the scopes to its list of paired devices.

B. Token Revocation Check

We used the introspection feature of ACE/OAuth [14] to support token revocation. Introspection is used when more information about a specific token is needed. This is commonly used by the RS when the token is not self-contained and only consists of an ID for a set of claims (i.e., information carried in an access token). By using introspection on the AS, the claims associated with the token can be obtained. A client can also make introspection requests if authorized by an AS, but it is not a common use case.[2]

We do not use introspection to obtain token claims, but rather to poll the AS to see if the tokens that the client and RS have are still valid and have not been revoked. Because an introspection response can be configured to not include any claims, but only to indicate if the token sent is still valid, we used this feature to implement polling of revoked tokens from both the client and RS to the AS. Because these requests will only work when the client or RS are in range of the AS, they periodically attempt to poll the AS until they are in range and are able to update their information about token validity. When a token needs to be revoked (e.g., RS or Client associated to the token has been compromised), the AS Admin marks the device or client as compromised in the AS user interface (UI), which in turn marks associated tokens as revoked in the AS internal database. The protocol to discover revoked tokens is:

1) RS or Client starts a thread that polls the AS every N minutes (configurable in UI) to see if one of its tokens has been revoked. This will only succeed if the AS is in range.
2) In each pass, for each token:
   a) Client or RS sends a request to the introspection endpoint of the AS, including the token.
   b) AS decrypts the token and checks whether it has been marked as revoked.
   c) AS replies to the Client or RS indicating whether the token is still active or not.
   d) Client or RS receives reply. If it indicates that the token is no longer active, it deletes it from its storage.

[2]Our recommendation for client introspection will be presented as an Informational RFC in the ACE Working Group at the next IETF meeting.

TABLE II
THREAT MODELING RESULTS

Category	Name	Description	P	Mitigation
Pairing				
Spoofing	Node impersonation	Attacker can impersonate any node to gain unauthorized access to nodes and obtain credential data that can be then used to access other nodes.	H	(1) For pairing between Client and AS we use our previous work that combines Identity-Based Encryption (IBE) with physical proximity and visual confirmation as out-of-band channels [2]. (2) For pairing between RS and AS we associate each RS with a QR code that contains a PSK. In addition, RS can only be paired once; it has to be rebooted to be re-paired. However, QR code is externally visible and could be read by nodes other than a valid AS. Higher levels of protection would require physical mechanisms to lock the QR code so that it is not visible when in the field.
Information Disclosure	Sniffing data flows	Attacker can sniff data flows between Client and AS and between RS and AS to obtain credential data that can be used to access other nodes.	H	(1) For pairing between the Client and AS we use our previous work [2]. (2) For pairing between RS and AS, the key encoded in the QR code is used to encrypt the pairing request using DTLS, which provides confidentiality protection.
Tampering	Tampering data flows	Attacker can tamper data flow to obtain unauthorized access to nodes.	H	(1) For pairing between Client and AS we use our previous work [2]. (2) For pairing between RS and AS, the key encoded in the QR code is used to encrypt the pairing request using DTLS, which provides integrity protection.
Information Disclosure	Unauthorized access to credential data stores	Attacker may gain access to credential data stores on any node during the pairing process and use them to access other nodes.	H	For client and AS, implementations can encrypt credential data stores. RS credential data stores will likely not be encrypted because of resource constraints and lack of a human operator to enter passwords. During pairing, there are no credentials stored on the RS because they are all deleted during the reboot required for pairing. However, QR code is externally visible and could be read by nodes other than a valid AS. Higher levels of protection would require physical mechanisms to lock the QR code so that it is not visible when in the field.
Elevation of Privilege	Execution of code other than authorized operations	Malicious client (Client or RS) could obtain access to run other code on the AS and obtain credential information or compromise the AS.	M	The pairing protocol implementation contains code to sanitize inputs, such that arbitrary code cannot be passed in the pairing messages.
Resource Access				
Spoofing	Node impersonation	Attacker can impersonate any node to gain unauthorized access to nodes and obtain sensitive information.	H	The communication between ACE actors takes place using DTLS with client authentication. Ciphertext inside the PoP token prevents the client from impersonating either the AS or RS. Client and RS token revocation were implemented to deal with situations in which there is belief that an RS or client has been compromised.
Tampering	Tampering data flows	Attacker can tamper data to obtain unauthorized access to nodes.	H	The communication between ACE actors takes place using DTLS to provide integrity protection.
Information Disclosure	Sniffing data flows	Attacker can sniff data flows between Client and AS and between Client and RS to obtain data that can be used to access other nodes.	H	The communication between ACE actors takes place using DTLS to provide traffic encryption.
Information Disclosure	Unauthorized access to credential data stores	Attacker may gain access to credential data stores on any node and use them to obtain access to other nodes.	H	For the client and the AS, implementations can encrypt the credential data stores. RS credential data stores will likely not be encrypted because of resource constraints and lack of a human operator to enter additional passwords. Higher levels of protection require hardware-encoded keys, with the tradeoff being deployment flexibility. Highest priority is to protect the AS credential data store because it would provide access to credentials of all paired RSs.
Elevation of Privilege	Execution of code other than authorized operations	Malicious client could obtain access to run other code on the AS and obtain credential information or compromise the AS, or run other code on the RS and obtain sensitive information or compromise the RS.	M	The resource access protocol implementation contains code to sanitize inputs such that arbitrary code cannot be passed in the resource access messages.
Elevation of Privilege	Elevation using impersonation	(1) This is a limitation of OAuth 2.0. The AS can issue access tokens to itself and impersonate clients, a potential problem for any federated system. (2) If a Client obtains an access token that is valid for several RS, one of these RS could use the token to impersonate the client towards the other RS.	M	For the first point, there is no mitigation as the the premise in OAuth is that the AS is a trusted entity. A potential mitigation is to bind the token to an asymmetric PoP key, which is an option in the ACE framework (similar to pass the hash). For the second point, the mitigation in our implementation is to generate a separate token for each RS.
Information Disclosure	Cross-Site Request Forgery [13]	Because of the use of OAuth it might be possible for an attacker to use node information to execute actions on other nodes.	L	Mitigation is to generate a separate token for each RS. At the moment, CoAP does not allow redirects. However, CoRE WG is considering a CoAP redirect protocol, which would make this kind of attack feasible.

P: Priority (H=High, M=Medium, L=Low)

C. Implementation

A dynamic view of the system is shown in Figure 2. There are three main components that correspond to the three ACE actors: IoT Client, AS, and RS. For the RS, we implemented an unconstrained RS and a resource-constrained RS. Both RSs are functionally equivalent, but are implemented with different constraints. In particular, the resource-constrained RS was designed to work on Class 2 devices [15], such as the Texas Instruments (TI) CC2538-based boards that we used for experimentation.[3] Class 2 Devices are limited to ∼50KB for data (i.e., memory) and ∼250KB for code (i.e., storage). The unconstrained RS, as well as the other components, were written in Java, and the constrained RS was written in C.

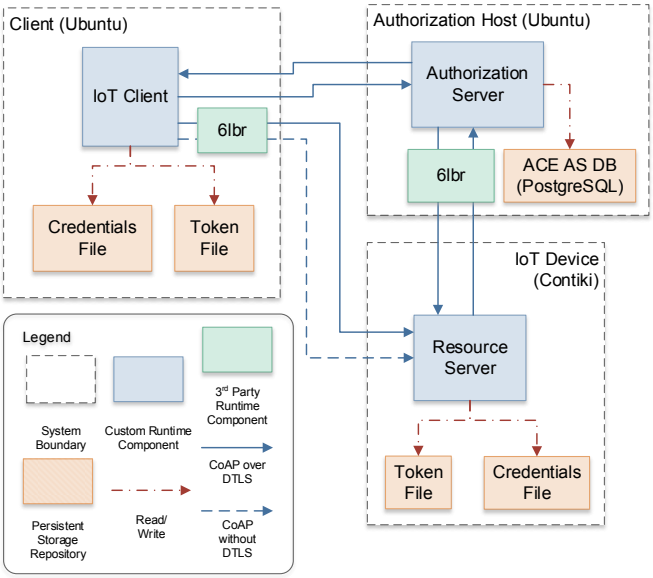

Fig. 2. Dynamic View of the System

We used the open-source ACE-java library[4] as a starting point for the system, which supports most of the functionality described in the ACE framework specification and associated DTLS profile. We extended this library to add support for pairing and periodic token revocation checks. For the constrained RS, we used the 6lbr Contiki fork project[5] as the starting point because it already included needed capabilities, such as CoAP over DTLS using tinydtls.[6] We had to implement the ACE functionality ourselves because there is no similar ACE library for constrained platforms.

As shown in Figure 3, for communicating with the RS, we used 802.15.4 radios. In particular, we used the TI CC2531EMK USB dongles as the physical adapters, and instances of 6lbr on each computer to handle conversion from plain IPv6 to 6LoWPAN [16]. All communication on top of that was done using the CoAP protocol and the DTLS profile for security. In the case of the unconstrained components, they were developed and tested on commodity laptops. The constrained RS was developed and tested on both a Raspberry Pi[7] and the TI CC2538 board.

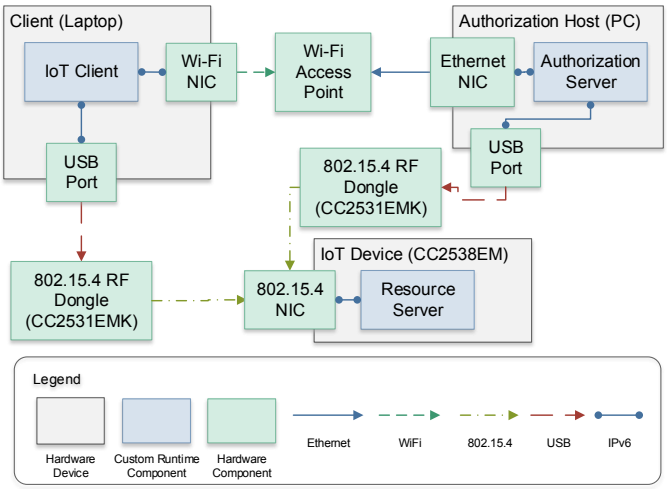

Fig. 3. Physical View of the System

VI. Evaluation

A. Resource Consumption

We evaluated our constrained RS implementation to see if it met the constraints of a Class 2 device (i.e., ∼50KB for data and ∼250KB for code) [15]. We compared a minimal version of 6lbr with a networking stack (Min Binary) to a version that also included the minimum requirements for ACE (ACE Binary): Erbium (CoAP server), CBOR parsing, COSE decryption, tinydtls, and the ACE code. The sizes of the resulting binaries (not including token or configuration storage space) and memory usage are shown in Table III. For the TI CC2538, which is a Class 2 device, the data size constraint (ACE RAM Usage) is satisfied, but the code size constraint (ACE Binary) is not. However, as noted by the 23.64% size difference between ACE Binary and Min Binary, the size of the ACE code is not the main contributor to code size. Reducing the code size to ∼250KB would require further eliminating unnecessary libraries and functions from 6lbr.

TABLE III
Resource Consumption

Platform	Min Binary (bytes)	ACE Binary (bytes)	Increase (%)	ACE RAM Usage
Raspberry Pi	995,783	1,452,153	45.83	2.3-3.5 MB
TI CC2538	286,660	354,432	23.64	< 32 KB

We also analyzed the size of the payloads and found it quite small and therefore appropriate for operation in DIL environments. The size of a sample unencrypted CBOR Web

[3]http://www.ti.com/product/CC2538
[4]https://bitbucket.org/lseitz/ace-java
[5]https://github.com/cetic/6lbr
[6]https://github.com/cetic/tinydtls
[7]https://www.raspberrypi.org/products/raspberry-pi-1-model-b-plus/

Token (CWT) is 74 bytes and the size of a sample COSE encrypted CWT is 113 bytes. Finally, we were interested in measuring roundtrip times for requests. We timed IPv6 CoAP requests on the Raspberry Pi in order to remove network latency. A pairing request using DTLS took on average 300 ms. A resource access request posting a CWT (no DTLS, but includes decryption) took on average 700 ms.

B. Vulnerability Analysis

We conducted vulnerability analysis using simple attack trees to determine potential attack vectors [17]. The resulting attack tree is shown in Figure 4. We identified four primary attack vectors: a compromised AS, Client, or RS; or a rogue node on the network. A compromised AS or RS, or a rogue node, could lead to (1) unauthorized access to the RS data or controls, or (2) unauthorized changes to the data provided to the Client. A compromised Client could only lead to the former, as it would not make sense for a Client to change data sent to itself. We mitigated the attack vectors as follows:

1) A compromised AS is mitigated by removing its pairing key from any Clients or RSs that trust it.
2) A compromised RS or Client has its pairing key removed from the trusted list on the AS.
3) An RS cannot be spoofed because a rogue RS would not have a key to decrypt the COSE-encrypted PoP keys.
4) Man-in-the-middle attacks are prevented by DTLS.
5) The only non-DTLS connection is posting a token to the authz-info endpoint. This token is encrypted by COSE so rogue devices or Clients cannot view the key.

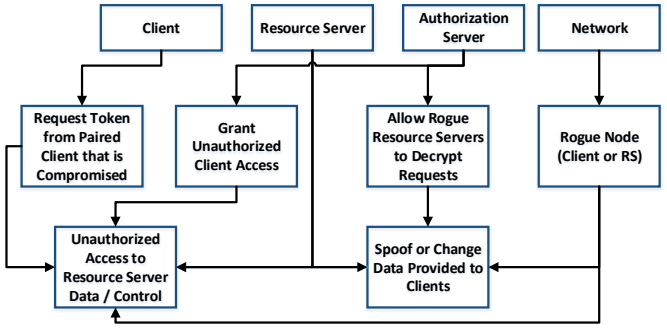

Fig. 4. Attack Tree

VII. Conclusions and Next Steps

We presented an implementation for authentication and authorization of IoT devices in disadvantaged environments that includes capabilities for bootstrapping of credentials and token revocation to account for threats and characteristics of these environments. Threat modeling was conducted as part of the design and architecture process. Resource consumption data was gathered and vulnerability analysis was conducted as part of the evaluation. All code and supporting documentation is available at https://github.com/SEI-TTG/ace-client/wiki. Next steps for our work include (1) creating a mesh of devices over an IEEE 802.15.4 wireless network and adding routing between peers (including extension of the threat model to account for untrusted intermediaries) to provide better guarantees for periodic introspection requests and better understand resiliency and scalability, and (2) further optimizing the constrained RS implementation for a Class 1 device.

Acknowledgments

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center (DM18-1269). Ludwig Seitz worked on this document as part of the Celtic Plus project CyberWI, with funding from Vinnova.

References

[1] P. Fraga-Lamas, T. Fernández-Caramés, M. Suárez-Albela, L. Castedo, and M. González-López, "A Review on Internet of Things for Defense and Public Safety," *Sensors*, vol. 16, no. 10, p. 1644, 2016.

[2] S. Echeverría, D. Klinedinst, K. Williams, and G. A. Lewis, "Establishing Trusted Identities in Disconnected Edge Environments," in *Edge Computing (SEC), IEEE/ACM Symposium on*. IEEE, 2016, pp. 51–63.

[3] D. H. (ed.), "The OAuth 2.0 Authorization Framework," Internet Engineering Task Force (IETF), Request For Comments (RFC) 6749, October 2012, http://www.ietf.org/rfc/rfc6749.txt.

[4] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol (CoAP)," Internet Engineering Task Force (IETF, Request for Comments (RFC) 7252, June 2014, http://www.rfc-editor.org/rfc/rfc7252.txt.

[5] C. Bormann and P. Hoffman, "Concise Binary Object Representation (CBOR)," Internet Engineering Task Force (IETF), Request For Comments (RFC) 7049, October 2013, http://www.ietf.org/rfc/rfc7049.txt.

[6] J. Schaad, "CBOR Object Signing and Encryption (COSE)," Internet Engineering Task Force (IETF), Request For Comments (RFC) 8152, July 2017, http://www.ietf.org/rfc/rfc8152.txt.

[7] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification," Open Mobile Alliance, Technical Specification OMA-TS-LightweightM2M-V1_020131105-D, November 2013.

[8] M. Vucnic, B. Tourancheau, F. Rousseau, A. Duda, L. Damon, and R. Guizzetti, "OSCAR: Object Security Architecture for the Internet of Things," *Ad Hoc Networks*, 2015.

[9] A. Neto, A. Souza, I. Cunha, M. Nogueira, I. Nunes, L. Cotta, N. Gentille, A. Loureiro, D. Aranha, H. Patil, and L. Oliveira, "AoT: Authentication and Access Control for the Entire IoT Device Life-Cycle," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*. ACM, November 2016, pp. 1–15.

[10] S. Sciancalepore, G. Piro, D. Caldarola, G. Boggia, and G. Bianchi, "OAuth-IoT: an Access Control Framework for the Internet of Things Based on Open Standards," in *Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, July 2017.

[11] Microsoft Corporation, "SDL Threat Modeling Tool," 2015. [Online]. Available: https://www.microsoft.com/en-us/sdl/adopt/threatmodeling.aspx

[12] ——, "The STRIDE Threat Model," 2009. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)

[13] OWASP, "Cross-Site Request Forgery (CSRF)," 2018. [Online]. Available: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[14] J. R. (ed.), "OAuth 2.0 Token Introspection," Internet Engineering Task Force (IETF), Request For Comments (RFC) 7662, October 2015, http://www.ietf.org/rfc/rfc7662.txt.

[15] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," Internet Engineering Task Force (IETF), Request For Comments (RFC) 7228, May 2014, http://www.ietf.org/rfc/rfc7228.txt.

[16] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 8002.15.4 Networks," Internet Engineering Task Force (IETF), Request For Comments (RFC) 4944, September 2007, http://www.rfc-editor.org/rfc/rfc4944.txt.

[17] B. Schneier, "Attack Trees," *Dr. Dobb's Journal*, vol. 24, no. 12, pp. 21–29, 1999.