Authentication for the Web of Things

*Submission to the **W3C Workshop on the Web of Things**, 25–26 June 2014, Berlin, Germany*

Author: Oliver Pfaff, Siemens AG, Corporate Technology, oliver.pfaff@siemens.com

Introduction

Authentication is a generic need. It appears in ubiquitous manner whenever valuable resources are at stake. This is axiomatic and also holds for the Web of Things (short: WoT). This text examines authentication for WoT by considering functional roles in authentication systems and resulting patterns.

Authentication System Roles

Following functional roles appear in authentication systems:

- **Claimant**: entity (e.g. user/Web server) that claims an identity (by presenting e.g. username/ certificate) and that has to provide evidence:
  - Request/response: the requestor with respect to requests; vice versa for responders
  - Messaging: the publisher
- **Verifier**: entity (e.g. Web server/browser) that accepts identity claims and verifies provided evidence (e.g. password/proof-of-possession signature). This is done according an authentication protocol:
  - Request/response: the requestor with respect to requests; vice versa for responders
  - Messaging: the subscriber
- **Relying party**: entity (e.g. servlet component/user) that consumes verified identity and performs depending tasks (e.g. present the contacts of a user/shop online):
  - Request/response: the originator of the actual response
  - Messaging: the recipient of the actual message

The roles of verifier and a relying party do not have to coincide: verification of claimed identity is often done in another component than consumption of identity. This can refer to another process or system owned by same organization or another organization.

Authentication System Patterns

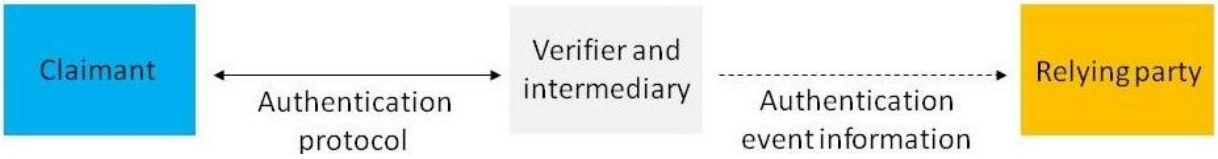
This chapter presents atomic ways of allocating these roles when building authentication systems.

Direct

- Relying party and verifier roles coincide: relying parties authenticate claimants themselves i.e. they terminate an authentication protocol on verifier side[1].
- Relying parties (resp. their backend validation services) need to hold verification data for all possible claimants (actuals [shared secrets, shared secret keys] resp. root values [public keys]).
- Example: HTTP Basic for request/user authentication, SSL/TLS for response/server authentication

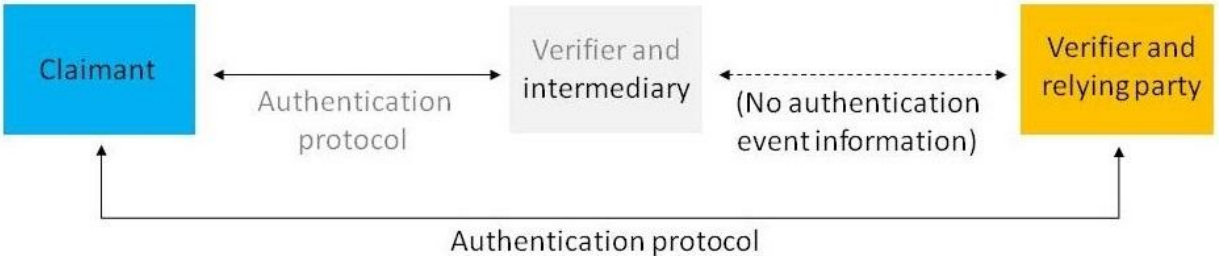
[1] They may delegate some tasks to validation services e.g. LDAP/RADIUS. Claimants do not become aware of that.

Inline Trusted Third-Party



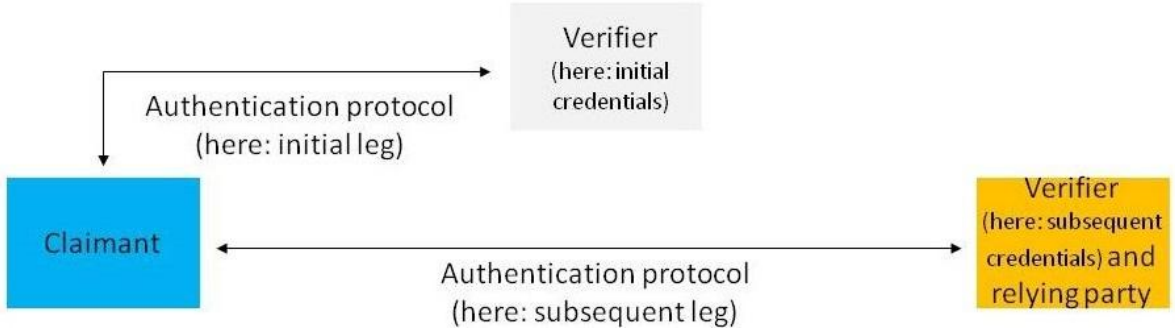
- Relying party and verifier roles do not coincide: relying parties do not terminate authentication protocols by themselves. They receive authentication event information. This data is created by verifiers that are inline in the processing chain i.e. process requests/responses or messages.
- Relying parties are decoupled from authentication scheme as well as user population details.
- Example: presentation-oriented Web application in front of service-oriented Web application that serves protected resources e.g. Web frontends to Cloud-based file storage

Inline Untrusted Third-Party



- Relying party and verifier roles do coincide: relying parties directly authenticate claimants. Inline third parties may also act as verifiers and authenticate claimants for their own purposes. They do not report about own authentication events to relying parties. Such an authentication process is independent/decoupled from the other.
- Relying party (resp. a backend validation service) needs to hold verification data for all claimants.
- Example: as above with Web application ownership by different organizations

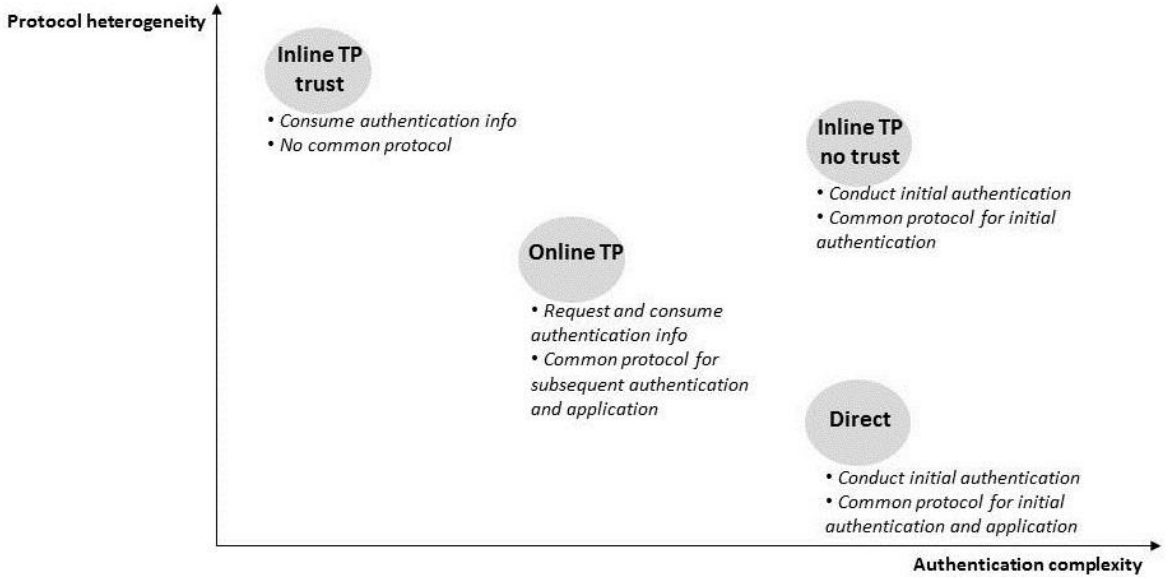
Online Trusted Third-Party



- Relying party and verifier roles do semi-coincide: relying parties do participate in authentication protocols but do not perform initial authentication by themselves. They request and receive authentication event information. This data is created by verifiers that are not inline in the processing chain and is submitted by claimants. Hence this data needs verification.
- Relying parties are decoupled from initial authentication scheme and user population details[2].
- Example: Kerberos for the proof-of-possession approach, Web SSO for the bearer approach

[2] Unlike the 'Inline Untrusted Third Party' case, relying parties may challenge for authentication on own behalf.

The following illustration shows resulting tradeoffs by highlighting tasks for relying parties:



Real-life authentication systems combine the identified patterns. For instance, Google and Facebook allow third-party access to personal resources e.g. contacts/timeline via Web APIs:

- Composite applications of other providers that call Google/Facebook Web APIs present 'Inline Untrusted Third Parties'. The Web APIs are protected by OAuth (authorization code grant).
- The OAuth authorization servers externalize user authentication/consent to presentation-oriented login/accounts Web applications that presents 'Online Trusted Third-Parties'.

Users, User Agents Access Networks and Identity Providers

In order to draw conclusions for WoT it is important to make assumptions about users, their user agents, access networks and preferred identity providers. This text makes following assumptions:

- **Users**: normal people distinguished into individual users acting on their own (e.g. smart home/ mobility) or employees/contractors acting on behalf of an organization (e.g. smart industry)
- **User agents**: Web/mobile browser, browser-based or native mobile apps i.e. HTTP clients
- **Access network:** via Internet - applications called by these user agents need to be public-facing
- **Identity providers**: almost all users are served by existing identity providers e.g. Google (Android users) or Facebook for individual users, corporate authentication systems/directories for enterprise users. Whether and how these providers qualify for a WoT deployment is an open question. It depends on several aspects including user preferences, provider assurances and value of resource assets. However raising user populations from scratch is a difficult proposition.

These assumptions are motivated by current practices from WoT deployments in automotive: *car-as-callee* refers to a common WoT scenario encompassing use cases such as control climate, find vehicle, flash light or blow horn (to locate a car), lock/unlock doors. This scenario is characterized by:

- Users: arbitrary car owners (subject to manufacturer, model, manufacturing year)
- User agents: native mobile apps running on smart-phones/tablets (e.g. MyBMWRemote)
- Access network: smart-phones/tablets connected to Internet via cellular network or WLAN
- Identity providers: manufacturer-specific customer repositories and authentication systems that usually integrate social identity providers such as Google or Facebook

Protocol Options

A number of protocols are considered in WoT for calling devices and interacting with them. This includes:

- AMQP (Advanced Message Queuing Protocol, OASIS standard)
- CoAP (Constrained Application Protocol, IETF draft [expired])
- DDS (Data Distribution Service, OMG specification)
- HTTP (Hypertext Transfer Protocol, IETF standard) according REST
- MQTT (Message Queue Telemetry Transport, OASIS draft)
- STOMP (Simple Text Oriented Messaging Protocol, community effort)
- XMPP (eXtensible Messaging and Presence Protocol, IETF RFC 6120 et al.)

The suitability of these protocols for a WoT scenario depends on various aspects. Probably there is no single protocol that fits all needs. Under the assumptions made above, HTTP plays a special role: it facilitates exchanges between user agents and devices without application protocol translations.

Authentication with Non-HTTP

This case implies that there is no common application protocol between the considered user agents and devices. According the identified patterns, user authentication needs to be done with the 'Inline Trusted Third-Party' or 'Inline Untrusted Third-Party' pattern. In the first case devices are concerned with:

- *Authenticating the requestor resp. publisher (here: intermediary)*: this can be done by means of the 'Direct' or the 'Online Trusted Third Party' pattern. The choice is subject to concerns such as number of interacting components and their lifecycle esp. change dynamics.
- *Obtaining information about an event of user authentication*: various form factors may be used for this task starting with plain, unprotected data (for e.g. device networks that are trusted or low-value assets). Security tokens issued by the intermediary or another system component present a further option. The form-factor selection depends on device and network exposure i.e. given attack surfaces as well as resource values that are at stake. Independent from the form factor standards are needed to express and bind such data. This seems to be a white-spot.

In case of the alternative pattern 'Inline Untrusted Third-Party', devices would need to get back to users (owners/admins) and authenticate them by themselves (rather than trusting else to do that) - possibly also asking them for consent. This becomes a need when devices would expose open APIs allowing an external developer community to create and (automatically self-)register clients resp. publishers that can interact with devices on behalf of users (owners/admins). This is a common use case in the API world. In WoT this scenario is not yet encountered as a common use case. For this reason an implementation option according the 'Inline Untrusted Third-Party' is not considered further here.

Authentication with HTTP

In this case communications are conducted according request/response and all above mentioned authentication system patterns are feasible. But not all are advisable as the following argument shows.

There is only a single option for response authentication in this case: this is addressed by SSL/TLS server authentication. This implies X.509 server certificates and private keys which creates a divide:

- WoT scenarios with mid to high-end devices in small numbers may be able to cope with this
- WoT scenarios with low-end devices in large numbers do probably present a no-match

So in case of large number of low-end devices, SSL/TLS cannot be used as authentication protocol between user agents and devices: user authentication may be done according the 'Inline Trusted Third-Party' pattern. From device perspective this results in the following tasks:

- *Authenticating the requestor (here: intermediary)*: this can be addressed by the 'Direct' (e.g. HTTP Basic) and the 'Online Trusted Third Party' (OAuth, client credentials grant) pattern.
- *Obtaining information about an event of user authentication*: this is usually addressed through custom HTTP headers. Depending on given attack surfaces and resource values at stake, the provided information may be in plain form or in form of bearer security tokens issued by the intermediary resp. by another component such as a Web SSO system.

For authentication between user agents and intermediaries the well-known means and practices of authentication for the Web apply.

Conclusions

The protocols which are employed to call connected devices and interact with them determine the level of WoT readiness in addressing authentication needs.

WoT deployments that use HTTP for this purpose will be able to use a wide variety of authentication schemes according various authentication system patterns. Since a few years the inventory of authentication mechanisms for HTTP grows significantly - after almost nothing happened for a long period of time. This is due to new use cases around mobile, social and Cloud whose common currency is Web APIs. They are protected by requiring HTTP requests to present OAuth tokens. Now, OAuth is a zoo in itself that covers a variety of security use cases including persisted login for native mobile apps and user-managed access for composite applications. WoT can take advantage of this evolution which is meant to mean: there is prior art that will help WoT to deliver.

In the non-HTTP case, authentication appears to be under-illuminated. The mentioned protocols usually do approach authentication (esp. accomplishing user-to-device authentication and vice versa) according to one or more of following recipes:

- *Get backing through naïve DIY*: define protocol fields for the transfer of username/password
- *Get backing by pointing to else*: refer to other protocols such as SSL/TLS or SASL
- *No backing at all*: not address request or message authentication

This does not match the needs of the important scenario where HTTP clients are used to engage in WoT: this can be addressed by the patterns 'Inline Trusted Third-Party' or 'Inline Untrusted Third-Party'. Rather than focusing on initial authentication schemes there is a need for standard ways to transfer authentication event information (possibly also related authorizations) from intermediaries to devices:

- In the HTTP space this used to be a white-spot until a few years ago. It is being addressed by current initiatives in Web security (including OAuth, JOSE resp. JSON Web Tokens). These mechanisms may provide a blueprint for non-HTTP protocols but are not a solution for them.
- In the security protocol space (outside HTTP) this can be addressed by means of the TLS authorization extensions. This mechanism is standardized (RFC 5878) but has not yet seen widespread implementation and deployment support.