LYLE, J., MONTELEONE, S., FAILY, S., PATTI, D. and RICCIATO, F. 2012. Cross-platform access control for mobile web applications. In *Proceedings of the 2012 IEEE international symposium on policies for distributed systems and networks (POLICY 2012), 16-18 July 2012, Chapel Hill, USA*. Los Alamitos: IEEE Computer Society [online], pages 37-44. Available from: https://doi.org/10.1109/POLICY.2012.9

Cross-platform access control for mobile web applications.

LYLE, J., MONTELEONE, S., FAILY, S., PATTI, D. and RICCIATO, F.

2012

© 2012 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

OpenAIR @RGU

This document was downloaded from https://openair.rgu.ac.uk

SEE TERMS OF USE IN BOX ABOVE

DISTRIBUTED UNDER LICENCE

Cross-platform access control for mobile web applications

John Lyle*, Salvatore Monteleone†, Shamal Faily* Davide Patti† and Fabio Ricciato‡
* Department of Computer Science, University of Oxford. *first.last@cs.ox.ac.uk*
‡ Innovation and Industry Relations, Telecom Italia. *fabio.ricciato@telecomitalia.it*
† Department of Electrical, Electronic and Computer Engineering (DIEEI), University of Catania. *first.last@dieei.unict.it*

***Abstract*—Web browsers are a common platform for delivering cross-platform applications. However, they currently fail to provide consistent access control for security and privacy sensitive JavaScript APIs, such as geolocation and local storage. This problem is exacerbated by new HTML5 APIs and the increasing number of personal devices people own and use. In this paper we present the *webinos* platform which aims to provide a single, cross-device policy system for web applications on a wide range of web-enabled devices including TVs, smartphones, in-car systems and PCs. *webinos* solves the existing deficiencies in web authorisation by introducing the concept of a *personal zone*, the set of all devices and services owned by a particular user. All devices in this zone can synchronize their access control policies through interoperable middleware and can create flexible rules which may refer to an individual user, device or the entire zone. We provide details of the architecture and explain how our experience during design highlighted several conceptual challenges.**

I. Introduction

Web applications are gaining access to an increasing amount of important functionality. Thanks to growing support for new HTML5 standards, such as JavaScript APIs for geolocation data, sensors and local storage, web applications have the potential to replace native applications in many situations [1]. Proponents of web applications will also claim that there is a potential security advantage: the web browser provides a sandbox, isolating each page from the local device. This compares favourably with relatively unrestricted native code.

However, the web browser (or web runtime) was not originally designed for complex applications, and its sandbox security model must be broken to accommodate the new JavaScript APIs. Unfortunately, HTML5 API specifications [2] do not provide detail about how authorisation should be implemented, beyond a requirement for consent and revocation, leaving this as an implementation option for each browser or user agent. Considering the number of new APIs under development – access to cameras, address books, calendars, and network information [3] – as well as the wide range of security and privacy implications they have for both individual and corporations, there is a pressing need for new standards for web application access control.

Any access control system for web applications must take into consideration the fact that people own and use several different devices, often at the same time. More types of device are becoming web-enabled, including smartphones, tablets, in-car systems (such as the BMW ConnectedDrive [4]), games consoles and smart TVs. Each will need to implement authorisation for new HTML5 API requests, taking into account the unique user interface and interaction patterns that the device offers. For example, an in-car system might be voice activated, and notify the user of a new access control request through audio. A television might be more context-sensitive, changing interaction method depending on whether the user was watching a movie or casually channel-hopping. Moreover, the rise of 'second screen' companion applications – which allow TV-watchers to interact with the programme through their smartphones and tablets – might enable more interesting authorisation decisions affecting both devices. It also seems reasonable that any access control decision made by a user on one device might also be synchronised with others. For example, if a user grants a web application permission to access the address book on their smartphone, they probably want to allow the same application access to the address book on the user's PC. Web application access control must therefore be flexible enough to support different devices and interactions, while allowing the synchronisation of user policies between each device.

Browsers are beginning to support synchronization of access control decisions (Google Chrome, for example), but these fail to account for the many different contexts of use associated with each device. For example, a 'prompt every time' policy for geolocation data may be appropriate on a smartphone – where current location changes – but would be irritating on a TV. There are also many ways of notifying mobile users about privacy events [5] and the most appropriate method will be context sensitive. These requirements are difficult to address with ad-hoc browser settings, which only allow limited authorisation decisions and synchronisation options.

We assert that users require, but currently lack, a common way of managing with web application permissions. Standard, cross-device access control policies, which can address web APIs while also being transferable to different devices, are needed. This would enable devices to create custom user interfaces but still produce interoperable authorisation decisions, which could then be synchronised between devices. This, in turn, would let users make access control decisions infrequently (where the same decision applies to multiple devices) but would also allow decisions based on device-specific interactions and context.

In this paper we describe the architecture and implementation of *webinos*: a cross-device web application platform. *webinos* is designed to give web developers access to a standard set of APIs for device features, as well as providing a common policy model for access control. It also aims to give users a 'seamless' experience by providing better connectivity between devices. The platform is targeted at four domains: PC, mobile, in-car and smart TVs. We have developed an XACML-based system which provides a single policy enforcement mechanism for a user's *personal zone* – the devices they own and use. Our main contributions are: the introduction of a personal zone architecture, the definition of a modified XACML policy system to support this abstraction on all devices, and several observations from design experience.

We begin in Section II by discussing the state of the art in access control and policy enforcement for mobile web applications and web widgets. In Section III we describe the requirements for a personal, cross-device authorisation system and give an introduction to the *webinos* personal zone model. Following this, Section IV describes the *webinos* policy framework and section V covers four of the conceptual challenges faced during design and development. Finally, in section VI we conclude.

II. Background

A. Web applications, widgets and browser security

A web application in a mobile context is a web page or collection of web pages delivered over HTTP which uses server or client-side processing to produce an application-like experience within a web browser [6]. In comparison, web widgets are interactive applications for displaying and/or updating local or remote data, packaged to facilitate downloads to a workstation or mobile device [7]. As such, widgets are similar to web applications but are packaged for installation.

Web browsers support a sandbox model of security, isolating each web application from the device. When a web application wants to break this sandbox, it requires either an implicit user consent based on an action (e.g. file uploads through clicking on a button and selecting a file) or explicit consent through some form of prompt (e.g. the geolocation API). This decision may be remembered by the browser for subsequent requests. On most browsers, saved explicit consent decisions can be revoked through a settings page. Web applications are identified by *origin* – their DNS name, port and protocol – and JavaScript on any one origin cannot (in general) access any other. Consent decisions, however, are often stored based on hostname [8].

Widgets are run in a *web runtime* which is similar to a browser. Widgets are packaged as zip files, containing (among other things) a configuration file listing the APIs they request access to. These can be used as permission requests in a similar manner to Android application manifests. Widgets can be identified by namespace and id attributes declared in the configuration file, and widget packages can be signed by their author. Widgets are not restricted by the same origin policy, but may use the Widget Access Request Policy instead [9].

B. Mobile applications and access control

The most popular native mobile application systems are Google Android and iOS. Android takes an 'all-or-nothing' approach to authorisation; this involves developers specifying a mandatory access control policy for APIs in the application manifest [10]. At install time, users either agree to grant all requested permissions or none at all. On iOS, applications are granted access to the full system by default, although users are able to enable or disable access to geolocation data.

The Android policy system has been criticised for being ineffective, although recent results suggest that application permissions are a valuable security mechanism [11]. Several attempts have been made to modify how policies are specified and used. A common complaint is that Android permissions cannot be individually allowed or denied, a feature provided by MockDroid [12]. Apex [13] does the same, as well as supporting controls such as limiting the number of times a permission may be used. The CRePe system [14] allows the specification of fine-grained context-dependent policies, which may be based on time, location and the originator of the policy. CRePe also supports optional notification that a particular context has been activated. Reddy et al. [15] make the observation that Android policies are *resource-centric* rather than *application-centric*, and that security would be better served if applications could request access just to common functionality such as 'scan barcode' rather than the whole camera. This is analogous to program-language security features, where only certain methods are made public to external callers. Finally, SEAndroid has been proposed to better sandbox applications, confine privileged daemon processes and provide a central, analysable system policy [16].

C. Bridging the gap: web application security frameworks

Web applications and widgets require standardised APIs for accessing device features, and these are only slowly being implemented by browsers. As a result, several initiatives have been started to speed up standardisation and create cross-platform web application runtime environments. These have also included security frameworks.

The BONDI security framework for web applications was created by OMTP, a consortium of operators and handset manufacturers. OMTP was eventually enlarged and renamed as the Wholesale Application Community (WAC). The WAC 2.0 framework [17] combined BONDI with work by the W3C Device APIs and Policy Working Group [3]. BONDI uses XACML (eXtensible Access Control Markup Language) to specify widget access control policies. All widgets requesting access to WAC APIs are mediated by the XACML policy enforcement system. XACML is a general-purpose access control policy language based on subjects, resources, actions and conditions [18]. XACML also defines a reference architecture for policy control and enforcement as well as a message schema. To better situate XACML for a mobile environment, BONDI reduced the number of allowed expressions and specified an appropriate dictionary.

A problem with the WAC model is the role of the handset user as a policy authority. Once an application has been granted access to a certain device capability then it can use, transmit, and share it without any further control or restriction. Work by the EU FP7 PrimeLife project [19] addressed this issue by developing tools to protect user data and manage privacy requirements; these tools included a policy language for privacy and data protection based on XACML [20].

Finally, the Chrome browser also supports up-front permission requests by installable web applications (equivalent to widgets) downloaded from the Chrome Web Store. Permissions are split into high, medium and low-risk categories, with APIs for accessing geolocation and browser history marked as 'low'. Geolocation may also be requested at runtime in the same manner as normal web pages. The Chrome browser supports synchronisation of settings between different devices when the user has logged in with their Google user account. However, the privacy and security implications of synchronising access control decisions and other data with a central provider such as Google are considerable. Users may need assurance of the integrity and confidentiality of their cloud-based data [21], and may wish to chose a synchronising host that they trust. This implies a need for standard cloud data policies to allow users to migrate between different providers.

D. Related literature

Mobile access control policies have been discussed extensively in academic literature, although rarely in the context of web applications. In ubiquitous computing research, Kim et al. [22] have proposed an extended role-based access control model which can take into consideration changes in user context. Roles are useful abstractions in working environments, but are perhaps less appropriate for personal devices and ad-hoc collaborations. Corradi et al. [23] also introduce a context-based access control system and context model. Again, however, the focus is on pre-defined policies for certain contexts rather than user-defined policies for personal resources. Indeed, user-defined policies for ad-hoc interactions are arguably a more complicated problem; Mazurek et al. show that in home environments people need fine-grained access control and want to be able to express policies requiring others to ask before access to a resource is granted [24]. Baldauf et al. give a further survey of context-aware systems [25].

Several authors have suggested modifications to web browsers and their security model, most of which aim to mitigate attacks by malicious web pages. The Gazelle Web Browser [26] aims to protect web applications (defined by origin) from each other by separating them and their resources into different OS-enforced protection domains. This protection also applies to browser plugins. The Tahoma Web Browsing System [27] isolates websites by compartmentalising them into virtual machines. It makes web applications first-class objects, requiring them to be installed and approved before first use. Tahoma mediates all network interaction, and requires web applications to have a widget-like manifest which define any plugins that are required by the application. Finally, Singh et al. [28] introduce the notion of a *user principle* and suggest that all access to certain APIs and actions (geolocation, browsing history) should be considered *owned* by the user and access to them should therefore also be mediated by users.

In native mobile applications, the many improvements have been suggested to the Android permission system, as discussed in section II-B. In addition, Ai et al. [29] propose an alternative synchronisation system for storing and recovering user data via an online service. If applied to permissions, this might be a way to implement cross-device policies. However, their focus is primarily on preventing the loss of user data, rather than multi-device use cases.

E. Summary

Users are required to make access control decisions for both native and web applications. The general principles are the same in both cases – users must mediate access to potentially sensitive resources – although the way in which permission is granted differs. However, in both cases users end up making decisions out of context, either at install time or first use, and only for one device at a time. Existing systems which are context-aware tend to be aimed at users with one device as part of larger organisations, and systems such as Chrome allow synchronisation but not context adaptation. Yet context is even more important when policies are synchronised: one's security and privacy preferences when installing a mobile application at home may not be same as those associated with using it outdoors. Unfortunately, designing access control policies for different device and contexts is easier said than done, and likely to become more complex for web applications as they become more functionally capable and used on a wider range of devices. Furthermore, additional scenarios involving multiple devices being used together at the same time, and multiple users interacting, increase the need for a common way of expressing access control policies.

III. Cross-device authorisation

The state of the art described in the previous section mostly focused on one user with one mobile device. This does not reflect how people use technology today: they may rely on several devices in different contexts. This section describes the requirements for access control within a personal network of devices and explains how the *webinos* architecture begins to satisfy them.

A. Requirements for personal cross-device access control

Each user device can potentially run several web applications, each requiring access to local hardware capabilities as well as functionality from other devices. In combination with the current state of the art described in the previous section, we have elicited the following requirements.

1) Users and other stakeholders shall be able to control access by web applications to JavaScript APIs. These APIs may allow access to local and remote resources.
2) Users shall be able to create both device-specific and device-agnostic policies.

3) The platform shall provide synchronization for access control policies, so that policies can be described on one platform and enforced on all.
4) The platform shall allow *context-sensitive* access control decisions: e.g. these may change depending on the environment.
5) The platform shall protect user privacy: access requestors shall be able to qualify how they will use the data they are requesting, and users shall be able to express constraints about data disclosure.

Existing browsers and systems such as WAC provide some support for requirement 1, and the Chrome browser supports limited synchronisation to satisfy requirement 3. Context-aware middleware provides support for requirement 4. The XACML language may be well suited to expressing policies to satisfy requirement 2, and work by Ardagna et al. [20] can adapt this to satisfy requirement 5. However, the combination of these requirements and the cross-device nature of web browsing provide motivation for the development of the *webinos* platform.

B. The webinos platform

webinos is a cross-device application platform for mobile web applications and widgets. It provides applications with a set of APIs for accessing local resources, such as sensors and contacts, as well as APIs for communication with other devices and services. The platform aims to create a seamless multi-device user experience through data synchronisation and a consistent access control system. *webinos* is supported on four main device domains: PCs, smartphones, in-car systems and set-top boxes. It was primarily aimed at people who use and own many web-enabled devices, with an emphasis on personal and social computing. More detailed personas of anticipated users are available in project deliverables [30], as are detailed use cases and scenarios [31].

The platform was designed with the following high-level goals in mind:

- **Interoperability** of applications across the aforementioned four device domains. Each application can communicate with others on the same device, with another device belonging to the same user, or with an unknown device elsewhere.
- **Compatibility** – achieved through standard JavaScript APIs. This allows applications to run on multiple devices with minimal modification.
- **Security and privacy** for users and application developers.
- **Adaptability** – allowing applications and devices to take advantage of information about the current environment.
- **Usability** – through the creation of a *seamless experience* for users of applications across multiple devices.

C. webinos 'personal zones' of devices

The *webinos* platform aims to meet requirements 2-5 from Section III-A by introducing the concept of a *personal zone*, the set of all devices owned by an end user. It is an abstraction which provides a basis for managing devices, together with the services running on them. From the web application perspective, all devices in the zone support and expose a set of standard APIs for accessing services such as device features (cameras, geolocation), networking with other devices, and cloud services.

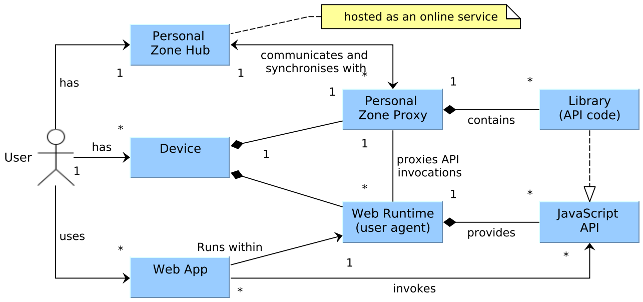

Fig. 1. Overview of the *webinos* personal zone model.

Personal zones contain three key components: the web runtime, proxy and hub. The *Web Runtime* is where web applications are executed. The runtime has been extended to provide a set of JavaScript APIs which allow applications to join a personal zone, communicate with other devices and access features. These API requests are forwarded to the personal zone proxy.

The *Personal Zone Proxy (PZP)* implements the APIs and provides the communication layer with other devices. It communicates with the hub through a mutually authenticated TLS session and communicates peer-to-peer with other proxies where internet connectivity is unavailable. Each proxy has its own policy enforcement and decision point.

The *Personal Zone Hub (PZH)* runs on a web server, which may be hosted by a third party or be part of a user's home networking equipment. It is the central point of access to the zone, so that each device can contact others and request access to resources and services. The hub is a repository for synchronized data and policies, and provides policy enforcement for incoming requests. The hub enrols each PZP into the zone by issuing it with a certificate.

The relationship between these components can be seen in Figure 1.

As an example, suppose that Alice is using her smartphone and wants to access a media file on her PC. She loads her media player web application, which requests access to the file via a JavaScript API provided by the *webinos* web runtime extension on her smartphone. The runtime extension passes the request to the PZP, which passes the request to the web-based PZH. The hub then forwards this to the PZP on Alice's PC, which contains the implementation of the File API. The file is then transferred through the hub back to the media player application on the smartphone.

IV. The *webinos* policy framework

A. Basic architecture

The *webinos* platform provides privacy protection and access control features to meet the privacy and security requirements of web applications and users. This means protecting against malware and other users who may attempt to access more data than they should or simply avoiding the unnecessary disclosure of personal data. To satisfy these requirements *webinos* relies on the strong identification of web applications, users and devices, combined with a least-privilege access control based on XACML.

However, while the latest version of XACML introduced a privacy profile specifying two *purpose* attributes [32], it is still not well suited for describing data handling policies. For this reason the XACML architecture has been adapted using PrimeLife extensions. [33], allowing *webinos* to make access control decisions based on both the request context and user preferences.

The object model depicted in Figure 2 shows the main components of the *webinos* policy framework. Both the PZP and PZH enforce policies using standard XACML components, supplemented by:

- The *Decision Wrapper* creates the initial policy enforcement query based on incoming requests.
- The *Access Manager* makes the final decision by combining XACML access control and DHDF data.
- The *Data Handling Decision Function* (DHDF) engine provides privacy and data handling functionalities.
- The *Request Context* manages all contextual information; it stores all the data and credentials released by a user in a given session.
- The *PDP Cache* (PDPC) stores PDP decisions that could be share among personal devices.

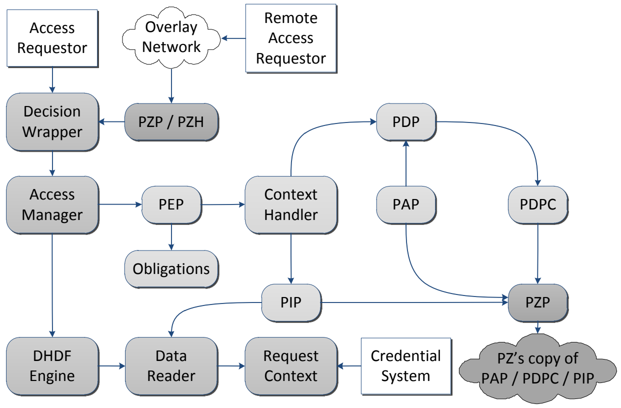

Fig. 2. Components of the *webinos* policy framework on a device. Requests come from either local or remote sources, and are then managed by the Decision Wrapper. Policies are held on a PZP and may depend on data fetched from the rest of the personal zone

B. Adapting the state of the art

Policies are expressed in a similar way to the format defined in BONDI [34]. However, BONDI policies do not support cross-device interaction. The *webinos* framework therefore modifies the *subject* of policies to refer to the device and user. Policies are also able to refer to a dynamically changing set of features, as new APIs may be added by new applications. *webinos* policies are usually composed in the following way, where device T is the *target device* and device R is the *requesting device*:

User U can access Feature F of Device T through application A on Device R

Figures 3 and 4 show an application requesting access to a remote device feature. Every subject (user, application and device) has an id used to determine if a specific policy should be taken in account when enforcing incoming requests. To perform this selection the Policy Manager tries to match the request's information against policy targets.

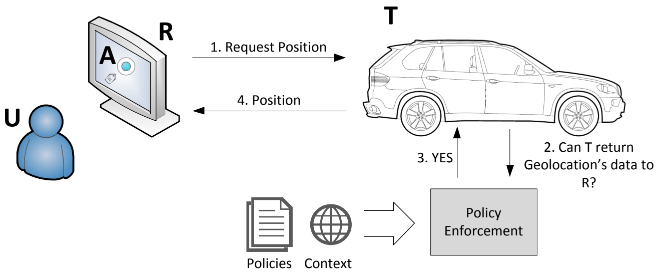

Fig. 3. Example of a request to access a remote device's features. Application A launched by user U in set-top box R requires access to geolocation data provided by car T.

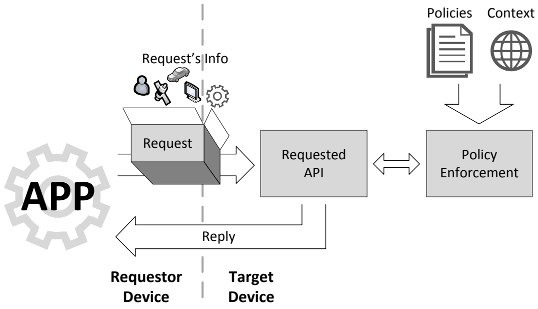

Fig. 4. Every application that requires to access a feature sends a request which is filled with information on the subjects and requested feature.

Special URIs are used as ids in policies to match subjects and services. Whenever an entity (user, device, application or service) is registered for the first time, a record with some basic identity information is stored. These records are used to convert a generic ID into a convenient URI.

There are four basic types of information in the policy's subject. The *user*, identified by the URI of their PZH. The *requesting device*, as identified by the TLS certificate issued to each device when it enrols in a personal zone; omitting this in a policy allows it to apply to applications running on any device. The *target device*, similarly identified; omitting this allows for policies which refer to an API on all devices, e.g. denying any access to location data. Finally, the *application* ID, author

```
<policy-set combine="first-matching-target">
  <policy combine="first-applicable">
    <target>
      <subject>
        <subject-match attr="user-id"
            match="U"/>
        <subject-match attr="id" match="A"/>
        <subject-match attr="requestor-id"
            match="R"/>
      </subject>
    </target>
    <rule effect="permit">
      <condition>
        <resource-match attr="api-feature"
            match="http://www.w3.org/ns/
            api-perms/geolocation"/>
      </condition>
    </rule>
    <rule effect="deny">
      <condition>
        <resource-match attr="api-feature"
            match="*"/>
      </condition>
    </rule>
  </policy>
  <policy combine="first-applicable">
    <rule effect="deny">
      <condition>
        <resource-match attr=
            "api-feature" match="*"/>
      </condition>
    </rule>
  </policy>
</policy-set>
```

Fig. 5. Example policy set, showing two policies on a device. The first policy allows user U (using application A on device R) to access geolocation data. The second denies access from every other combination of users, applications and devices.

and distributors. This is necessary in device domains (such in-car systems) where only applications developed by the device manufacturer may access certain resources.

Information about the application must be provided by the web runtime, which may be more vulnerable to attack. An advantage of moving policy enforcement into the PZP is that the impact of a malicious web runtime is limited: the proxy may be able to assess the state of the runtime, and policies applying to all applications will still be enforced.

Figure 5 gives an example of a policy which would be held on a local device.

C. *Inter-zone policy enforcement*

When web applications are running on different devices within a personal zone, access control is managed by the policies defined by the user and distributed on each device. When applications in *different* personal zones are requesting resources (e.g. Alice accesses Bob's location API), it depends on policies created by both users. In this scenario, Bob can create policies referring to Alice in the subject, and therefore control her access to each resource. It can be argued that other attributes in the subject may be omitted, as Bob has little assurance in Alice's software and may not trust it to correctly identify applications or devices. However, these fields are still present because Alice and Bob may be able to provide such assurances: e.g. they may be in the same physical location, and Bob may know exactly which device Alice is using and whether or not it is trustworthy. To save wasted bandwidth from denied access requests, Alice may chose to implement the more specific aspects of the policy herself. This also makes sense when considering trust: Bob trusts Alice to only let certain applications and devices use the resources he has made available to her. Alice can do this by creating a policy within her personal zone where the resource is an API on Bob's device, but the subjects are her applications and devices. Such a policy would need to be synchronised between all of Alice's devices.

V. Conceptual Challenges

A first proof-of-concept of the *webinos* system has been specified and implemented. This section will describe three conceptual challenges that have been experienced when designing and developing a cross-domain access control system.

A. *Integration with existing security frameworks*

Our experiences with *webinos* have added weight to the argument that existing browser access control is inadequate for increasingly sophisticated web applications. Browsers tend to either *always* prevent access to local resources, *prompt* the user for permission, or use implicit consent based on user actions. In a multi-device scenario, however, authorisation prompts cannot always be dictated by just the browser. Some applications require remote access to data and APIs – e.g. remote data wiping features for lost or stolen devices, and 'in case of emergency' applications [1]. These scenarios require more than just authorisation prompts and stored decisions.

Display capabilities on devices are also different: compare a TV to an in-car system, which may be displaying route information. Browser-style access notifications may be difficult to notice or disrupt the experience of the user. The same is true of how users input their access control decisions. For some devices, such as sensors or embedded systems, it may even make sense to delegate policy creation to a more capable device. The *webinos* security framework was conceived in order to be flexible enough for all of these settings. Adopting a common policy description language and enforcement framework would allow just interfaces to be adapted on individual devices, rather than the whole authorisation system.

The integration of *webinos* with a browser security model – a sandbox with time-of-use prompting – is in conflict with the *webinos* policy-driven approach. This means that existing browser APIs such as geolocation would need to be changed. We believe this is not necessarily a problem for compatibility, as *webinos* policies can be configured to behave like browsers. For example, access to the geolocation API in the browser

[1] http://ice-app.net/whatis.html

results in a user prompt and this decision can be saved per domain. In *webinos*, a policy can be configured to allow or deny access, or prompt the user based on application credentials, user and device context.

B. Subject and resource definitions

In a standard access control scenario, the resources to be protected are unequivocally defined, as is the access requestor. However, in a multi-device scenario different platforms may have different capabilities and implementations, and the importance of related assets to be protected will vary. For example, on a smart phone, access to telephony capabilities must be protected whereas on an in-car system the travelling data may be more important. It is therefore difficult to come up with a definition of high, medium and low risk permissions in the same way that the Chrome Web Store does.

We assume that each sensitive resource is only exposed by a certain API, reducing the problem to the definition of *webinos* APIs. These had to be specialized in order to cover all the features of each device domain and, conversely, abstract in order to describe common features. For example, many assets are common to all domains (e.g. File APIs), some are common to a subset of sub-domains (e.g. Telephony for in-car systems and smartphone) and some are exclusive to a domain (e.g. TV APIs). The APIs were designed to support domain-specific applications but also to facilitate portability of policies (e.g. a user would like to give an application access to a temporary directory for file system access whatever the target device is or give access to the Telephony API either from his smartphone or car).

As an added complication, the same API can be implemented with different technologies on each platform. The geolocation API can be provided by GPS, Cell-ID on smartphones, IP on laptop, or fixed ADSL line on a TV set. The implication in terms of privacy may vary as a result. For example, a user might grant access to the geolocation API on their PC, as this could only reveal their current home town through IP-based location services. If this permission also revealed their GPS-derived location while in their car or on their mobile device, this could reveal a history of their movements and interactions. The Telephony API is another example: if the end user is roaming on their mobile, then the potential impact of misuse (due to network costs) is greater compared to a home landline. Users might therefore wish to change the authorisation method from ‘always allow’ to ‘prompt first’. Policies should be capable of reflecting which underlying technology is used and the operating mode of the device. As a result, any default policies will be difficult to define and may quickly become complicated.

C. Policy management with varying ecosystems

Actors, value chains and liability frameworks are different in each device domain. This means that the policy management authority and associated hierarchy can vary from one domain to another as well as the system for provisioning such policies. XACML’s policy combining algorithms make it possible to write policies for all of these domains. The use of XACML makes web applications potentially more acceptable in environments where additional security policies may apply. For instance, when an institution’s employees may be the target of physical violence (e.g. animal testing laboratories), policies could be created limiting access to their location data from untrusted web applications.

The *webinos* design decision for the first proof-of-concept implementation was to separate policy enforcement from the policy storage and deployment system, leaving each domain to define means and rules for policy deployment and composition of policies from different sources. Again, this implies that each device will need to be shipped with different access control policies and defaults.

D. Shared devices

In this paper we have not discussed how to manage devices with multiple owners and users. There are two distinct scenarios. First, devices which have several owners but are only used by one person at a time. For example, a tablet or laptop. In this case multiple personal zone proxies can be installed on each device, running under different operating system user accounts. This modifies Figure 1 to add a ‘user account’ as a level of indirection between the user and their device.

The second case is more complicated: some devices are used by several people at the same time. For example, a car with several passengers, or a TV in a shared room. At present, *webinos* requires that these devices still belong to one user and be part of one personal zone. However, the user can limit this device so that it has fewer permissions to access the other devices in his or her personal zone. This prevents a guest from using it to access resources on the owner’s other devices. We are also investigating linking the policy system to authentication, such that a policy enforcement point on a shared device can demand additional authentication from the end user. This would be OS-specific, for example requiring a short PIN entry using a remote control, and would then make the device capable of accessing other resources and editing local policies. This approach is possible (without policy integration) as part of the defined *authentication API* [35]. However, further features are needed to adapt and control notifications by applications running on shared devices. For example, a shared device might be prevented from displaying notifications about private events, such as receiving a new email.

VI. Conclusion and future work

This paper has argued that existing browser-based authorisation methods for JavaScript APIs are insufficient for multi-device, context-aware scenarios. The policy architecture developed in *webinos* is proposed as an alternative: it makes a number of significant changes from the current state of the art, showing policies which are both abstract enough to be transferable to different devices, as well as specific to particular scenarios. Our experience in the first implementation of the platform has highlighted several challenges, such as the different implementation of common APIs and how to managed shared devices.

The main focus of future work will be to continue developing the *webinos* platform and to further evaluate its effectiveness through several concept applications. Other future work will aim to support the delegation of access control decisions, e.g. to a tech-savvy friend, a user support group, a forum on a social network, or a trusted service provider. In addition, we plan to implement tools to support developers in writing appropriate permissions and designing applications. In particular, allowing developers to trial their applications based on typical user policies (perhaps gained through user groups or anonymous feedback) would help identify where too few or too many permissions were requested, as well as where applications do not fail gracefully.

Acknowledgements

The research described in this paper was funded by the EU FP7 *webinos* project (FP7-ICT-2009-05 Objective 1.2).

References

[1] A. Taivalsaari and T. Mikkonen, "The web as an application platform: The saga continues," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, September 2011, pp. 170 –174.

[2] "Geolocation API Specification: W3C Editor's Draft," http://dev.w3.org/geo/api/spec-source.html, February 2010.

[3] "Device APIs Working Group Website," http://www.w3.org/2009/dap/, December 2011.

[4] "Bmw connecteddrive," April 2012. [Online]. Available: http://www.bmw.com/com/en/insights/technology/connecteddrive/2010/info%tainment/information/internet_information.html

[5] L. Jedrzejczyk, B. A. Price, A. K. Bandara, and B. Nuseibeh, "On the impact of real-time feedback on users' behaviour in mobile location-sharing applications," in *Proceedings of SOUPS '10*. ACM, 2010, pp. 14:1–14:12. [Online]. Available: http://doi.acm.org/10.1145/1837110.1837129

[6] The W3C, "Mobile Web Application Best Practices," December 2010. [Online]. Available: http://www.w3.org/TR/2010/REC-mwabp-20101214/

[7] The Wholesale Application Community, "Glossary of terms," August 2011. [Online]. Available: http://www.wacapps.net/glossary

[8] M. Zalewski, *The Tangled Web: A Guide to Securing Model Web Applications*. No Starch Press, 2011.

[9] "Widget Access Request Policy: W3C Recommendation," February 2012. [Online]. Available: http://www.w3.org/TR/widgets-access/

[10] W. Enck, M. Ongtang, P. McDaniel, W. Enck, M. Ongtang, and P. McDaniel, "Understanding Android Security," *Security Privacy, IEEE*, vol. 7, no. 1, pp. 50 –57, Jan - Feb 2009.

[11] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *Proceedings of the 2nd USENIX conference on Web application development*, ser. WebApps'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 7–7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002168.2002175

[12] A. Beresford, A. Rice, N. Skehin, and R. Sohan, "Mock-droid: trading privacy for application functionality on smartphones," in *Proceedings of HotMobile 2011*, 2011. [Online]. Available: http://www.cl.cam.ac.uk/research/dtg/android/mock/

[13] M. Nauman, S. Khan, and X. Zhang, "Apex: extending android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, 2010, pp. 328–332. [Online]. Available: http://doi.acm.org/10.1145/1755688.1755732

[14] M. Conti, V. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," in *Information Security*, ser. Lecture Notes in Computer Science, M. Burmester, G. Tsudik, S. Magliveras, and I. Ilic, Eds. Springer Berlin / Heidelberg, 2011, vol. 6531, pp. 331–345. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-18178-8_29

[15] N. Reddy, J. Jeon, J. A. Vaughan, T. Millstein, and J. S. Foster, "Application-centric security policies on unmodified android," UCLA Computer Science Department, Tech. Rep. 110017, 2011. [Online]. Available: http://fmdb.cs.ucla.edu/Treports/acp-tr.pdf

[16] "SEAndroid Website," http://selinuxproject.org/page/SEAndroid, March 2012.

[17] "WAC Specifications 2.1," http://specs.wacapps.net/, January 2012.

[18] S. Godik and T. Moses, "EXtensible Access Control Markup Language (XACML) version 1.1," http://www.oasis-open.org, May 2005.

[19] "Primelife policy language," http://www.primelife.eu/results/documents/153-534d, August 2010.

[20] Claudio A. Ardagna et al., "Advances in access control policies," in *Privacy and identity management for life*. Springer, 2011, pp. 327–341.

[21] D. Lin and A. Squicciarini, "Data protection models for service provisioning in the cloud," in *Proceedings of SACMAT*. ACM, 2010, pp. 183–192. [Online]. Available: http://doi.acm.org/10.1145/1809842.1809872

[22] Y.-G. Kim, C.-J. Mon, D. Jeong, J.-O. Lee, C.-Y. Song, and D.-K. Baik, "Context-aware access control mechanism for ubiquitous applications," in *Advances in Web Intelligence*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, vol. 3528, pp. 932–935. [Online]. Available: http://dx.doi.org/10.1007/11495772_37

[23] A. Corradi, R. Montanari, and D. Tibaldi, "Context-based access control management in ubiquitous environments," in *Network Computing and Applications, 2004. (NCA 2004). Proceedings. Third IEEE International Symposium on*, aug.-1 sept. 2004, pp. 253 – 260.

[24] Mazurek et al., "Access control for home data sharing: Attitudes, needs and practices," in *Proceedings of the 28th international conference on Human factors in computing systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 645–654. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753421

[25] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, June 2007.

[26] H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter, "The multi-principal os construction of the gazelle web browser," in *Proceedings of the 18th conference on USENIX security symposium*, ser. SSYM'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 417–432. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855768.1855794

[27] R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen, "A safety-oriented platform for web applications," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, ser. SP '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 350–364. [Online]. Available: http://dx.doi.org/10.1109/SP.2006.4

[28] K. Singh, A. Moshchuk, H. J. Wang, and W. Lee, "On the incoherencies in web browser access control policies," in *Security and Privacy (SP), 2010 IEEE Symposium on*, may 2010, pp. 463 –478.

[29] C. Ai, J. Liu, C. Fan, X. Zhang, and J. Zou, "Enhancing personal information security on android with a new synchronization scheme," in *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*, sept. 2011, pp. 1 –4.

[30] The webinos consortium, "User expectations of privacy and security," March 2011. [Online]. Available: http://webinos.org/blog/2011/03/16/d2-3-industry-landscape-ipr-licensing-governance/

[31] ——, "Use cases and scenarios," March 2011. [Online]. Available: http://webinos.org/blog/2011/03/22/webinos-report-use-cases-and-scenarios/

[32] E. R. B. Parducci, H. Lockhart, "XACML v3.0 Privacy Policy Profile Version 1.0," http://docs.oasis-open.org/xacml/3.0/xacml-3.0-privacy-v1-spec-cd-03-en.pdf, March 2010.

[33] C. A. Ardagna, S. De Capitani di Vimercati, S. Paraboschi, E. Pedrini, and P. Samarati, "An xacml-based privacy-centered access control system," in *Proceedings of the first ACM workshop on Information security governance*, ser. WISG '09. ACM, 2009, pp. 49–58.

[34] "Bondi architecture and security requirements appendices," http://bondi.omtp.org/1.01/security/BONDI_Architecture_and_Security_Appendices_v1_01.pdf, July 2009.

[35] The webinos consortium, "Phase 1 device, network and server-side api specifications," November 2011. [Online]. Available: http://webinos.org/blog/2011/11/01/webinos-report-phase-i-device-network-and-server-side-api-specifications/