Developing secure, unified, multi-device, and multi-domain platforms: a case study from the webinos project.

ATZENI, A., LYLE, J. and FAILY, S.

2014

Architectures and Protocols for Secure Information Technology Infrastructures

Antonio Ruiz-Martínez
University of Murcia, Spain

Rafael Marín-López
University of Murcia, Spain

Fernando Pereñíguez-García
University of Murcia, Spain

A volume in the Advances in Information Security, Privacy, and Ethics (AISPE) Book Series

Information Science REFERENCE
An Imprint of IGI Global

Managing Director: Lindsay Johnston
Editorial Director: Joel Gamon
Production Manager: Jennifer Yoder
Publishing Systems Analyst: Adrienne Freeland
Development Editor: Monica Speca
Assistant Acquisitions Editor: Kayla Wolfe
Typesetter: Travis Gundrum
Cover Design: Jason Mull

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Architectures and protocols for secure information technology infrastructures / Antonio Ruiz Martinez, Rafael Marin-Lopez and Fernando Pereniguez Garcia, editors.
pages cm
Includes bibliographical references and index.
ISBN 978-1-4666-4514-1 (hardcover) -- ISBN 978-1-4666-4515-8 (ebook) -- ISBN 978-1-4666-4516-5 (print & perpetual access) 1. Information technology--Security measures. 2. Computer networks--Security measures. I. Martinez, Antonio Ruiz, 1976-, editor of compilation. II. Marin-Lopez, Rafael, 1977-, editor of compilation. III. Garcia, Fernando Pereniguez, 1984-, editor of compilation.
QA76.9.A25A73 2014
005.8--dc23
2013020692

This book is published in the IGI Global book series Advances in Information Security, Privacy, and Ethics (AISPE) (ISSN: 1948-9730; eISSN: 1948-9749)

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 12
Developing Secure, Unified, Multi-Device, and Multi-Domain Platforms:
A Case Study from the Webinos Project

Andrea Atzeni
Politecnico di Torino, Italy

John Lyle
University of Oxford, UK

Shamal Faily
University of Oxford, UK

ABSTRACT

The need for integrated cross-platform systems is growing. Such systems can enrich the user experience, but also lead to greater security and privacy concerns than the sum of their existing components. To provide practical insights and suggest viable solutions for the development, implementation, and deployment of complex cross-domain systems, in this chapter, the authors analyse and critically discuss the security-relevant decisions made developing the Webinos security framework. Webinos is an EU-funded FP7 project, which aims to become a universal Web application platform for enabling development and usage of cross domain applications. Presently, Webinos runs on a number of different devices (e.g. mobile, tables, PC, in-car systems, etc.) and different Operating Systems (e.g. various Linux distributions, different Windows and MacOSx versions, Android 4.x, iOS). Thus, Webinos is a representative example of cross-platform framework, and even if yet at beta level, is presently one of the most mature, as a prototype has been publicly available since February 2012. Distilling the lessons learned in the development of the Webinos public specification and prototype, the authors describe how potential threats and risks are identified and mitigated, and how techniques from user-centred design are used to inform the usability of security decisions made while developing the alpha and beta versions of the platform.

DOI: 10.4018/978-1-4666-4514-1.ch012

INTRODUCTION

People use multiple devices with different form factors every day. These devices provide access to similar services but in different ways - native apps, Websites, mobile-specific Websites, etc. As such, these devices are interacting with each other more often, either to synchronize data or to provide cross-device user experiences, e.g., using a smart phone as a remote control for a smart TV, or having a companion application to a live TV programme. These new activities, scenarios and cross-domain user experiences require greater communication and increase the potential for misuse.

For example, *Gloria likes to personalize her online experience by setting application preferences, but also for privacy reasons she retains separate online identities. Gloria may be used to adopting a mobile device for one identity and a laptop for another, each of which covers two separate contexts. With smart systems and identity providers both available, Gloria's device may switch from one identity to another, but Gloria may be unaware of this switch if she set up her device to move between services without any intervention. In fact, she may not be aware which identity is exposed unless her activities are such that she would be conscious of an identity switch*.

Every different device may make a different trade-off considering authentication, authorization, and usability. For example, some devices may only infrequently ask the user to authenticate in order to minimize the use of a small keyboard or screen. However, when devices are used together, their different settings may conflict and either harm the user experience or reduce the system's security.

Security control can introduce usability problems (Schneier, 2009) as configuring and then using complex security features, like access control systems, can be difficult, time-consuming and fundamentally at odds with the primary goals of the end user. As each new platform may have a different system and interface for doing this, the access control problems in cross-device systems are magnified.

How security problems can be addressed in such a complex scenario without losing focus on the usability of the system is the topic of this chapter. The chapter describes a case study in multi and cross-device access control based on the *Webinos* project. The *Webinos* project has designed and implemented a cross-platform application environment which allows developers to create applications which can communicate seamlessly between each platform. This includes the development of a personal device network (Niemegeers and Heemstra de Groot, 2002) which attempts to solve many of the related problems. User-centred design techniques are one of the most important points in our approach. Users are personified as specific entities (like Gloria), with skills, attitudes and motivations, to avoid talking about generic users" who might become contradictory when based on solely on the imagination of developers.

This chapter is structured as follows: section 2 introduces the *Webinos* project and gives a high-level technical overview, as well as listing desired goals, implementation details and the most important concepts related to the *Webinos* architecture. Section 2 also introduces related work on similar architectures to inform our security framework. Section 3 explains in detail how we approached the key usability problems. Section 4 states the main threats we identified in the cross-domain *Webinos* platform. Section 5 introduces the security of the *Webinos* execution environment. Section 6 describes the different types of authentication mechanisms introduced in *Webinos*. Section 7 highlights how to secure a communication session to avoid confidentiality and integrity losses. Section 8 addresses the core of the access control system: the policy framework. Section 9 approaches another task performed as part of *Webinos*: the security analysis of the APIs introduced in *Webinos* so far. Section 10 briefly describes the need for a secure storage to keep confidential information in our cross-device system. Section 11 finally discuss our findings in a multi-platform system and draws conclusions.

BACKGROUND

Webinos

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 address books, as well as APIs for communication with other devices and services. The platform aims to create a seamless multi-device user experience through data synchronization and a consistent access control system. *Webinos* is supported on four main device domains: PCs, smartphones, in-car systems and set-top boxes. The *Webinos* project (The *Webinos* Consortium, 2012c) consists of a consortium of over twenty partners, including mobile network operators, device manufacturers, industry research institutions, universities and software companies.

Webinos is suitable for augmenting common scenarios like the following: *Helen and her family see an advert on television for a skiing holiday and decide to book it using their TV. Automatically, their calendars are updated, the car navigation system adds the destination, and a post is added to Helen's social network. On the long car journey, Helen plays a game with her children using their in-car entertainment system and her smartphone. A few minutes later, Helen's parents call, and she invites them to take over playing the game with the children remotely, giving her a much-needed break (*a set of scenarios and use cases are available in the Webinos deliverables [The Webinos consortium, 2012d]).

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

- Interoperability of applications across the 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.

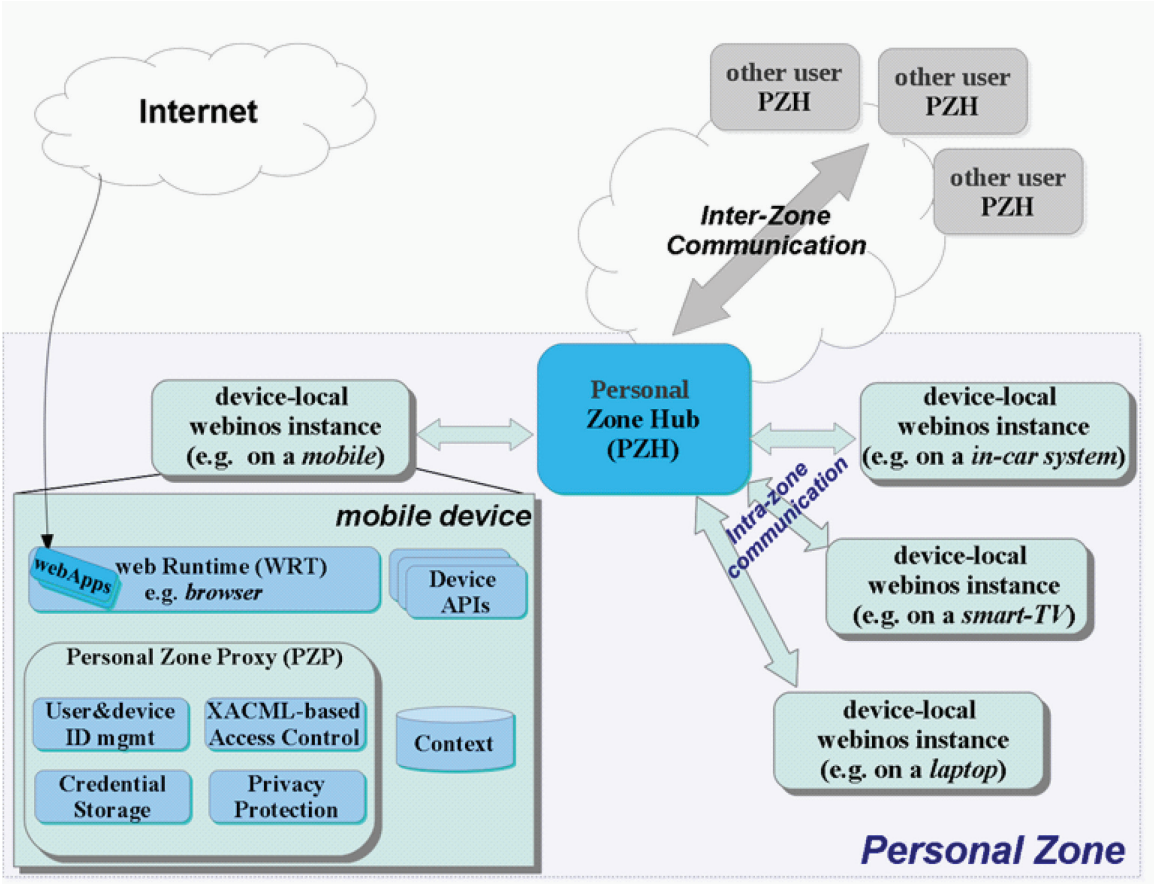
The *Webinos* runtime has been officially implemented so far for three target platforms: Android for smartphones and tablets, Windows for PC, and Linux variants for in-car systems and set-top boxes (other unofficial versions exist, e.g., for MacOSX and iOS).

Webinos is based around the concept of *personal zones*, as shown in Figure 1, and consists broadly of three components, as listed in Table 1.

A user's personal zone is the set of all their devices. Each personal zone has a *personal zone hub* (PZH), which coordinates communication, synchronizes data and provides access to devices from the Internet. All other devices have a *Web Runtime* (WRT) (much like a browser) which displays Web applications and process widgets. The Web runtime has been extended with a *Webinos* plug-in to connect it to a local *Personal Zone Proxy* (PZP), which implements APIs, provides local access control and communicates with the personal zone hub.

The hub is responsible for discovering Internet services and appropriately routing requests from and to each device in the zone. The hub must be constantly online and addressable, so that any device on the Internet can potentially communicate with devices within the zone. This allows for remote data sharing and resource usage both within and between personal zones. The proxy will cache all routing information, so that when a hub is not accessible (e.g. when a device loses network

Figure 1. Overview of Webinos



access) the proxy can perform many of the same tasks and support peer-to-peer communication. The *Webinos* architecture is therefore federated: each user has their own personal zone hub and each device has its own proxy. The hub is a key component in the architecture because it provides a central location for storing and synchronizing data, but also because it has many useful security features. The hub can act as a trusted party. In one example application it is used to host personal data rather than trusting it to the application provider.

The hub can be installed on any device, but we have implemented it to be either a cloud-based virtual platform or installed on a home router. The personal zone hub is an essential component within *Webinos* and potentially useful for any Web application middleware.

Table 1. Personal zone components

Component	Key Features and Capabilities
Personal Zone Hub (PZH)	Constantly available and addressable, routes messages, acts as a certificate authority. The hub provides a Web-based user interface to control the personal zone and audit activity.
Personal Zone Proxy (PZP)	Implements most of the Webinos JavaScript APIs, and provides policy-driven access control. As the name suggests, it proxies requests between the PZH and the Web runtime
Web Runtime (WRT)	The user interface to Web applications and widgets.

Component Technologies

The Webinos project makes use of several existing technologies and frameworks.

For initial user registration and subsequent management of the personal zone through a Web interface on the personal zone hub, Webinos uses *OpenID*. OpenID is a decentralised identity and authentication system for Web-based systems. It allows users to register with an *identity provider (IdP)* and then re-use this provider when authenticating to other Websites. For example, when registering with an online Web forum, the end user can specify their identity provider and then will be redirected to this provider to authenticate. The identity provider will send the forum an *identity assertion* proving that the user has ownership of the claimed identity. This allows users to have fewer identities and authentication credentials, and means that they do not have to give passwords to potentially untrustworthy third party services.

The Webinos implementation uses NodeJS, an open source JavaScript runtime for distributed network applications (Joyent, Inc., 2012). NodeJS had to be ported to Android, but was then available on all target platforms. The rest of the platform was written in device-agnostic JavaScript as well as some native C++.

RELATED WORK

The emerging field of cross-platform security is only partially covered by well-established security research on home and personal area networks, which focus on logically or physically co-located collection of devices, like in a home wireless network. Security is only partially covered since, for example, a home network tends to use only one medium, such as WiFi, and therefore cannot encompass mobile devices and connection via mobile device networks.

Since the *Webinos* project is an attempt to make the creation of secure cross-device platform, it addresses the security and privacy issues of the new scenarios' increased connectivity, applying some concepts and adapting and evolving previous security solutions when needed.

In particular,

- Kinkelin et al. (2011) adopted a method to create device-user link and trust relationship among different (home) networks. We extend the solution to the mobile world and add certificates exchange.
- UPnP Device Protection Service (UPnP Forum, 2011) is interesting for combination of certificates and device pairings, which is also our approach, but we adopted a simpler schema of level of privileges and hierarchies.
- SHAMAN (Mitchell and Schaffelhofer, 2004; SHAMAN Project, 2002) investigated the personal CA concept that we borrowed, but we make different assumptions about the place of the personal CA, which is cloud based and in principle always available.
- UIA (Ford et al., 2006) proposed a similar but more general solution. We argue that, in practice, a *Webinos*-like structure would be the most common way in which this solution would be realised, considering the necessity of dealing with Network Address Translation and mobile networks. Differently from UIA, we introduce a further step of usability leveraging on existing user identities when available (e.g. in social networks).

Where previous proposal failed to convincingly cover a particular usage scenario, we tried to fill the gap. The main gaps we identified are mostly related with user acceptability of the security mechanisms.

- Web PKI are not suitable for users of Web servers and their devices, as the scarce use

of client certificates on the public Web demonstrates. This means that two users of the Web cannot identify each other with secure and robust mechanisms.
- A home network tends to use only one medium, such as WiFi, thus Home PKI cannot interoperate with mobile networks, and therefore cannot encompass mobile devices which often connect from remote locations via mobile data
- User PKIs have usability problem, and are not integrated with social networks

Our focus on user experience in Webinos is motivated by some milestone articles on usable security: Whitten and Tygar (1999) suggests that security needs a usability standard that is different from those applied to 'general consumer software', while Sasse et al. (2001) advocates properly applying standard usability design techniques for addressing security problems, interest in HCI-security is also witnessed by specific sessions at major conferences (e.g. Faily et al., 2013).

The decision to use domain certificates to bootstrap a PKI is based on what we believe users will accept. By incorporating the complex PKI mechanisms into the underlying Webinos middleware, specifically in the PZH (in principle always available in the "cloud"), we have made the PKI metaphor usable for end users who might otherwise be unwilling to invest in PKI management.

Mobile Application Projects

Android is an open source platform derived from Linux 2.6, shaped for mobile devices. Android security (And, 2012) is based on two different mechanisms: *sandboxing* and access control based on read-write-execute permission tuple.

Each Android application is hosted in a Dalvik VM, which is an optimized interpreter for resource (power, memory) scarce devices. Each application runs sandboxed (isolated) from each other in its own instance of the Dalvik virtual machine. The kernel is responsible for sandboxing management. Each instance of the Dalvik virtual machine represents a Linux kernel process. Applications must declare needed permissions for capabilities not provided by the sandbox, so the system prompts the user for consent (at install time). Permission may be enforced at the time of a call into the system, starting an activity (i.e. an application component), sending and receiving broadcasts, accessing and operating on a content provider, and binding to or starting a service.

The second security mechanism is essentially the same of Linux OS. Files and data held by an application are isolated from other applications enforced by the Android Linux kernel and traditional Unix file permissions. To access data from another application, it must first be exposed via a content provider accessed by the message bus.

To ensure application integrity and authenticity, applications must be signed with a certificate whose private key is held by their developer. The certificate identifies the author of the application and does not need to be signed by a certificate authority.

iOS previously known as iPhone OS (iOS, 2010), is a Unix-like operating system developed by Apple for its smartphones and tablets. In iOS, every application is sandboxed during installation. The application, its preferences, and its data are restricted to a unique location in the file system and no application can access another application's preferences or data. In addition, an application running in iOS can see only its own keychain items. The keychain is used to store passwords, keys, certificates, and other secrets.

Its implementation, therefore, requires both cryptographic functions to encrypt and decrypt secrets, and data storage functions to store the secrets and related data in files. To achieve these aims, Keychain Services calls the Common Crypto dynamic library. Digital signatures are required on all applications for iOS. In addition, Apple adds its own signature before distributing an iOS application. Apple does not sign applications

that have not been signed by the developer, and applications not signed by Apple simply will not run (Mac, 2010a, 2010b).

Webinos has incorporated several useful ideas from mobile operating systems' security:

- *Code signing*, to prevent installation/instantiation of non-trusted applications (i.e. not authenticated and/or not modified by non-authorized parties and/or provided by untrusted parties).
- *Sandboxing*, to prevent unwanted influences of one application to another one and or to the runtime.
- *A security policy framework*, that is as simple as possible to avoid usability problems and lead to misconfiguration, but expressive enough to allow detailed access control to any key features and functions.

BONDI (bon, 2009) is a composite specification allowing Web applications (widget and Web pages) to interoperate over BONDI defined execution environments. The security framework introduced in BONDI allows different forms of security policy to be expressed based on widget resource signatures (compliant with W3C Widgets 1.0 digital signature specification [W3C Widgets, 2011]). Signatures associated to each widget are also used to assure provenance and integrity. It allows blacklisting and/or whitelisting of widgets, authors, and Websites. The model identifies identity types, resources, attributes and conditions that can be expressed in an XML-based interchange format.

While *Webinos* took inspiration from this work, he management of a security policies can be a source of usability problems, particularly given BONDI's focus on mobile devices rather than device owners. In the following section we will describe how we improved access control system to cope with these multiple domains, and studies were carried out to improve usability.

Usability

In addition to enabling the convergence of different device platforms, we also designed *Webinos* to meet the expectations of a broad user base. This meant that not only would *Webinos* need to be secure in light of a broad range of risks, these risks would need to be addressed without compromising the user experience of *Webinos*-enabled applications. The consequences of failing to do this are well reported in the HCI Security literature. For example, Whitten & Tyger's seminal work on the usability of PGP (Whitten and Tygar, 1999) illustrates how, despite developing an aesthetically pleasing graphical user interface, users were unable to correctly configure and use PGP to encrypt email; this was because the mental models used by the developers of PGP were at odds with those associated with its end users. Surprisingly, insights into how to incorporate human factors into the design of secure systems have been limited, despite the growing interest in HCI-Security at both Information Security and HCI conferences. This is slowly beginning to change, as evidenced by dedicated sessions at major conferences (e.g. Faily et al., 2013); however, the state-of-the-art for designing usable security remains the application of classic user-centred design techniques to voice the security expectations of a system's stakeholders.

To account for these user expectations, we created and extensively used behavioural specification of archetypical users called *personas*. Personas (Cooper, 1999) are artefacts designed to deal with programmer biases arising from the word *user*. These biases can lead to programmers bending and stretching assumptions about users to meet their own expectation. To address these

biases, designers explicitly develop for specific user profiles; these represent the target segment of the system or product being designed. This approach brings two benefits. First, designers only have to focus on those requirements necessary to keep the target persona happy. Second, the idiosyncratic detail associated with personas makes them communicative to a variety of stakeholders. Since their initial proposal over a decade ago, Personas have become a mainstay in User-Centred Design, with articles, book-chapters, and even a book (Pruitt and Adlin, 2006) devoted to the subject of developing and applying them to support usability design. Personas have also been found to be useful as a tool for eliciting requirements for secure systems (Faily and Fléchais, 2010).

When designing *Webinos*, personas were used to surface assumptions that different project team members held about prospective users.

Despite their popularity, the process for developing personas is often methodologically weak, with little concrete guidance available about how to begin personas development effort, and how to structure their analysis. To address these weaknesses, we devised methodologically grounded process to develop them. This led to several "end-user" and "developer" personas (The *Webinos* Consortium, 2011).

Personas were also used to inform threat modelling by the creation of *attacker personas*. The adversarial element is an intrinsic part of the design of secure systems, but usually assumptions about attackers and threat is often limited or stereotypical. One component of a threat is a threat agent, the person or organisation who is motivated to fulfil the threat by attacking the system. We used attacker personas to model these agents, using an approach for developing them which is both grounded and validated by structured data about attackers (Atzeni et al., 2011).

These personas were created in the same way as other *Webinos* personas, but their characteristics were based on data sources about known attackers. The attacker personas were chosen to be representative of OWASP (OWASP Foundation, 2011) human threat agents. To mitigate the risk of developing irrational attacker models, we chose not to model rare but possibly very dangerous attackers, such as government or organised-crime sponsored professional hackers; accurate information about such attackers is not generally available.

The grounding of attacker personas is based on three important characteristics: they are representative of known attacker classes; they are representative of criminals convicted for common online crimes; and they are situated within the context of *Webinos* by design and workshop discussions. As a result, supplemental threat modelling artefacts appeared more realistic, because they were grounded in what a concrete attacker can and is willing to do.

THREAT MODEL

As detailed in section 2, the *Webinos* platform can be split into two key components:

- An application runtime environment (the WRT, essentially an environment providing Web browser's functionalities) for executing applications securely and providing APIs for accessing local resources.
- An overlay network connecting devices belonging to different people and on different networks to support multi-device use cases.

As such, threats tend to exist either at the application execution or network layer. In addition, we must consider the impact of physical threats such as device loss, theft or interference during maintenance. We also consider threats to data storage.

Threats were identified by approaching the problem through a structured risk-analysis approach, which also addressed human factors. We developed a model of *Webinos* based on the IRIS (Integrated Requirement and Information Security) meta-model (Faily and Fléchais, 2010), fed

by the *Webinos* user and attacker personas. The attacker personas were grounded in data sources accredited by the security community, such as the Common Attack Pattern Enumeration (CAPEC) (The MITRE Corporation, 2012) and The Open Web Application Security Project (OWASP) 'Top Ten Project' (OWASP foundation, 2011). In addition, as part of the development of *Webinos*, we identified misuse and misusability cases (Faily and Fléchais, 2011) and threats early on in the design phase and applied security *pre-mortems* (Faily et al., 2012) to elicit sources of threats.

Application Environment Threats

Threats to the application environment are broadly the same as threats to any Web browser, but with the added impact of new device APIs and services being misused. We identified *Webinos*-specific threats and attacks including:

1. **Unauthorised use of APIs and remote resources through content injection (XSS/CSRF):** A vulnerable application could be trusted by the end user but load malicious JavaScript from a third party. This JavaScript could take advantage of the application's privileged status and misuse the APIs that it has access to. For example, misusing the messaging API to send unauthorised text messages to premium numbers.
2. **Vulnerability exploitation in the underlying device platform through Webinos APIs:** If a *Webinos* API to access a local resource, such as a sensor, was implemented in native code and contain a buffer overrun or similar attack, a Web application could exploit this to gain access to the system. This would allow the machine to be added to a botnet, or for user data to be stolen.
3. **Eavesdropping on communication between applications and Webinos:** Applications served over HTTP are vulnerable to requests and responses (which may contain valuable data or credentials) being intercepted and modified.
4. Application Denial of Service by competing application developers. If Web applications are competing for users, then one might attempt to exploit a vulnerability to render the other unusable and drive people to alternatives. For example, a content injection attack could deface the Web application or crash on start-up.
5. Applications capturing hidden analytics about end users. In particular, *Webinos* allows for recording of user context evolution, and provides an API (the context API [The *Webinos* Consortium, 2012b]) to allow application to access these data. This might be misused to track the user's activities and behaviour in unwanted and privacy-invasive ways.
6. **Device availability loss through battery exhaustion:** Malicious or poorly developed Web applications might run resource-intensive code and exhaust the battery of a mobile device, rendering it temporarily unusable.
7. **Cross-site scripting (XSS):** If an insecure application loads JavaScript injected by a malicious third party, it could result in loss of data or cause the Web application to misbehave.
8. **Theft of identity credentials:** If an attacker gains access to the user's OpenID credentials used to log into the PZH, this could result in loss of confidentiality and integrity of stored data, loss of access to administration console, impersonation, loss of other credentials.
9. **Man in the Browser' attacks:** Malicious plugins might be installed which are able to steal Web application data.
10. **Evasion of access control policies through use of non-Webinos APIs:** If the underlying device platform offers alternative ways of accessing device resources, a *Webinos*

application might use them to circumvent the access control system.

11. **Spoofing of PZH administration page to steal user credentials:** The user might be tricked into entering credentials into an unauthorised page through spoofing.
12. **Use of accidentally-enabled test code and experimental APIs in Webinos deployments:** If the *Webinos* platform was developed with test code that remained enabled after deployment, attackers could misuse this capability to bypass policy controls or exploit the platform.
13. **User linkability through fingerprinting browser APIs:** The addition of new browser APIs would make it easier for advertisers to track the same user between sites; they would have a similar set of APIs available.
14. **Two-factor authentication defeat through misuse of Webinos messaging APIs:** The *Webinos* messaging APIs might allow a Web application to view SMS messages used as a second factor of authentication.
15. **Misconfigured access controls exploitation:** if users set overly permissive access control policies, applications may be able to gain unexpected access to resources.
16. **Identification of weak policies through context framework:** Since the context framework log also policy usage, poorly restricted access to context data can allow policy related information leakage.
17. **Insecure storage of Webinos data:** Offline Web application data might become available to local malware, or to a thief who has gained physical access to a device.
18. **Failure to check permissions on access requests:** A weakness in the *Webinos* implementation could be exploited to gain unauthorised access to APIs.

While several more threats and attacks still remain, this list does provide useful coverage of threats to the *Webinos* application environment and test cases for the specification and requirements.

Network Threats

Threats to a *Webinos* personal zone may impact the security of every device within it. The following threats and attacks have been identified and must be mitigated in the architecture.

1. **Insecure key storage and use:** The *Webinos* platform uses PKI to identify devices. If a device key was stolen or copied by an attacker, they would be able to join another device to the personal zone and either impersonate the user or gain access to other services.
2. **Unauthorised joining of a personal zone:** A user's PZH might allow an unauthorised user to add a new device to the personal zone. This attack might be hard to detect and would allow a range of misuses of personal devices.
3. **Impersonating a friend when requesting access the user's personal zone:** If Mallory is able to impersonate Bob, he could do this to create a connection to Alice's personal zone and access her resources. This may be possible if Alice is not sure of how Bob can be identified, or if Bob's user credentials are easy to guess or steal.
4. **Unauthorised enrolment of a user device to a malicious personal zone hub:** Before a personal zone proxy is configured to point to the user's personal zone hub, an attacker might force it to join another, malicious hub without the user realising. This would then give the attacker access to Alice's device and allow the attacker to impersonate Alice.
5. **Unauthorised transfer of data from secure to insecure devices:** If one *Webinos* device is well-secured and difficult to access, an attacker might use a less secure device to access it via *Webinos*.

Physical and Environmental Threats

1. **Misuse of physical access to access data stored on remote devices:** Similarly to the previous attack, this involves a malicious engineer or technician misusing their access to a device during maintenance (such as a car during its annual service) to access data stored on the rest of the personal zone.
2. **Exploiting NFC capabilities to impersonate users via a relay attack:** Alice's NFC reader might be made available over *Webinos* and then relay attacks could allow Eve to impersonate her NFC device for mobile payment or identity theft.
3. **Exploiting Bluetooth capabilities:** Since Bluetooth is developed for easy connectivity, often it is used in "mode 1" (no encryption and authentication), and thus bluetooth connections allow for impersonation, eavesdropping and connection hijacking.

Data at Rest

The following components in *Webinos* will be storing data.

1. Applications may store data locally on each device, as well as using data (such as media files) exposed by each device. To support this, an application specific, isolated storage area is made available to each *Webinos* application. In addition, *Webinos* can also expose arbitrary data storage. Access to arbitrary data storage will be mediated by policies and require a different permission. Isolated storage from one application is never exposed to another.
2. Devices. Each device with a PZP will store some or more of the following:
    a. Application data
    b. Data in policies, certificates, and preferences. This may include the names of applications the user has installed, the devices they use and their friends' identities. It is therefore considered private.
    c. Browser histories and system logs
    d. Context data (a temporary log file), if enabled.
    e. Downloaded widget data containing potentially valuable intellectual property
3. Cloud-based components (PZH, online services) may store:
    a. Context data
    b. Data in policies, certificates, and preferences. This may include the names of applications the user has installed, the devices they use and their friends' identities. It is therefore considered private.
    c. Application data (outside of *Webinos* control)

We therefore identify the following threats in Table 2.

In the sections that follow, we will describe how these threats have been addressed.

Trusted Execution Environment

Threats like unauthorized use of APIs and resources, application denial of service, cross site scripting, man in the browser demand a trusted environment of execution to be mitigated.

The security of the *Webinos* execution environment is specified to achieve security properties, starting from some assumptions. The *Webinos* core components, i.e. the PZH, the PZP and the *Webinos* runtime system are assumed as trusted, so, threats involving the corruption of code while on the device or modification of the runtime itself are not considered.

However, these components are considered at different level of security. While the PZH is the core of the security framework, it is usually placed in particularly secure place (e.g. in the cloud

Table 2. Reference threats for data at rest

Threat	Description	Attacker	Motivation
Native malware	Malware is installed on *Webinos*-enabled device and is used to access and transmit application data to a third party. This may be targeted to particular apps or users	Irwin	Corporate espionage or discrediting an existing application. Monetary gain.
Device theft	A *Webinos*-enabled device is stolen and data is down loaded by the thief.	David	Most likely selling the device, but this could be a targeted attack on an individual or corporation
Webinos malware	A malicious *Webinos* application accesses user data	Frankie	Stealing personal data for personal or monetary gain, may be looking for credentials or credit card details.
Online data leak	A PZH provider exposes their entire file system by mistake. This compromises the certificates, keys and settings of each user	Gary	May be discrediting former employee, or may be looking for recognition from the user community
App content theft	A *Webinos* widget data is stolen by another developer	Jimmy	Monetary gain – copying valuable IPR
Data blackmail	User data is encrypted and the key held by an attacker. The user is extorted to get back their personal data	Ethan	Monetary gain

under control of personal zone provider) and little can be done in case of its violation, information available to the PZP and the WRT are minimized to mitigate information leakage if compromised.

Webinos components of the runtime system allows execution on the base of the application trust, which can be given through certificates signed by a recognised authorities and/or through user authorisation (secured by a previous authentication) at application install or runtime. The only authorised applications shall be from signed, trusted sources, which may be defined by the manufacturer, network provider, or end user.

The *Webinos* runtime must be able to control all application (origin) authenticity and integrity, and this for the firun time before being installed or updated, and *Webinos* must protect the integrity of application instances as they are transferred between devices. Application integrity and authenticity is enforced by the *Webinos* system, in particular the personal zone proxy which acts also as policy enforcement point.

At the same time the system must restrict the application from loading untrustworthy external code, when embedded in a downloaded page (e.g. in a <*script*> tag) The external source must either be an HTTPS location, trusted by the *Webinos* system, or the script must has a signature file linked in the HTML.

Ambiguities in security information must be avoided, i.e. *Webinos* personas shall be able to easily recognize *Webinos* applications running and the authority that certified them and *Webinos* personas should be able to the *Webinos* policy editing tool shall allow policy specification based on assets recognizable by the user and based on comprehensible actions and effects.

The *Webinos* runtime shall protect policies from tampering or modification by unauthorised applications, by exploiting mechanisms to isolate them in memory and in the file system.

Default behaviour should be set to a conservative posture, to avoid common problem of weak configuration left untouched. The installation (or filef use) of an application is the time when a trust decision must be made. By default, unless there are good reasons (based on a conscious decision of the end user), applications should not be

installed. If there is doubt about the provenance of the application—whether it is from the right source and has the right name—it should also not be installed

IDENTITY AND AUTHENTICATION

The problem of user authentication and identity management was identified as one of the most sensitive within the *Webinos* security group. Threats like unauthorised joining of a personal zone, spoofing of PZH page, friend impersonation, and unauthorised enrolment can be mitigated by proper authentication mechanisms.

Since *Webinos* provides for local and remote access, in different domains, and from different platform, the provided identity management mechanisms should be usable by a large variety of users and for a large variety of purposes (i.e. it should be suitable in principle for all end user personas, difficult to trespass for attacker personas, and possible to exploit for developer personas).

Webinos should allow for identification of the user to any *Webinos* device and application while paying attention to privacy issues. (e.g. a blind answer to any identity request would allow for information leakage, thus there is the choice to not respond to any request by any device, at least not before some trust relationship has been established).

The unusual *Webinos* approach requires identification for devices with multiple users on one device (e.g. TV system), thus a one-to-one device-user relationship cannot be always assumed

Many reference personas developed in *Webinos* (e.g. Peter), would distressingly manage multiple authentications and multiple authentication systems. Thus, *Webinos* should expose a single sign-on experience when possible. This problem is even more difficult, as *Webinos* must interact with services on the Web which adopt different authentication schemes (e.g. OAuth), as much seamlessly as possible, while preserving the security of the authentication.

To allow for transparency, the user should know the less of different identities (to not mention of different identity providers) while the appropriate identity required to access services should be always accessible.

To allow for simplicity, existing identities (e.g. Google) should be re-used when possible, but only if appropriate. For example, authentication to a specific device often depends on an identity agreed between the user and the device OS (e.g. finger print, user name/password, etc.), and the device does not need to rely on a third party identity provider to perform authentication. In this case, adopting a third party identities could enable denial of service attacks due to unavailability of the third party identity provider, while not reducing the identities already adopted by the person.

Identity management through certificates and public key infrastructure is also a valuable point, since it allows for a solid framework and for securing communications, with the strong contraindication that almost all *Webinos* reference personas cannot understand the PKI metaphor, due to their technical background.

Final aspect to consider is that authentication is a requirement to solve security problems, but has its own security requirements that must be addressed, in particular is desirable to carry it on a secure channel, to achieve integrity and confidentiality of the exchanged authentication credentials. To challenge these requirements, in *Webinos* we developed five independent authentication mechanisms to handle the combination of users, devices and applications, while hiding complexities and re-using identities. In the next sections we detail these five mechanisms, two related to the users, two to the devices and one to applications.

Users

Users in *Webinos* are primarily identified via an OpenID (OpenID, 2013) account URI. This may be a URL or email address. This identity is used to authenticate the user to the personal zone hub's Web interface and administer the personal zone. How the authentication occurs is up to the OpenID provider. However, on some devices users may forego the OpenID authentication in favour of they device-held private key and certificate. For devices with constrained user interfaces, this prevents users from repeatedly having to enter passwords. Furthermore, allow for re-use of existing identities if so desired by the user.

Users may also authenticate to the underlying platform via a device specific mechanism, wrapped using the *Webinos authentication API*. This allows them to take advantage of any fingerprint readers, screen-locks or other systems. This authentication is only used by the authentication API and to authenticate the user if they want to make changes to the local platform. E.g., enabling or disabling a local service. Also in this case, the choice is to not introduce new identities but instead re-use what already available and known by the user.

Devices

Devices are identified by their public key certificate, issued by the personal zone hub. The private key is stored using a platform-provided key store facility, such as Gnome Keyring on Linux or Keychain Services on OS X. Certificates and keys are used as part of the mutually-authenticated TLS connections established between devices in the personal zone.

Certificates are granted to devices when they are enrolled into the personal zone. Enrolment happens through the user visiting their personal zone hub and obtaining a temporary short authentication token. This token is then used when the device connects over TLS to the hub, who exchanges it for a certificate. The authentication metaphor is here more complex, but it is acceptable since it is performed transparently by the end user that likely could misunderstand and misuse certificates and public key mechanisms.

Applications

Applications are identified by the name they are given in the widget manifest, as well as a set of signatures from various authorities. When running in a widget runtime, they have a *recognised origin* referring to the author's own domain, if it exists. The signature scheme is based on the W3C Widget Signatures standard w3c-widget and recognised origins are described in WAC core specifications (The Wholesale Application Community, 2012).

SESSION MANAGEMENT

A functioning *Webinos* network consists of multiple devices, multiple servers, and multiple applications. It requires the interaction of PZPs and PZHs belonging to different users, over multiple different networks.

In *Webinos* there are many notions of session at different levels, due to the different types of possible communications. In particular,

- *Intra Personal Zone Sessions* that are established among PZPs and PZH for any type of communication, particularly for authentication and exchange of data information. The PZH is permanently addressable on the Internet, and requires that the information exchanged in the session are secure.
- *Synchronization* is special case of intra zone sessions, which is required to seamlessly migrate information among different devices in the personal zone, like user data and identity information, certificates, context events and stream, application data and so on. Since synchronisation touches a number of sensitive objects, it can create major issues if leaked.

- *Inter Personal Zone Sessions* which can be created when accessing services on devices/servers outside the user personal zone (e.g. a service on a friend's device). These sessions (for example, between two applications in two different zones) are mediated by a suitable enforcement point entity in each of the two zones personal zone (for anyone of the two zones, a PZP on the device where the service is provided or is requested, and/or the PZH of the other zone). As special case, it is sometimes necessary to route messages between two PZH's to allow monitored communications between applications and services on different zones

Different sessions for different usage enables threats like eavesdropping on communication between applications, Spoofing of PZH administration pages as well as unauthorized joining of a personal zone and friend impersonation.

Another aspect is that the developer personas we elicited (namely, Jessica and Jimmy) are not incline to pay attention to the security and privacy of the users, unless legally obliged to. Thus, *Webinos* should provide a mechanism both easy to use, and to implement. While lack of care in protecting user security and privacy may have many roots (from time pressure to inexperience), and while a systematic re-education of Web developers would greatly improve user security, still sessions require protection now, since session data may contain private information, and may be even possible to use a hijacked session to impersonate a user on an application. From the other hand, the hijacking or unauthorised disclosure of Web session is feasible even inexperienced attackers, as widely available tools like Firesheep and FaceNiff can show.

In *Webinos* we mitigate this by imposing privacy and authentication protocols to implement the following properties.

- Traffic between zones and services is encrypted to make snooping traffic more difficult.
- Client-server sessions are mutually authenticated. This mutual authentication includes, user and device.
- Traffic can be monitored at the PZH for anomalies. For example sudden changes in IP address, can be challenged be asking the device to re-authenticate. This helps mitigate against real time token stealing attacks.

To achieve this goal we impose a mandatory use of Transport Layer Security (TLS) (Dierks and Rescorla, 2008) and relevant extensions, and to allow this PZPs need to be installed securely on devices PZP installation bootstrap. Once this "Intra Personal Zone Pairing" is done, the established long term trust relationship between that PZP and the PZH can be exploited to build up secure sessions.

DISTRIBUTED ACCESS CONTROL POLICIES

A key security feature provided by *Webinos* is a policy-based access control system. This mediates every attempt by an application, device and user to use a *Webinos* API. This is designed to limit the privileges of applications which may be trusted to access certain features but not others. For example, a 'news' application might request access to geolocation data in order to find the most relevant stories, but does not require the ability to send SMS or use the camera. Following the principle of least privilege has been proven effective in other application frameworks (Felt et al., 2011) and *Webinos* attempts to do the same. We believe a properly developed access control system would limit a number of identified threats, like

unauthorised use of APIs and remote resources, improper use of non-*Webinos* APIs, exploitation of poor access control configuration.

However, the distributed and heterogeneous nature of *Webinos*, along with the ability to form ad-hoc peer-to-peer collaborations creates challenges, as described in previous work (Lyle et al., 2012).

Firstly, the devices in a user's personal zone must have a synchronised and consistent set of policies. Because different devices come online and offline at different times (a car, for example, might not be used for a week or two) it is necessary to aggressively check that the most recent policies are in place before making access control decisions. Although a *default-deny* policy would prevent out-of-date devices from doing too much harm, revocation of application permissions becomes a slower process. Synchronisation is therefore a core requirement of the *Webinos* policy framework.

Another challenge is integrating privacy controls. Rather than simple access control for confidentiality, users of applications may want assurance that their personal data is not misused or disclosed to third parties. While these requirements are ultimately impractical to enforce—Web applications given access to personal data will always be capable of sharing them with remote servers—it would be helpful for users to be informed of how applications *intend* to use their data.

For example, the news application might state that it only uses *geolocation* to filter results, and that this is not shared with any third party. Such statements can then be matched against the user's preferences and a more informed decision can be made about whether to grant the application access to this resource. The challenge that *Webinos* is facing is to integrate security and privacy controls without introducing excessive complexity.

While *Webinos* tries to build on existing work in policy enforcement, another challenge is adapting more limited access control systems to a more complicated environment. The initial plan for *Webinos* was to use the BONDI-defined (bon, 2009) XACML-based (Godik and Moses, 2005) policy enforcement language and framework, but this turned out to need modification. BONDI policies do not support cross-device interaction or multiple users. Policies must also be able to refer to a dynamically changing set of features, as new APIs may be added by new applications.

Finally, the policy framework had to take into account the different level of confidence in user identity that each application has. For example, a mobile device may be assumed to always belong and be used by one person. However, a shared television might regularly be used by several people, including guests. This means that policies need to refer to the authentication level of the user, and may need to request re-authentication before permission can be granted.

The *Webinos* policy framework is based on XACML, but with the reduced vocabulary defined by the BONDI policy engine (bon, 2009). The BONDI approach has also been adapted to allow the *subject* of policies to refer to the device and user. Abstractly, *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.

An example using XACML syntax is shown in Table 3, for user *jessica@example.com*'s mobile device, denying access to the contacts API. Taken from the *Webinos* Consortium (2012a).

Note that policies refer to *features* of APIs rather than APIs themselves. This is because APIs may include both low- and high-risk functions, such as reading SMS versus sending SMS. Policies are defined in the *policy.xml* file on each platform. New policies are created through user actions: when an application is installed, it prompts the user to allow or deny the application installation and to grant access to requested APIs. This approval is translated into XACML and added to the policy file. The policy implementation architecture also follows from XACML, with policy enforcement (PEP), decision (PDP), information (PIP), and access (PAP) points. On *Webinos*,

Table 3. Example of denying access policy

```
<policy-set combine="deny-overrides">
  <policy combine="first-applicable">
    <target>
      <subject>
        <subject-match attr="id" match="appID"/>
        <subject-match attr="version" match="1.0"/>
        <subject-match attr="user-id"
            match="pzh.isp.com/jessica@example.com/Jessica's+Mobile/App"/>
      </subject>
    </target>
    <rule effect="deny">
      <condition combine="or">
        <resource-match attr="api-feature"
          match="http://www.w3.org/ns/api-perms/contacts.read"/>
      </condition>
    </rule>
    <rule effect="permit"> ...</rule>
    <rule effect="deny" />
  </policy>
</policy-set>
```

policy caching is required for requests to remote devices in order to improve efficiency, making the PDP cache (PDPc) a core component.

Synchronisation of policies is implemented through the PZH, which each proxy queries when it starts to receive changes. Each proxy may also make changes to policies (for example, installing a new application) but these require authorisation at the PZH. PZPs also contain an *exceptions.xml* file in the same format.

This file is not synchronised between devices, and is designed to set broad policies that do not need to be changed remotely. This limits the impact of one malicious device attempting to change policies for the whole personal zone.

To describe data handling policies, the XACML-based architecture has been adapted using extensions from the PrimeLife project (Ardagna et al., 2009). This allows *Webinos* to make access control decisions based on both the request context and user preferences. *Webinos* applications contain a *manifest* which let developers give reasons for requesting access to APIs as well as stating obligations with regards to how they will store data. An example excerpt from a manifest is given in Table 4.

THREAT-AWARE API DEVELOPMENT

The *Webinos* platform offers a set of APIs to expose the device and the personal zone's capabilities to applications. These include an *authentication*

Table 4. Privacy-related excerpt of the manifest file

```
<ProvisionalAction>
  <AttributeValue>http://Webinos.org/geolocation</AttributeValue>
  <AttributeValue>#pseudo-analysisDHP</AttributeValue>
  <DeveloperProvidedDescription language="EN">
    The geolocation feature is required by this application in
    order to customise search results.
  </DeveloperProvidedDescription>
</ProvisionalAction>
```

API, which provides to authorized applications the current authentication status of users, and may ask the runtime to (re)authenticate the user; *discovery* which allow applications to discover services without any previous knowledge of the service, and many others (the complete list and description is available in the official deliverable (The *Webinos* Consortium, 2012b).

Each API may have unique security and privacy concerns. Each must be considered in turn and any API-specific threats need to be mitigated. This process also helps to mitigate one of the bigger threats identified in section 4: the exploitation of the underlying platform through misuse of *Webinos* APIs. To approach this problem, we performed a risk analysis process shared among several partners of the project. Selected parties reviewed each API, either due to their security expertise or their involvement with the development of the API and therefore insights into potential threats. Most analysis data was reviewed by another partner before it was considered finished. Creativity was encouraged as part of this exercise, as well as other suitable data sources, like The Mozilla WebAPI security analysis (Mozilla, 2012) and, since many APIs derive from previous efforts by the W3C and WAC, their original considerations on security and privacy.

The findings of this process were conveyed as a series of recommendations and reported to the API developers to inform and modify the API (which within the project is an iterative process). Since API developers were often aware of performance issues, but more rarely of the security implications, this analysis allows for a threat-aware development.

Analysts structured their feedback in a manner that was quick to read, following a specifically developed template. The template suggested that analysts consider various personas, data sources and attack vectors. More specifically, the template allowed obvious threats that misuse of this API could cause to users to be highlighted, taking into account the assets the API gives access to, as well as what happens if the API is excessively used. This considered several *Webinos* personas and took advantage of any persona-specific security and privacy consideration (developed in *Webinos* in parallel to the specification development).

The template has a specific section for threats based on remote invocation of the API (i.e., when called from another device and/or by another user, what are the additional security concerns?). It also allowed analysts with implementation experience to describe their concerns. Developer-specific threats were considered through use of the two developer personas. For instance, analysts were asked how a developer or their application might be caused harm if they used or relied on a certain AP. Finally, threats to device manufacturers, operators, and other stakeholders were also considered: e.g. excessive bandwidth consumption for telecommunications operators.

As a result of the threat analysis several mitigations were recommended. These primarily involved setting default permissions or identifying excessive use of an API, but others were also suggested.

Example Analysis

As an example, we consider the Context API (other API's threat analyses are described in the *Webinos* official deliverable). This API allows access to a user's context data through either explicit queries or a subscription model. Context events include all API calls. Specific threats identified include privacy, confidentiality and non-repudiation topics:

- Any application can monitor what the user is doing and has done, and can react to particular events. This might be used for targeted adverts, physical or cyber stalking, targeted theft or burglary and identity theft.
- Any application can potentially see which files have been opened, where the user has been, contact information, and so on. Depending on implementation, this could

include the content of files and more. An application might use this to gain access to APIs and resources it does not have permission for.

- Users may want to go unmonitored and need to turn off context collection at times. If this is hard to do or unclear, it might result in embarrassment or a loss of reputation.

No specific threats have been identified for remote invocation of this API, because this API primarily uses a central database.

From implementation experience, two availability risks were raised: subscribing to common context events might slow down the platform considerably, making it unusable, and too many queries to the context API might over-use bandwidth and cause either a loss of battery power of expensive mobile phone bills.

Developers using this API should be aware that context data could be inconsistent or misused, thus provide a false impression for developers. For example, if a user turns on and off their context data, it may make them appear to have different behaviour to reality.

Finally, from other stakeholders' point of view, too many queries to the context API might over-use bandwidth and reduce battery life.

Mitigations suggested varied from conservative prevention, turning off context collection by default and providing controls for turning on/off collection and clearing the database, to allowing for fine grained controls and deletion of data (suitable for users aware of what context aspect make available). To enable awareness of the API usage, the system should provide feedback for when applications query context data. Another suggestion was the creation of a figr-grained taxonomy of context data, to allow strict integration with the access control system, and allowing for a bare minimum collection of information about API calls.

Finally, as default policy, access to this API should be denied by default to trusted sources (this allow user to be aware when the requests are done) and always denied to untrusted (unknown) sources.

Analysis Results

The API threat analysis highlighted the fact that APIs should be more privacy aware, e.g. the Discovery API is privacy-invasive due to its use of persistent identifiers for *Webinos* services; this facilitates user fingerprinting, and is unnecessary for "impersonal" services, while the Calendar API and the Messaging API expose a great deal of information to requesting applications, even if not always required. e.g. for Calendar, an application which requires only the free/busy status of a user will still be given details of every calendar entry. Privacy friendliness could be achieved by a finer grained detail of the API interface (so application can obtain only the interface which exposes the minimum required user information) and the use of an Intent-style approach where the application is unaware of the service fulfilling a request.

It is also useful to make a clear distinction between types of application: 'system level' and 'Web level'. These have different potential security levels, and address the two types of use case considered within *Webinos*. On the one hand, applications requiring only slightly more functionality than that provided by the Web browser. On the other, system-level applications which are more trusted and need to have greater access to device features. Web applications could be executed in a normal Web browser and should be given access to a limited set of APIs in a more privacy-friendly way (e.g. discovery). This is because Web applications are vulnerable to a wide range of attacks which are difficult to mediate at the API level. By contrast, system level applications will run in a secure execution environment (a widget runtime) which protects them from at-

tacks by plugins, or through cross-site scripting. They have the potential to access more APIs and may have fewer privacy constraints. These apps should only be installed from app stores and must be pre-vetted, but when they are installed they are able to do much more.

SECURE STORAGE

The *Webinos* platform uses and stores data which may have security and privacy requirements. For example, many of the personas may use applications to monitor health data, to read personal emails, or to store valuable work fipla. As such, it is important to address threats and mitigations to vulnerabilities affecting data at rest.

Because most platforms already provide some mitigation to attacks on stored data, *Webinos* does not attempt to solve every problem. Table 5 shows how the threats from section 4 may be mitigated. In the case of Application content theft, we do not attempt to provide a DRM solution but expect that one could be implemented over the top of *Webinos*.

DISCUSSION

We sum up now a set of principles the lessons learned from two years of work in developing the security framework.

Tablets, smartphones, laptops, and cars are all designed to be mobile. This significantly increases the risk of a device being lost or stolen. Sensitive data should either not be stored in mobile devices or protected by secure storage available on the devices. More sensibly, revocation must be primarily concerned with removing a lost and potentially rogue device from the personal network, recalling the security inclination of the potential users (for many *Webinos* personas revocation must be a one-click process, and must also not rely on the user having another enrolled device to hand).

The interoperability of Web applications—which provide a common, accessible Web server for communication—should be re-used to make personal networks available to any device capable of making outgoing connections to Web servers. Mechanisms which requires an always-on connection (e.g. for authentication, synchronization, PKI management) would greatly benefit of this availability, while the management cost would be outsourced to the Internet infrastructure and to the provider. We believe this is an enabler for a efficient scaling of complex security solutions (like PKI) on mobile networks with frequently changing IP addresses.

It is better to delegate the tasks to the underlying OS when a security mechanisms is platform specific and low-level. It is time consuming to design an application middleware, which is capable of interacting with all low-level security

Table 5. Examples of suggested mitigation for data-at-rest's reference threats

Threat	Mitigation
Native malware	Provide anti-malware tools and allocate each native application in its own private storage area
Device theft	Provide full disk encryption
Webinos malware	*Webinos* will provide isolated storage for each application, require additional permissions to access shared areas
Online data leak	The PZH can be designed to minimize risk by storing as little data as possible. PZH providers should provide disk encryption and should follow best practice guidelines to avoid vulnerabilities. Keys should be stored privately using either a trusted hardware device or a separate fipr system.
App content theft	No mitigation
Data blackmail	Offer backup and recovery tools.

mechanisms on all platforms. Each platform has different application security infrastructures, so protection from malware is hard to achieve in a truly cross-platform manner.

Furthermore, the underlying OS can use a device-specific solution, with the advantage that this should be well tested by frequent use. For example, the best way to protect private keys is likely to be device-specific. Some devices support secure hardware which may provide a high level of protection. Furthermore, devices in different contexts will have different authentication requirements: e.g., a shared PC might only unlock private keys after authenticating the end user, whereas a mobile device may be assumed to belong to one person only.

The use of secure hardware capabilities should be the object of further investigation inside *Webinos*. It would be useful because hosted applications may be running on insecure remote platforms and this could be assessed through use of attestation on the host and verification on the user's device (Lyle, 2010). If the host is found to be running an untrustworthy configuration then the application may not be installed, or if the host changes configuration it could result in a new assessment.

The emerging cross-platform scenarios are young and admit misinterpretation as the security implications of these scenarios are unclear. For example, device keys are not always a factor of user authentication, this is because personal devices are designed to be mobile, thus is more secure to use a device key to identify the device only, and only as a second factor when the device is appropriate: e.g., a laptop or mobile phone with a single user and a login prompt. This contrasts with some literature (e.g. Balfanz et al., 2005), which identifies that certificates could be treated like capabilities, but is corroborated by some threat scenarios we investigated, e.g. when a device is stolen and then access to core security zone management is admitted only on the basis of the private key.

CONCLUSION AND FUTURE WORK

In this paper we presented the design of *Webinos*, providing insights into the motivation behind security choices and the overall architecture, as well as the literature which inspired our work

We used *personas* in order to support usable security design. Most of the *Webinos* personas we expect to already have identities on the Web (e.g. from social networks, email accounts, and homepages). Some were tech-savvy, and thus not comfortable with having multiple online accounts in different contexts. A mapping from Web-based identity to a public key or certificate is therefore a good way to allow users to find each other through acceptable Web identities (obtained by a combination of user and device as suggested by (Ford et al., 2006)), while benefitting from the stronger security guarantees inherent in a public key infrastructure. This leverages existing relationships and avoids the discovery and bootstrapping problems often associated with PKI (Balfanz et al., 2005). It also minimizes the need for passwords. Since PKI terminology must not be exposed to end users, all keys and certificates should be generated automatically, and there should never be a prompt or question asked to users referring to these things. However, even if we conceal details of the authentication and identity management *Webinos* system, we believe that further research is needed in this field, especially to develop empirical evidence of the usefulness of our proposed approach.

We also plan further studies on authentication and identity classification, as well as investigations into stronger integration with social networks. For example, adopting a social network reputation and review system could be a way to reduce reliance on the public key infrastructure for *application* security. Application certificates are one source of information on trustworthiness, but social networks may provide more useful information, e.g. if 90% of the user's friends rate an application highly, this information may help the user

decide whether to trust the application or not. On a similar theme, recommendations from particular users might trigger policy settings which allow the application to be installed with minimal authorisation. In this way, *Webinos* could present to the end user a comfortable security model, since similar to today's social network model, and thus could enhance acceptability, avoid misinterpretation and enable user's security-wise behaviour.

ACKNOWLEDGMENT

The research described in this chapter was funded by EU FP7 *Webinos* Project (FP7-ICT-2009-5 Objective 1.2). We thank all our project partners as this chapter draws upon their work.

REFERENCES

Android. (2012). *Developer guide: Security and permissions, October 2012*. Retrieved from http://developer.android.com/guide/topics/security/security.html

Ardagna, C. A., di Vimercati, S. D. C., Paraboschi, S., Pedrini, E., & Samarati, P. (2009). An XACML-based privacy-centered access control system. In *Proceedings of the First ACM Workshop on Information Security Governance, WISG '09*, (pp. 49–58). ACM.

Atzeni, A., Cameroni, C., Faily, S., Lyle, J., & Fléchais, I. (2011). Here's Johnny: A methodology for developing attacker personas. In *Proceedings of the 6th International Conference on Availability, Reliability and Security*, (pp. 722–727). IEEE.

Balfanz, D., Durfee, G., & Smetters, D. (2005). Making the impossible easy: Usable PKI. In *Security and Usability: Designing Secure Systems that People Can Use* (pp. 319–334). Sebastopol, CA: O'Reilly.

BONDI. (n.d.). *Architecture and security requirements appendices*. Retrieved from http://bondi.omtp.org/1.01/security/BONDI_Architecture_and_Security_Appendices_v1_01.pdf

Cooper, A. (1999). *The inmates are running the asylum: Why high tech products drive us crazy and how to restore the sanity* (2nd ed.). Upper Saddle River, NJ: Pearson Higher Education.

Dierks, T., & Rescorla, E. (2008). *The transport layer security (TLS) protocol version 1.2*. RFC 5246 (Proposed Standard). Retrieved from http://www.ietf.org/rfc/rfc5246.txt

Faily, S., Coles-Kemp, L., Dunphy, P., Just, M., Akama, Y., & De Luca, A. (2013). Designing interactive secure systems: CHI 2013 special interest group. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*. ACM. doi:10.1145/2468356.2468807.

Faily, S., & Fléchais, I. (2010a). A meta-model for usable secure requirements engineering. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, (pp. 29–35). ACM.

Faily, S., & Fléchais, I. (2010b). Barry is not the weakest link: Eliciting secure system requirements with personas. In *Proceedings of the 24th BCS Interaction Specialist Group Conference, BCS '10*, (pp. 124-132). London: British Computer Society.

Faily, S., & Fléchais, I. (2011). Eliciting usable security requirements with misusability cases. In *Proceedings of the 19th IEEE International Requirements Engineering Conference*, (pp. 339–340). IEEE Computer Society.

Faily, S., Lyle, J., & Parkin, S. (2012). Secure system? Challenge accepted: Finding and resolving security failures using security premortems. In *Proceedings of Designing Interactive Secure Systems: Workshop at British HCI 2012*. London: HCI.

Felt, A. P., Greenwood, K., & Wagner, D. (2011). The effectiveness of application permissions. In *Proceedings of the 2nd USENIX Conference on Web Application Development, WebApps'11*. Berkeley, CA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=2002168.2002175

Ford, B., Strauss, J., Lesniewski-Laas, C., Rhea, S., Kaashoek, F., & Morris, R. (2006). Persistent personal names for globally connected mobile devices. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI '06*, (pp. 233–248). Berkeley, CA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm

Godik, S., & Moses, T. (2005). *Extensible access control markup language (XACML) version 1.1, May 2005*. Retrieved from http://www.oasis-open.org

iOS. (n.d.). *Developer library: iOS technology overview introduction*. Retrieved from http://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html

Joyent, Inc. (2012). *Nodejs website*. Retrieved from http://nodejs.org/

Kinkelin, H., Holz, R., Niedermayer, H., Mittelberger, S., & Carle, G. (2011). On using TPM for secure identities in future home networks. *Future Internet, 3*(1), 1–13. Retrieved from http://www.mdpi.com/1999-5903/3/1/1/

Lyle, J. (2010). *Trustable services through attestation*. (PhD thesis). University of Oxford, Oxford, UK. Retrieved from http://www.cs.ox.ac.uk/people/John.Lyle/thesis-final-25-06-11.pdf

Lyle, J., Monteleone, S., Faily, S., Patti, D., & Ricciato, F. (2012). Cross-platform access control for mobile web applications. In *Proceedings of the IEEE International Symposium on Policies for Distributed Systems & Networks*. IEEE.

Mac. (n.d.a). *OS X developer library: Security architecture*. Retrieved from http://developer.apple.com/library/mac/documentation/Security/Conceptual/Security_Overview/Introduction/Introduction.html

Mac. (n.d.b). *OS X developer library: Security services*. Retrieved from http://developer.apple.com/library/mac/documentation/Security/Conceptual/Security_Overview/Security_Services/Security_Services.html

Mitchell, C.J., & Schaffelhofer, R. (2004). *Security for mobility: The personal PKI*. Institution of Engineering and Technology.

MITRE Corporation. (n.d.). *Common attack pattern enumeration and classification (CAPEC)*. Retrieved from http://capec.mitre.org/

Mozilla. (n.d.). *Boot to gecko project website*. Retrieved from http://www.mozilla.org/en-US/b2g/

Niemegeers, I.G., & de Groot, S.M.H. (2002). From personal area networks to personal networks: A user oriented approach. *Wireless Personal Communications, 22*, 175–186. Retrieved fromhttp://dx.doi.org/10.1023/A:1019912421877

Open, I. D. (n.d.). *What is OpenId?* Retrieved from http://openid.net/get-an-openid/what-is-openid/

OWASP Foundation. (n.d.). *The open web application security project*. Retrieved from http://www.owasp.org/index.php/

Pruitt, J., & Adlin, T. (2006). *The persona lifecycle: Keeping people in mind throughout product design*. Amsterdam: Elsevier.

Sasse, M. A., Brostoff, S., & Weirich, D. (2001). Transforming the 'weakest link': A human-computer interaction approach to usable and effective security. *BT Technology Journal*, *19*, 122–131. doi:10.1023/A:1011902718709.

Schneier, B. (2009). *Security versus usability*. Retrieved from http://www.schneier.com/blog/archives/2009/08/security_vs_usa.html

SHAMAN Project. (2002). *Deliverable 13, work package 3, November 2002*. Retrieved from http://www.isrc.rhul.ac.uk/shaman/docs/d13a3v1.pdf

UPnP Forum. (2011). *UPnP device protection service* (Technical report). Retrieved from http://upnp.org/specs/gw/deviceprotection1. w3c-widget

Webinos Consortium. (2011). *User expectations on security and privacy phase 1, February 2011*. Retrieved from http://Webinos.org/content/Webinos-User_Expectations_on_Security_and_Privacy_v1.pdf

Webinos Consortium. (2012a). *Phase 2 platform Specifications: Policy*. Retrieved from http://webinos.org

Webinos Consortium. (2012b). *Phase 2 API specification*. Retrieved from http://Webinos.org/blog/2012/09/24/Webinos-report-phase-ii-api-specifications

Webinos Consortium. (2012c). *The webinos project website*. Retrieved from http://Webinos.org/

Webinos Consortium. (2012d). *Updates on scenarios and use-cases*. Retrieved from http://www.webinos.org/wp-content/uploads/2012/06/D2.4-Updates-on-Scenarios-and-Use-Cases.public.pdf

Whitten, A., & Tygar, J. D. (1999). Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th Conference on USENIX Security Symposium*, (pp. 169–184). Berkeley, CA: USENIX Association.

Wholesale Application Community. (2012). *WAC core specifications 2.1: Security and privacy*. Retrieved from http://specs.wacapps.net/core/

World Wide Web Consortium. (W3C). (2011, August 11). *XML digital signatures for widgets, W3C proposed recommendation*. Retrieved from http://www.w3.org/TR/2011/PR-widgets-digsig-20110811/